České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra počítačů

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Pavel Matyáš**

Studijní program: Otevřená informatika
Obor: Softwarové systémy

Název tématu: **Servisně orientovaný aspektový vývoj uživatelských rozhraní pro mobilní aplikace**

Pokyny pro vypracování:

Prostudujte použití knihovny AspectFaces pro aspektově orientovaný vývoj uživatelských rozhraní pro Java EE aplikace [1,2]. Prostudujte možnosti využití této knihovny pro servisně orientovaný aspektový vývoj aplikací [3]. Navrhněte řešení pro mobilní platformu. Řešení demonstrujte na konkrétní aplikaci na dvou různých mobilních prostředích. Porovnejte vaše navržené řešení s AspectFaces. Porovnejte možnosti vytvoření UI bez použití navrhovaného řešení.

Seznam odborné literatury:

[1] CERNY, T. _ CHALUPA, V. _ DONAHOO, M. Towards Smart User Interface Design. In Information Science and Applications (ICISA), 2012 International Conference on, s. 1_6, May 2012. doi: 10.1109/ICISA.2012.6220929.
[2] CERNY, T. et al. Aspect-driven, Data-reflective and Context-aware User Interfaces Design. SIGAPP Appl. Comput. Rev. December 2013, 13, 4, s. 53_66. ISSN 1559-6915. doi: 10.1145/2577554.2577561.
[3] TOMÁŠEK, M. and T. ČERNÝ. On Web Services UI In User Interface Generation in Standalone Applications. In: Proceeding of the 2015 Research in Adaptive and Convergent Systems (RACS 2015). Research in Adaptive and Convergent Systems, Prague, 2015-10-09/2015-10-12. New York: ACM, 2015, pp. 363-368. ISBN 978-1-4503-3738-0.

Vedoucí: Ing. Martin Tomášek

Platnost zadání: do konce letního semestru 2016/2017

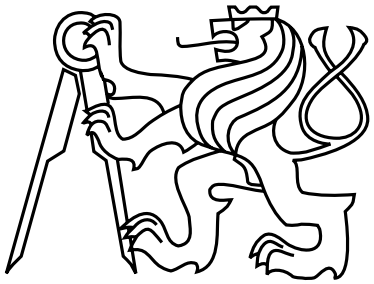
[redacted]
prof. Ing. Filip Železný, Ph.D.
vedoucí katedry

L.S.

[redacted]
prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 15. 1. 2016

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů



Bakalářská práce

Servisně orientovaný aspektový vývoj uživatelských rozhraní pro mobilní aplikace

Pavel Matyáš

Vedoucí práce: Ing. Martin Tomášek

Studijní program: Otevřená informatika, Bakalářský

Obor: Softwarové inženýrství

8. dubna 2016

Poděkování

Zde můžete napsat své poděkování, pokud chcete a máte komu děkovat.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Kořenovicích nad Bečvárkou dne 15. 5. 2008 .....

Abstract

Translation of Czech abstract into English.

Abstrakt

Abstrakt práce by měl velmi stručně vystihovat její obsah. Tedy čím se práce zabývá a co je jejím výsledkem/přínosem.
Očekávají se cca 1 – 2 odstavce, maximálně půl stránky.

Obsah

1 Úvod **1**
1.1 Motivace . . . 1

2 Popis problému a specifikace cíle **3**
2.1 Popis problematiky . . . 3
2.1.1 Různá uživatelská rozhraní . . . 3
2.1.2 Tvorba uživatelského rozhraní . . . 4
2.1.3 Využití webových služeb pro zisk a odeslání dat . . . 5
2.1.4 Existující řešení . . . 6
2.1.4.1 PHP Database Form . . . 7
2.1.4.2 AspectFaces . . . 7
2.1.4.3 AFSwinx a AFRest . . . 7
2.1.5 Cíle práce . . . 8

3 Analýza **9**
3.1 Funkční specifikace . . . 9
3.1.1 Funkční požadavky . . . 9
3.2 Popis architektury a komunikace . . . 10
3.2.1 Definice komponent . . . 10
3.2.2 Získání definice ze serveru . . . 11
3.2.3 Reprezentace metadat ve frameworku . . . 12
3.2.3.1 AFClassInfo . . . 12
3.2.3.2 AFFieldInfo . . . 13
3.2.3.3 AFValidationRule . . . 14
3.2.3.4 AFOptions . . . 14
3.2.3.5 TopLevelLayout . . . 15
3.2.3.6 Layout . . . 15
3.2.4 Tvorba komponent . . . 15
3.2.4.1 AFComponent . . . 16
3.2.4.2 AFField . . . 16
3.2.4.3 AFComponentBuilder . . . 17
3.2.4.4 RequestMaker a JSONParser . . . 17
3.2.4.5 FieldBuilder a WidgetBuilder . . . 17
3.2.4.6 Lokalizace . . . 18
3.2.5 Práce s vytvořenou komponentou . . . 18

3.3 Případy užití 18
3.3.1 Případ užití: Validace formuláře 19
3.3.2 Případ užití: Vygenerování odesílaných dat 20
3.3.3 Případ užití: Odeslání dat na server 21
3.3.4 Úprava vzhledu komponenty 21
3.4 Práce na existujícím řešení 22

A Seznam použitých zkratek **25**

B UML diagramy a obrázky **27**

C Obsah přiloženého CD **31**

Seznam obrázků

3.1 Doménový model objektů obsahující metadata o komponentě 13
3.2 Část doménového modelu systému pro validaci 14
3.3 Část doménového modelu systému realizující tvorbu komponenty 16
3.4 Diagram případů užití pro odeslání formuláře 19

B.1 Diagram aktivit popisující proces tvorby formuláře 28
B.2 Diagram aktivit popisující práci s formulářem 29
B.3 Model případů užití frameworku 30

C.1 Seznam přiloženého CD — příklad 31

Seznam tabulek

Kapitola 1

Úvod

Tato bakalářská práce se zabývá servisně orientovaným generováním uživatelského rozhraní pro mobilní zařízení s využitím knihovny AspectFaces.

První část práce popisuje aktuální situaci v tvorbě UI, specifikuje požadavky a cíle práce a zkoumá již existující řešení. V druhé části se pak analyzují požadavky, které by měla práce splňovat a návrh řešení, které stanovené požadavky a cíle splňuje. Ve třetí části je popsána struktura a vlastní implementace řešení. Poslední část pak obsahuje otestování řešení a ukazuje vzorovou aplikaci na dvou různých mobilních prostředích.

ZMENIT PO DODELANI Práce obsahuje seznam použitých zkratek, které lze najít v příloze A, použité UML diagramy a obrázky viz. příloha C a zdrojové kódy aplikace, které jsou přiloženy na CD, obsah tohoto cd je v příloze D.

1.1 Motivace

Nedílnou součástí většíny dnešních aplikací je uživatelské rozhraní. Uživatelské rozhraní by mělo uživateli co nejvíce usnadňovat manipulaci se softwarem a tudíž být intuitivní a použitelné, nemluvě o tom, že by mělo pěkně vypadat. Vývoj takového rozhraní je však časově velmi náročný proces, který zahrnuje nejenom samotný vývoj, ale také rozsáhlé testování, hlavně z hlediska funkčnosti a použitelnosti. Celý problém navíc umocňuje fakt, že se vývojáři snaží zajistit podporu software na více platformách, neboť chtějí uživatelům nabídnout možnost operovat s jejich vytvořeným systémem nejen z počítače či laptopu, ale také z tabletu nebo mobilního zařízení. Mobilní verze grafického uživatelského rozhraní nebývá často moc rozdílná od rozhraní ostatních platforem v tom smyslu, že se v ní vyskytují vesměs stejné grafické prvky, a tak pro mobilní verzi vzniká téměř indentická kopie tohoto rozhraní, čímž vzniká duplicita. Problémem je, že se často uživatelská rozhraní mění, ať už se změna týká rozložení komponent, přidání nebo odebrání komponenty nebo třeba validace uživatelského vstupu, protože takováto změna se pak musí provést na všech platformách a někdy i na více místech v rámci aplikace, což může být v případě rozsáhlých systémů nejednoduchý úkol, který stojí vývojáře spoustu zbytečného času. Pokud bychom byli schopni nadefinovat uživatelské rozhraní jen jednou pro všechny platformy na jednom místě, tento problém bychom odstranili. Definice by tedy byla obecná, ale každá platforma je jiná a něčím

specifická, proto je třeba vytvořit pro různé platformy frameworky, které obecnou definici pro danou platformu interpretují.

Bylo mi nabítnuto vytvořit takovýto framework pro dvě mobilní platformy, kontrétně pro Android a Windows Phone, což mi přislo velmi užitečné a zajímavé a proto jsem se rozhodl zpracovat toto téma jako bakalářskou práci.

Kapitola 2

Popis problému a specifikace cíle

2.1 Popis problematiky

Softwarový systém má sloužit člověku k řešení problémů. Uživatel přitom často problém blíže specifikuje a systém musí mít způsob, jak uživateli sdělit jeho řešení. K tomu slouží uživatelská rozhraní, která umožňují vzájemnou komunikaci systému s uživatelem. Takové uživatelské rozhraní by mělo v prní řadě sloužit uživateli, to znamená umožnit mu jednoduchou interakci se systémem, být intuitivní, funkční a hlavně použitelné.

2.1.1 Různá uživatelská rozhraní

Aby bylo uživatelské rozhraní použitelné a uživatelsky přívětivé je třeba prozkoumat, jakým způsobem člověk s počítačem a jeho aplikacemi spolupracuje. Nejen tímto se zabývá disciplína zvaná Human Computer Interaction, která zkoumá potřeby uživatelů z různorodých hledisek. Studium uživatelských potřeb vedlo ke vzniku různých uživatelských rozhraní, pomocí kterých může člověk s počítačem komunikovat. Jedním z takových rozhraní je textové uživatelské rozhraní, značené CUI, jehož typickým zástupcem je příkazová řádka. Dalším typem je Hlasové uživatelské rozhraní, které dokáže interpretovat povely zadané lidskou řečí. Nezanedbatelným zástupcem je taktéž multimodální rozhraní, které používá k interakci s počítačem více lidských smyslů, a tak je vhodné i pro lidi s postižením. Nejrozšířenějším a nejoblíbenějším rozhraním je však grafické uživatelské rozhraní, zkráceně GUI, protože je jednoduché a grafické prvky v člověku vyvolávají podobnost s vnějším světem, čímž uživatel získává pocit, že pracuje s něčím, co už dávno zná. Také není nutné znát žádně specifické příkazy, jako v případě příkazové řádky, nebo hlasové povely jako v případě hlasového rozhraní. [9].

V softwarových systémech je nejběžnějsím způsobem interakce uživatele se systémem právě grafické uživatelské rozhraní a tím se taky tato práce bude zabývat.

Běžně GUI disponuje ovládacími prvky, pomocí kterých lze aplikaci ovládat. V mobilních aplikacích jsou ovládacími prvky nejčastěji tlačítka, menu, formuláře, posuvníky či seznamy položek. Formulář je skupina vstupních polí, která zachycují uživatelský vstup neboli grafické prvky, které umožní uživateli zadat text, zaškrtnout či vybrat z více možností, vybrat datum atd. Formulář a seznam položek může být aplikací využit také k zobrazení svého výstupu,

respektive aktuálního stavu systému, který musí být uživateli k dispozici, neboť účelem GUI je i mimo jiné informovat uživatele o výsledku jeho akcí a dopadu akcí na systém. K zobrazení informací uživatelovi dále slouží statické texty, tabulky, dialogy.

Návrh GUI je potřeba důkladně zvážit, neboť závisí na mnoha aspektech. Důležité je, pro jaké zařízení je GUI tvořeno, jaký účel má aplikace, která bude rozhraním disponovat a stav uživatele a prostředí, ve kterém se nachází.

Typ zařízení je důležitý hlavně proto, protože každé zařízení používá jiné ovládací prvky. Zatímco u mobilního zařízení lze očekávat použití dotykového displeje, u počítače zase použití myši, klávesnice nebo i jiných externích vstupních zařízení, například grafický tablet. Taky se zařízení liší ve velikosti displeje a rozlišení, což hraje roli zejména při návrhu GUI z hlediska množství, velikosti a rozložení komponent.

Účel aplikace ovlivňuje GUI hlavně z hlediska obsahu, tedy jaké komponenty je nutné mít, aby byla aplikace využívána k daném účelu. Například emailový klient musí obsahovat ovládací prvek, který odešle zprávu.

Stav uživatele a prostředí může zase ovlivnit způsob ovládání aplikace. Příkladem může být palubní počítač v automobilu, na kterém by měl uživatel být schopen přepnout rádiovou stanici, aniž by se přestal věnovat řízení.

2.1.2 Tvorba uživatelského rozhraní

Je známo, že vývojáři vkládají do tvorby uživatelského rozhraní velké úsilí a značné množství času, což dokazuje i zjištění, že uživatelské rozhraní zabírá přibližně 48% kódu aplikace a zhruba 50% času, který je vývoji aplikace věnován [15]. Další čas a úsilí také zabere testování rozhraní hlavně z hlediska použitelnosti, které opět stojí spoustu času i nákladů. Vývojář mnohdy nedokáže odhadnout chování cílové skupiny, která systém bude používat, a tak se často dělají testy s koncovým uživatelem, u kterých se zkoumá, jak uživatel software ovládá. Z těchto testů se často odhalí, že uživatelské rozhraní je nedostačující a neposkytuje uživateli potřebný komfort při ovládání systému. Velkým problémem je uživatelský vstup, protože musí být validován, aby uživatel nevložil data, která jsou v rozporu s modelem, na který je rozhraní namapováno [17]. Také je žádoucí zobrazovat uživateli pouze to, co by vidět měl, například na základě jeho uživatelské role v systému. V neposlední řadě je také podstatné, jak rozhraní vypadá. Důležitým aspektem rozhraní je, jakým způsobem jsou v něm reprezentována data a jak jsou uspořádány jeho jednotlivé části. Z výše uvedeného lze vidět, že je tvorba uživatelského rozhraní opravdu náročný a rozsáhlý proces a právě proto je poskytovat pro systém více verzí uživatelských rozhraní, například pro různé platformy nebo pro různé uživatelské role, obtížný úkol [17].

Jedním z hlavních a kritických aspektů dobrého softwaru jeho udržovatelnost, anglicky maintainability. Udržovatelnost je schopnost systému se dále měnit a vyjívet na základě požadavků zákazníka. Změny by přitom měly být lehce proveditelné a neměly by nijak výrazně ovlivnit stav systému. Požadavky na změnu lze očekávat vždy, neboť potřeby zákazníků se neustále mění [19]. Bohužel uživatelské rozhraní moc udržovatelné není, což ukáže následující příklad.

Mějme například desktopovou a mobilní aplikaci, které obě obsahují formulář namapovaný na určitou entitu v databázovém modelu. Tento model se nějak změní, například v dané entitě rozdělíme jeden sloupec na dva. Bohužel neexistuje žádný machanismus, který

by automaticky zaručil, že je UI v souladu s modelem [17]. Z pohledu vývojáře to pak znamená, že pokud změní databázový model, musí také změnit uživatelské rozhraní v obou klientských aplikacích, aby korespondovalo s novým databázovým modelem. Zde nejenom, že musí vývojář udělat dvakrát stejnou věc, ale také může udělat chybu, což může vyústit v nefunkčnost systému. Také pokud se takový formulář vyskytuje třeba na pěti místech v aplikaci, změna je už časově náročnější, hůře proveditelná a ještě více náchylná na chybu vývojáře, který může nějaký výskyt formuláře opomenout. Takovým zásahem do systému nemusí být jen změna databázového modelu, ale také změna validací uživatelského vstupu nebo změna rozložení či pořadí jednotlivých polí ve formuláři.

2.1.3 Využití webových služeb pro zisk a odeslání dat

Jak už bylo zmíněno, v grafickém uživatelském rozhraní máme výstupní grafické prvky, jako například tabulky či seznamy položek. Tyto komponenty jsou určeny v první řadě k tomu, aby zobrazovaly uživateli soubor dat. Existuje více možností, kde tato data skladovat a odtud je získávat a prezentovat je uživateli. Jednou z možností je, že má aplikace vlastní databázi. Taková aplikace není určena k tomu, aby komunikovala nebo sdílela data s dalšími instancemi této aplikace na jiných zařízeních. Pokud je komunikace vyžadována, je vhodná architektura klient-server. Server může mít vlastní databázi, ze které poskytuje klientům informace například prostřednictvím webových služeb. Webová služba umožňuje jednomu zařízení interakci s jiným zařízením prostřednictvím sítě[10]. V tomto případě je jedním zařízením server, druhým klienstká aplikace a interakcí je myšlen vzájemný přenos dat.

V mobilních aplikacích jsou velmi populární interpretací webových služeb RESTful Web Services využívající Representional State Transfer (REST), který byl navržen tak, aby získával data ze zdrojů pomocí jednotných identifikátorů zdrojů (URI), což jsou typicky odkazy na webu. Využívá se právě v aplikacích s klient-server architekturou a ke komunikaci používá HTTP protokol, jehož výhodou je, že jeho metody poskytují jednotné rozhraní pro manipulaci se zdroji informací poskytovanými webovou službou. Http metoda PUT se využívá k vytvoření nového zdroje, DELETE zdroj maže, GET se používá pro získání aktuálního stavu zdroje v nějaké dané reprezentaci a POST stav zdroje upravuje [6].

Aby klient mohl data z webové služby získat a následně je reprezentovat v UI, musí znát jejich strukturu, formát dat, metodu, kterou musí použít a dodatečné parametry, kterými lze službu nastavit. Data jsou ve spojitosti s RESTful službami nejčastěji přenášena ve formě XML nebo alternativně ve formě JSON[18], což jsou formáty obsahující data ve formě párů klíč - hodnota. Zmíněné formy dat vzikají serializací objektů [8], jejichž definici lze většinou získat z dokumentace poskytovatele webové služby, stejně tak jako další potřebné výše zmíněné věci.

Přijatá data lze na klientovi zpracovat více způsoby. Jednou z možností je napsat si vlastní parser, což si lze usnadnit nějakou knihovnou, která s formátem umí pracovat. Pro formát JSON v Javě existuje například knihovna org.json [5]. Dalším způsobem je využití nějaké knihovny, která umí data rovnou deserializovat do objektu. Příkladem takové knihovny je Gson[4].

Obdobně webové služby fungují i v případě odesílání dat. Zdroj webové služby definuje v jakém formátu data přijme a v dokumentaci lze opět nalézt definici objektu, do kterého se bude snažit data deserializovat, respektive přijatá data vložit.

Z uvedeného plyne, že se klientská aplikace musí v obou situacích, jak při zisku, tak při odesílání dat, adaptovat na určitou, předem danou strukturu dat. Pokud se tato struktura změní, tedy zění se objekt ze kterého serializací data vznikají a deserializují se [8] do něj odeslaná data, je nutné upravit i příslušná místa v klienstké aplikaci, která zpracování a odesílání uživatelského vstupu mají nastarosti.

Demostrujeme problém na příkladě. Máme server a na něm model naříklad Tým, který obsahuje dva sloupce - název týmu a počet členů. Vytvoříme si klienstkou aplikaci, která tato data získá a zobrazí, například v tabulce. Nyní se rozhodneme, že by měl přibýt v modelu sloupec, obsahující zkratku týmu.

Nejdříve na to nahlédname z pohledu zisku dat. Upravíme-li model na serveru, v datech, která jsou poskytována webovou službou, přibude další hodnota. Proto je nutné upravit klientskou aplikaci, aby s těmito dodatečnými daty počítala a rovněž je zobrazila v tabulce. Pokud se však rozhodneme, že se nějaký sloupec odstraní, je situace o trochu složitější. Po získání dat nám na klientovi hodnota bude chybět. Pokud je nad hodnotou prováděna nějaká operace a klient není správně ošetřený, může to vyústit i v pád aplikace.

Nyní budeme data posílat. Server může určovat, které hodnoty vyžaduje. Pokud tedy přidáme novou hodnotu, kterou server označí jako povinnou, bude pokus neupraveného klienta zaslat data neúspěšný, neboť je server odmítne. Opět je nutné upravit tak, aby bylo možné novou hodnotu zadat, to znamená přidat nové vstupní pole a upravit parser, či objekt, ze kterého se data připravují serializací na odeslání. Nastane-li odstranění nějakého sloupce z modelu na serveru, bude to pro klienta opět problém, protože bude zasílat data obsahující hodnotu, kterou server nezná a ten data opět odmítne a znovu je nutné klienta upravit.

Pokaždé, kdy jsou provedeny úpravy v klienstké aplikaci, se musí vydat její nová verze. Bohužel v dnešní době je možnost aktualizaci neprovést, a to hlavně na mobilních zařízeních, příkladem může být Google Play na Androidu [14]. Když se aplikace neaktualizuje, uživatel může mít nefunkční aplikaci nebo může nastat chyba na serveru, záleží na provedené změně. Tento problém se řeší například podmínkami na verzi aplikace nebo vynucením aktualizace při startu aplikace. Výše zmíněný problém by byl eliminován, pokud by server klienta informoval o tom, co vyžaduje a klient by se dynamicky těmto potřebám přizpůsobil, aniž by musela být klienská část jakkoliv upravována.

2.1.4 Existující řešení

Tato sekce popisuje existující řešení pro mobilní aplikace, která se snaží o řešení výše uvedených problémů. Jedno řešení nabízí vývojáři z IBM developerWorks ve svém článku Build dynamic user interfaces with Android and XML [12]. Článek popisuje možnost dynamického vytvoření formuláře z XML souboru pro Android aplikace. Podle návodu aplikace stáhne z URL adresy určitý XML soubor, ve kterém je nadefinována struktura formuláře. Návod dále ukazuje, jak stažené XML parsovat a dynamicky vytvořit na jeho základě v aplikaci formulář. Tento způsob formulář centralizuje, tedy pokud se formulář vyskytuje na více místech v aplikaci a je třeba ho změnit, stačí upravit daný XML soubor.

Dalším řešením je projekt AFSwinx společně s AFRest [20], který uvedený princip rozšiřuje o to, že se popis formuláře automaticky generuje z modelu na serveru a klient tento popis získává pomocí webových služeb. Tento framework však není určen pro mobilní apli-

kace, nýbrž pro Java SE platformu. AFSwinx a AFRest vycházejí z AspectFaces [3], který uvedené problémy řeší v Java EE aplikacích.

2.1.4.1 PHP Database Form

PHP Database Form [7] je rozšíření pro jazyk PHP. Toto rozšíření dokáže automaticky z modelu v databázi vytvořit HTML kód formuláře, včetně validací jednotlivých polí. Umožňuje vybrat pro vytvoření pouze některou část tabulky a to pomocí SQL dotazu. Dále pak poskytuje možnost dodatečných nastavení názvů polí, jejich viditelnosti či způsobu zobrazení. Lze také dodat validace tam, kde nebyly určeny databázovým modelem. Hlavními výhodami tohoto rozšíření jsou: menší množství kódu, jednoduché validování dat a možnost upravit si formulář dle libosti pomocí CSS. Využití vyžaduje PHP verzi 5.3 a Apache, Tomcat nebo Microsoft IIS web server. PHP Database Form podporuje všechny majoritně využívané databáze a webové prohlížeče. Dnes už by se i toto rozšíření dalo použít pro mobilní aplikace, neboť existují možnosti vytvářet multiplatformní mobilní aplikace pomocí HTML, CSS a JavaScriptu, které spouští aplikaci na mobilním zařízení v režimu webového prohlížeče. Takovou možností je například Apache Cordova [2].

2.1.4.2 AspectFaces

AspectFaces je framework, který se snaží o to, aby bylo UI generováno na základě modelu [16], k čemuž využívá inspekci tříd. To umožní nadefinovat UI pouze jednou a veškeré změny v modelu jsou automaticky do uživatelského rozhraní reflektovány. UI lze nadefinovat v modelu pomocí velkého množství anotací z JPA, Hibernate nebo si lze nadefinovat i anotace vlastní. Lze určit například pravidla pro dané pole, pořadí v UI nebo label. Framework zatím poskytuje dynamickou integraci pouze s JavaServer Faces 2.0, ale pracuje se na integraci i s jinými technologiemi. Poslední stabilní verze frameworku je 1.4.0 a je dostupný pod licencí LGPL v3.

2.1.4.3 AFSwinx a AFRest

Tento framework byl vytvořen jako koncept a slouží pro generování uživatelského rozhraní v Java SE aplikacích využívajících pro tvorbu UI knihovnu Swing [20]. Framework používá RESTful webové služby pro zisk definic komponent, díky kterým je schopen dynamicky postavit formulář či tabulku. Takové definice komponent vznikají za pomocí části frameworku AFRest, která ke generování dat využívá inspekce příslušného modelu na serveru, na který by měla být komponenta namapována. Jelikož se tvoří komponenta na základě tohoto modelu, nenastane tak, že by s ním nebyla v souladu. Inspekci tříd zprostředkovává knihovna AspectFaces, uvedená výše. Definice komponenty je přenášena ve formátu JSON a obsahuje informace o komponentě, například její rozložení, pole, které má obsahovat nebo pravidla, která pro jednotlivíá pole platí. Pole z definice se v případě formuláře interpretuje jako vstupní políčko, v případě tabulky jako sloupec.

2.1.5 Cíle práce

Vzorem pro tento projekt je výše zmíněný framework AFSwinx spolu s AFRest[20]. Framework se snaží o zjednodušení tvorby uživatelských rozhraní hlavně z hlediska množství kódu a udržovatelsnosti. Jak bylo popsáno, framework na straně serveru využívá inspekce tříd k vytvoření definice modelu, které poskytuje klientovi pomocí webových služeb, stejně tak jako data, kterými se má budoucí komponenta naplnit. Klient tyto informace pouze získává a interpretuje je, nemá tedy informaci o celém procesu tvorby komponenty, zná pouze nutné informace jako je formát dat, například JSON, XML a připojení na zdroje webových služeb, ze kterých data získává. Na vytvoření komponenty stačí klientovi pouze pár řádků kódu. Cílem této práce je vytvořit obdobný framework pro mobilní platformy Android a Windows Phone. Žádoucí je také některé prvky z AFSwinx a AFRest znovupoužít a případně přinést do stávajícího frameworku i něco navíc.

Kapitola 3

Analýza

3.1 Funkční specifikace

V rámci této práce bude zpracován framework ve dvou verzích, pro mobilní platformu Android a mobilní platformu Windows Phone. Musí umožňovat jednoduše vytvářet dva typy komponent, formulář, který umožní uživatelský vstup a list, pro zobrazení většího množství dat uživateli. Kromě vytvoření komponent je nutné poskytkout další funkce, které umožní práci s vytvořenými komponentami, jako je například odeslání dat z komponenty na server. Framework musí samozřejmě disponovat funkcionalitou, která umožní správné vytvoření a nastavení komponenty z hlediska zabezpečení, získávání dat a jejich vložení do komponenty, vzhledu komponenty či její lokalizace. Všechny funkční požadavky jsou uvedeny v následujícím seznamu položek.

3.1.1 Funkční požadavky

- Framework bude umožňovat automaticky vytvořit formulář nebo list na základě dat získaných ze serveru.
- Framework bude umožňovat získat ze serveru data, kterými komponentu naplní.
- Framework bude umožňovat naplnit formulář i list daty.
- Framework bude umožňovat odeslat data z formuláře zpět na server.
- Framework bude umožňovat používat lokalizační texty.
- Framework bude umožňovat validaci vstupních dat na základě definice komponenty, kterou obdržel od serveru.
- Framework bude umožňovat upravit vzhled komponenty pomocí skinů.
- Framework bude umožňovat koncovému uživateli specifikovat zdroje definic komponent, dat a cíle pro jejich odeslání ve formátu XML.
- Framework bude umožňovat vytvářet následující formulářová pole - textové, číselné, pro hesla, pro datum, dropdown pole, checkboxy, option buttony.

- Framework bude umožňovat resetovat úpravy ve formuláři nebo formulář vyčistit.
- Framework bude umožňovat získat data z formuláře i listu.
- Framework bude umožňovat schovat validační chyby.
- Framework bude umožňovat jednoduše získat komponentu i na jiném místě v programu, než kde ji vytvořil.
- Framework bude umožňovat generování komponent určených pouze pro čtení.

Pro uživatele, který bude framework využívat, bude proces tvorby komponenty zapouzdřen. Nemusí znát strukturu definice komponenty, ani jak se komponenta tvoří či naplňuje daty. Bude potřebovat znát jen kód pro vytvoření komponenty, akce, které lze nad komponentou provádět a jak specifikovat, odkud se bere definice komponenty, data pro její naplnění a kam se případně data odešlou.

3.2 Popis architektury a komunikace

3.2.1 Definice komponent

Frameworky pro mobilní platformy Android a Windows Phone, které je cílem vytvořit, navazují, jak už bylo zmíněno, na projekt AFSwinx a AFRest [20]. Tento framework vytváří na straně serveru tzv. definice komponent, které komponentu popisují z hlediska vzhledu, rozložení i obsahu. Jedná se tedy o metadata[**?**], neboli data, která popisují další data. Taková definice vzniká na serveru ve formátu XML na základě inspekce modelu, kterou zprostředkovává knihovna AspectFaces a AFRest ji zobecňuje a převádí do formátu JSON. Na serveru zastupuje roli modelu databázová entita a vlastnosti, které má inspekce modelu zachytit a do definice promítnout, jsou určeny datovými typy atributů a pomocí anotací. Definice komponenty, kterou lze získat ze serveru, obsahuje tyto infomace:

- název definice
- celkové rozložení komponenty
- informace o polích v daném pořadí, které se mají v komponentě vyskytnout

V informacích o poli lze nalézt

- typ widgetu, kterým má být vytvářené pole reprezentováno,
- jednoznačný identifikátor v rámci komponenty,
- popisek neboli label,
- viditelnost,
- zda má být pole určeno jen pro čtení,
- zda se jedná o primitivní či složený datový typ,

- validační pravidla,
- v případě některých widgetů i možnosti, ze kterých si má uživatel vybírat.

Cílem autora AFSwinx a AFRest bylo, aby tyto definice komponent byly nezávislé na platformě, což se i jejich využitím na mobilních platformách potvrzuje. S tvarem těchto definic oba frameworky počítají, na základě tohoto tvaru se definice na klientovi udržuje a z ní tvoří uživatelské rozhraní.

3.2.2 Získání definice ze serveru

Definici komponenty je možné získat pomocí HTTP dotazu na konkrétní zdroj na serveru, který schopen takovou definici za použití AFSwinx a AFRest poskytnout. Tento konkrétní zdroj, poskytující definici komponenty, je nutné specifikovat. Také lze určit dva další zdroje - zdroj dat a zdroj, na který se má odeslat uživatelský vstup. Frameworky mají, dle požadavků výše, umožňovat uživateli tyto zdroje specifikovat ve formátu XML. Již v AFSwinx a AFRest byl pro tento účel vytvořen XML soubor a k němu příslušný XML parser, které je Android verzi frameworku žádoucí z hlediska efektivity využít. Jelikož je AFSwinx a AFRest napsaný v Javě a Windows Phone nepodporuje Javu, nýbrž jazyk C#, a tedy ani import .jar souborů, nelze parser znovu použít i ve Windows Phone frameworku a je nutné ho přepsat. Způsob specifikace všech tří zmíněných zdrojů, konkrétně pro profilový formulář, je zobrazen v následujícím urývku kódu 3.1 ze zmíněného XML souboru.

Listing 3.1: Ukázka XML specifikace zdrojů

```
<?xml version="1.0" encoding="UTF-8"?>
<connectionRoot xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <connection id="personProfile">
      <metaModel>
         <endPoint>toms-cz.com</endPoint>
         <endPointParameters>/AFServer/rest/users/profile</endPointParameters>
         <protocol>http</protocol>
         <port></port>
         <header-param>
            <param>content-type</param>
            <value>Application/Json</value>
         </header-param>
      </metaModel>
      <data>
         <endPoint>toms-cz.com</endPoint>
               <!-- ... obdobne jako metamodel-->
         <security-params>
            <security-method>basic</security-method>
            <userName>#{username}</userName>
            <password>#{password}</password>
         </security-params>
      </data>
      <send>
         <endPoint>toms-cz.com</endPoint>
               <!--... obdobne jako metamodel -->
         <security-params>
               <!--... obdobne jako data -->
         </security-params>
```

```
        </send>
    </connection>
</connectionRoot>
```

Jak lze z ukázky vidět, jsou zdroje nadefinovány URL adresou rozdělenou na části a dodatečnými parametry, jako je forma dat, která lze očekávat nebo zabezpečení. Například definice profilového formuláře se nachází na adrese <http://toms-cz.com/AFServer/rest/users/profile> a je očekávána ve tvaru JSON souboru. Pokud by byl specifikován port, přibude za toms-cz.com ještě dvojtečka a jeho hodnota. Zdroj dat je nadefinován v uzlu <data> a zdroj, na který se odešle uživatelský vstup, určuje uzel <send>. Lze také specifikovat metodu (get, post, put, delete), která se pro kontantování zdroje použije. Pro data a meta model je v základu použita metoda GET a pro odeslání metoda POST.

Výrazy ve složených závorkách označené vpředu pomocí znaku # jsou určeny k nahrazení hodnotou. V AFSwinx a AFRest [20] se klíč ve složených závorkách hledá v mapě parametrů pro připojení, kterou framework předává jako argument metodě kontaktující zdroj, a nahrazuje se hodnotou v ní pod klíčem uloženou. Umožňuje to tak nadefinovat zdroj v XML souboru pouze jednou, například pro více různých uživatelů. Klíč a hodnotu si může uživatel nastavit sám, jen se musí shodovat klíče ve zmíněné mapě a v souboru. V zájmu znovupoužití XML souboru a parseru se tedy tomuto chování oba tvořené frameworky přizpůsobují.

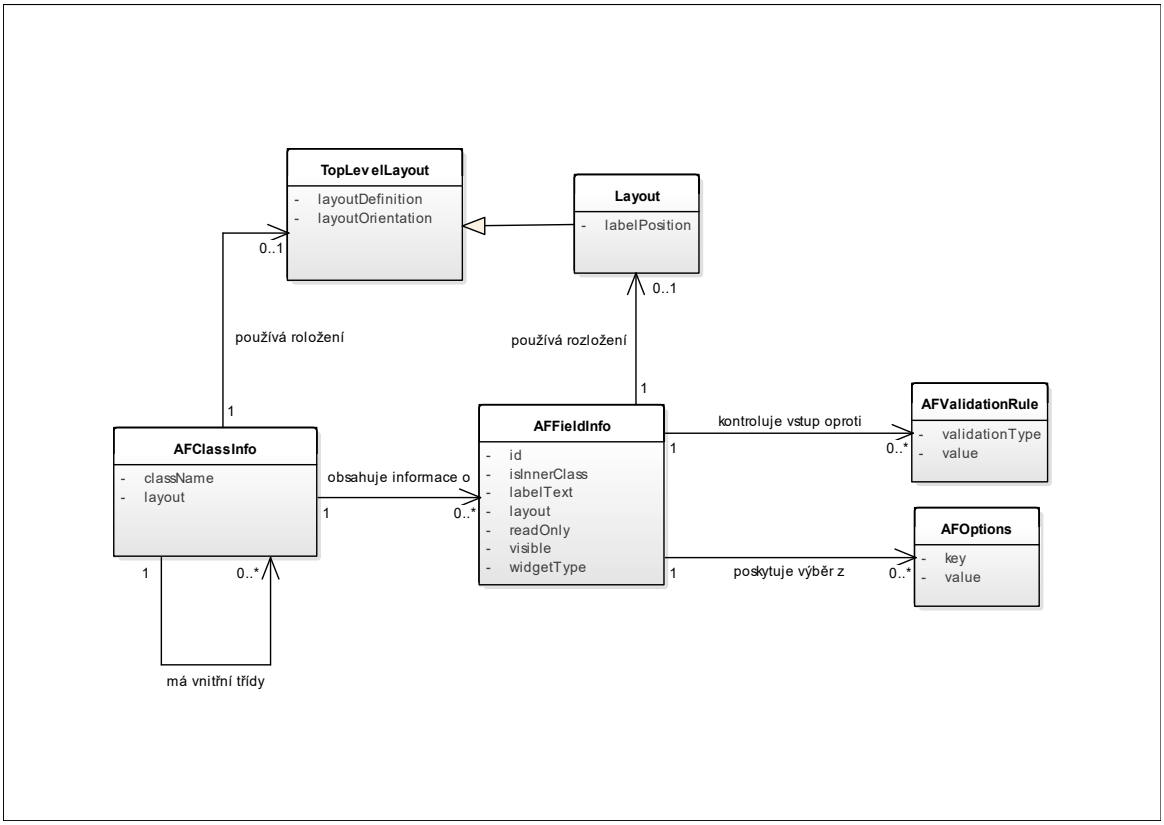
3.2.3 Reprezentace metadat ve frameworku

Získaná metadata je potřeba ve frameworku nějak rozumně udržovat. AFSwinx už pro to určitou strukturu definuje [20] a je tedy žádoucí ji opětovně využít. Tuto část systému, která uchová získané informace o komponentě, zachycuje následující doménový model, vytvořeného za pomocí UML. Dle Arlowa je UML, česky unifikovaný modelovací jazyk, univerzální jazyk pro vizuální modelování systémů. UML je velice silný nástroj hlavně proto, že je srozumitelný pro lidi a zároveň je navržen tak, aby byl univerzálně implementovatelný [13]. Doménový model je výsledkem hledání analytických tříd, definuje jaké části je potřeba v systému mít a jak se vzájemně ovlivňují. Jde tedy o model popisující strukturu i chování systému. Reprezentuje se ve formě diagramu tříd, ve kterém však není nutné uvádět datové typy atributů, ani metody, které bude třída poskytovat.

Následující část práce podrobněji rozebírá diagram z obrázku 3.1, neboť je pro vývoj obou frameworků důležitý. Android framework tuto část AFSwinx a AFRest recykluje a Windows Phone ji transformuje do jazyku C#.

3.2.3.1 AFClassInfo

AFClassInfo udržuje informace o hlavním objektu metadat. Obsahuje informace o názvu objektu a rozložení komponenty. Dále drží definice 0 až N informací o polích, která se mají v komponentě vyskytnout. Součástí je také 0 až N vnitřích tříd, tedy referencí na objekt stejného typu AFClassInfo. Může se totiž stát, že v modelu, nad kterým je prováděna inspekce a ze kterého se metadata vytváří, obsahuje neprimitivní datový typ. Například v modelu Osoba to může být objekt typu Adresa, který obsahuje další atributy jako třeba název ulice či město. Tento typ je ale nutno v komponentě reprezentovat také, a tak je zevnitř provedena jeho inspekce, která je později v metadatech reprezentována jako vnitřní třída.



Obrázek 3.1: Doménový model objektů obsahující metadata o komponentě

3.2.3.2 AFFieldInfo

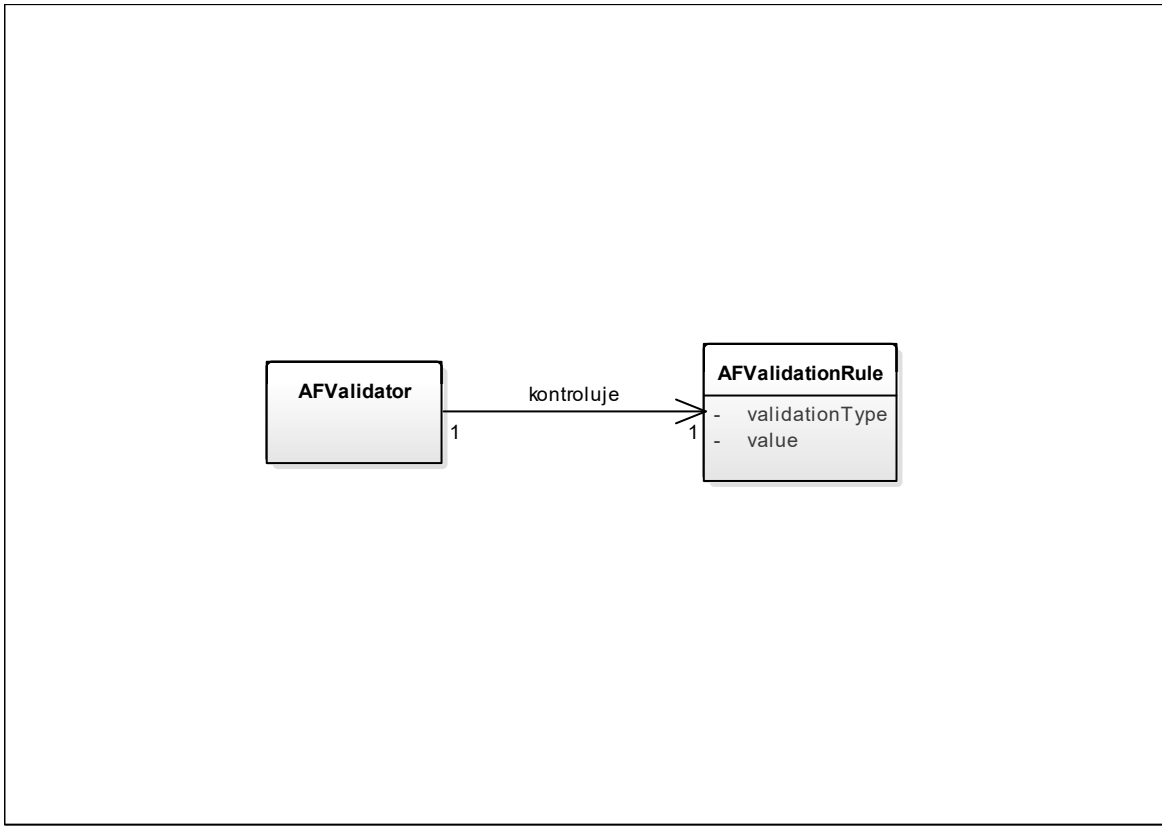
Tento objekt popisuje jednu proměnnou, nad kterou byla provedena inspekce a ze které se má vytvořit pole, které se v komponentě vyskytne. Informuje jaký widget má být při vytváření pole použit, určuje jednoznačný identifikátor pole v rámci komponenty, dále pak, zda má být tvořené pole viditelné a upravitelné, repektive jen pro čtení. Definuje jak bude pole rozloženo, hlavně z hlediska pozice labelu, jehož hodnota je ve AFFieldInfo rovněž zaznamenána. V neposlední řadě jsou v tomto objektu uloženy informace o validační pravidlech, oproti kterým se má validovat uživatelský vstup. Navíc ještě v případě, že by uživatel měl mít na výběr pouze z určitých předem definovaných možností, zahrnuje AFFieldInfo i informace o těchto možnostech.

V tomto objektu je také uloženo, zda se jedná o vnitřní třídu, která je popsána výše. Tento fakt je velmi důležitý, neboť záleží na pořadí polí v komponentě, ve kterém mají být vykreslovány. Inspekce modelu na serveru s tím počítá, a tak pole umístí na správné místo v metadatech a označí ho jako classType, tedy vnitřní třídu, jejíž popis můžeme nalézt v metadatech v části s vnitřními třídami. V rámci zachování správného pořadí vykreslení polí je tedy nutné, aby klienstký framework fakt, že se jedná o složený datový typ, při vytváření polí komponenty zaznamenal a na pozici, kde tuto skutečnost objeví, vložil pole, o nichž jsou informace uloženy v příslušné vnitřní třídě.

3.2.3.3 AFValidationRule

Tento objekt popisuje pravidlo, které má splňovat uživatelský vstup ve vytvářeném poli. Obsahuje typ validace, který určuje o jakou validaci se jedná a případně hodnotu pravidla. Referenční framework AFSwinx obsahuje výčtový typ s názvy validací, které podporuje a které se mohou tedy v metadatech objevit. Například definuje validační pravidlo typu MAX a hodnotou je nějaké číslo. Tedy popisuje, že hodnota v poli nesmí přesáhnout číslo určené hodnotou pravidla.

Každé validační pravidlo má příslušný validátor, což lze vidět na obrázku 3.2, který zobrazuje tuto část systému. Ten při validaci polí komponenty provede kontrolu uživatelského vstupu podle daného pravidla a informuje framework o výsledcích. Ten pak musí disponovat funkcionalitou, který umožní zobrazit případné chybové hlášky uživateli.



Obrázek 3.2: Část doménového modelu systému pro validaci

3.2.3.4 AFOptions

Pro určité typy widgetů, které mají být použity pro vytvoření polí, je nutné specifikovat možnosti, ze kterých si bude uživatel vybírat. Takovými widgety je například dropdown menu nebo skupina radio buttonů. Tento objekt popisuje tyto možnosti formou klíče a hodnoty. Klíč je hodnota, kterou by měl framework odesílat na server a hodnota by měla být zobrazována klientovi.

3.2.3.5 TopLevelLayout

Objekt by měl být využit k popisu rozložení celé komponenty. Objekt definuje dvě vlastnosti. Za prvé je to orientace, tedy ve směru jaké osy bude komponenta či její část vykreslována. Dále je to pak definice rozložení, která má určovat, jestli bude komponenta či její části vykreslovány v jednom či více sloupcích.

3.2.3.6 Layout

Popisuje rozložení částí komponenty, tedy vytvářených polí. Jak je vidět na obrázku **??**, dědí z TopLevelLayoutu orientaci a definici rozložení, které jsou popsány výše. Navíc má vlastnost LabelPosition, která by měla být tvořenými frameworky využita k umístění labelu vzhledem k vytvářenému poli.

3.2.4 Tvorba komponent

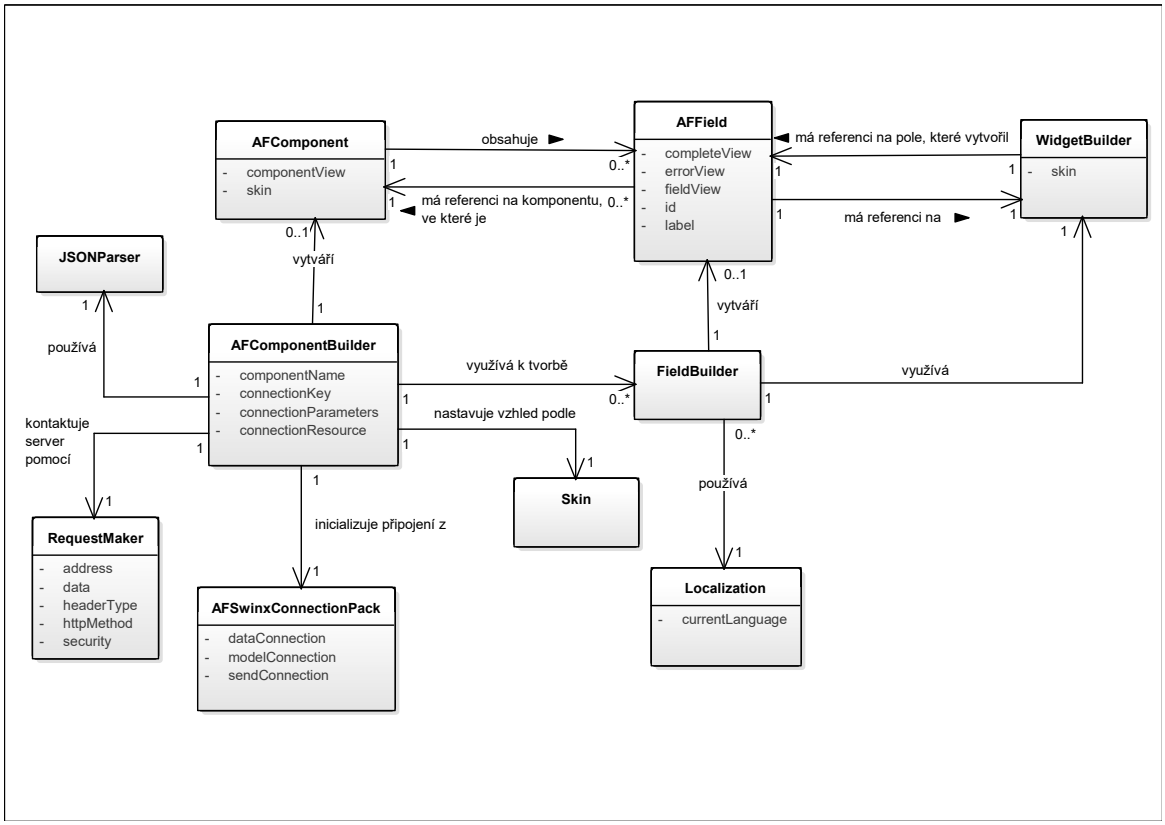
Z přijatých a uložených metadat umí frameworky vytvořit prozatím dva typy komponent. Jednou komponentou je formulář, který řeší hlavně uživatelský vstup, ale může být využit, v případě, že je předvyplněn daty, i ke zobrazení informací uživateli. Druhou komponentou je list, neboli seznam položek, který uživateli umožňuje přehledné zobrazení většího množství informací. Ve frameworku AFSwinx tuto možnost zajišťovala tabulka [20], která však není pro mobilní zařízení úplně vhodným způsobem, protože vyžaduje pro rozumné zobrazení mnoho místa, které na mobilních zařízeních většinou není k dispozici.

Strukturu metadat, získaných ze serveru, je možné využít zároveň pro formulář i list, neboť se liší pouze grafická reprezentace metadat. Zatímco ve formuláři lze využít definic proměnných, nad kterými byla provedena inspekce, k tvorbě formulářových polí, v listu je lze použít k tvorbě informací o jedné z jeho položek. Rozložení, které je definováno ve výše popsaném TopLevelLayoutu, zase lze použít v případě formuláře k určení uspořádání polí. Stejně to lze udělat v listu s uspořádáním informací o položce. Některé informace sice nejsou v listu využity, jako například typ widgetu nebo validace, ale pořád je výhodnější některé informace z definice nepoužít, než aby obě komponenty měly vlastní strukturu metadat.

Pro zobrazení procesu tvorby formuláře, který je tou složitější komponentou, jelikož uživatelský vstup, který do něho bude zadáván se musí validovat a odesílat na server, narozdíl od listu, který je pouze pro čtení, byl využit diagram aktivit. Diagramy aktivit popsují určitý proces složený z dílčích podprocesů a můžou být použity například právě pro analýzu a popis algoritmu [13].

V diagramu B.1 lze vidět, že pokud bude chtít uživatel frameworku vytvořit formulář, bude muset nejdříve specifikovat, kde framework najde metadata. Ten pak o tato data požádá server. Server musí samozřejmě data dynamicky vytvořit, a tak provede inspekci, o které jsme již psal výše a vytvoří metadata, která pak klientskému frameworku předá zpět. Ten je určitým způsobem zpracuje a postaví na základě nich požadovanou komponentu. Pokud uživatel zároveň nadefinoval i zdroj dat, kterými by se měl formulář naplnit, požádá klient znovu server o tato data. Server je vygeneruje a opět zašle zpět, načež se komponenta těmito daty naplní. Poté se takto vytvořená komponenta předá uživateli, tedy vývojáři, který s ní může dále pracovat, například ji vložit do GUI tam, kam potřebuje.

Navržená struktura frameworku sloužící k realizaci procesu tvorby komponenty je zobrazena v části doménovém modelu systému na následujícím obrázku.



Obrázek 3.3: Část doménového modelu systému realizující tvorbu komponenty

3.2.4.1 AFComponent

Tato třída zastřešuje vytvořenou komponentu. Jak bylo již zmíněno, frameworky podporují dva typy kompoenent, a to formulář a list. Grafická reprezentace komponenty je uložena v atributu componentView. Také obsahuje skin, podle kterého je vzhled komponenty nastaven. Komponenta obsahuje 0 až N tříd typu AFField, které jsou popsány v následující sekci.

3.2.4.2 AFField

AFField popisuje vytvořenou část komponenty, tedy její pole. Pole má svůj jednoznačký identifikátor a je složeno ze tří grafických prvků. Prvním prvkem uloženým v atributu fieldView je widget, kterým je pole v GUI reprezentováno. Druhým prvkem je errorView, který slouží pro zobrazení validačních chyb uživateli a posledním prvkem je label neboli popisek pole. Všechny tři prvky se kombinují v určitém rozložení do atributu completeView, který v první řadě poskytuje jednoduchý přístup k celkové grafické reprezentaci pole.

3.2.4.3 AFComponentBuilder

Pro rozlišení, zda se z metadat vytvoří formulář nebo list, jsou vytvořeny dva typy builderů, které komponenty staví. Uživateli, který bude framework používat, by tedy mělo stačit specifikovat builder, který požadovanou komponentu vytvoří. Aby bylo používání frameworků co nejvíce uživatelsky přívětivé, způsob tvoření builderů a jejich používání by se neměly lišit. Uživateli pak bude stačit naučit se pouze vytvořit jeden typ komponenty a druhý typ vytvoří pouze výměnou builderu. Na obrázku 3.3 jsou buildery vyobrazeny jako AFComponentBuilder. V digramu lze vidět, že může builder vytvářet 0 až 1 komponent, to znamená, že builder může být pouze nadefinován a nemusí být použitý pro tvorbu komponenty, čímž však ztrácí svůj smysl, proto případ tvorby žádné komponenty s největší pravděpodobností nenastane, nicméně je možný. Uživatel při specifikaci builderu může napstavit jednoznačný identifikátor komponenty, kterou tvoří, což reprezentuje atribut componentName. Dále je nutné určit již zmíněný XML soubor s definicemi připojení, který reprezentuje atribut connectionResource. V tomto souboru se pak vybere příslušné připojení na základě uživatelem zadaného klíče v atributu connectionKey. Atribut connectionParameters pak definuje dodatečné parametry ve formě klíč-hodnota pro připojení na server. Takovými parametry můžou být například uživatelské jméno a heslo sloužící pro autorizace uživatele, kterému se má komponenta zobrazit. Na základě těchto specifikací se ze zmíněného XML souboru 3.1 vytvoří připojení pro definici komponenty, data a odeslání dat z komponenty na server, která se uloží AFSwinxConnectionPack.

3.2.4.4 RequestMaker a JSONParser

Definovaných připojení pak využije třída RequestMaker, zajišťující komunikaci builderu se serverem. Tato třída vytváří HTTP requesty na URL adresu definovanou v atributu address. Použije k tomu metodu specifikovanou v httpMethod, možné metody jsou get, post, put, delete. Atribut data obsahuje případná data, která se mají na server odeslat, atribut headerType určuje formát těchto dat. Některé zdroje mohou být zabezpečené, proto je zde i atribut security, který definuje uživatele a bezpečnostní metodu. Prozatím je podporována pouze BASIC autentifikace.

Pokud se třída využije pro zisk definice komponenty, následuje, jak již bylo zmíněno, rozbor této definice a její následné umístění do struktury pro uložení metadat, která byla popsána na obrázku 3.1. K tomuto účelu AFComponentBuilder využívá dle obrázku 3.3 třídu JSONParser.

3.2.4.5 FieldBuilder a WidgetBuilder

Komponenta se skládá z několika částí, tedy polí komponenty, které je potřeba také vytvořit. O to se stará FieldBuilder, který vytváří žádné nebo jedno pole. Podobně jako u AFComponentBuilder případ nevytvoření pole je velmi nepravděpodobný, protože by builder opět ztratil svůj účel. FieldBuilder slouží pro vytvoření tří prvků GUI, které pole obsahuje, jak bylo popsáno u AFField.

K vytvoření části, která určuje widget komponenty se využívá WidgetBuilder. V metadatech se nachází pro každé pole, které má být součástí komponenty, typ widgetu, kterým má být pole reprezentováno. Například to může být textové pole, checkbox nebo skupina radio

buttonů. Ke každému widgetu existuje vlastní builder, reprezentovaný právě touto třídou. WidgetBuilder nejen widget vytváří, ale také určuje, jak se z widgetu dají získat data a naopak, jak je do něj vložit. WidgetBuilderu je také předáván skin, který určuje, jak bude widget vypadat.

3.2.4.6 Lokalizace

V rámci metadat, která přijdou ze serveru se mohou objevit texty určené pro lokalizaci, tedy překlad. Tyto texty jsou realizované pomocí klíčů, ke kterým lze překlad přiřadit a vyskytují se v labelech, v možnostech, ze kterých může uživatel v daných typech widgetů vybírat nebo ve validačních chybách. Frameworky tedy musí disponovat funkcionalitou, který tuto lokalizaci umožní včetně změny jazyka za běhu aplikace. K tomu FieldBuilder používá třídu Localization, která má atribut currentLanguage zachycující aktuální jazyk.

V Androidu se typicky tyto lokalizační texty umisťují do souboru strings.xml ve složce values. Tuto vlastnost je požadováno zanechat, a tak i musí být Android verze frameworku navržena. Pro Windows Phone je to obdobné. Zde se texty vkládají do souborů Resources.resw, které jsou většinou umístěny ve složce Strings. Aby byl způsob lokalizace textů v aplikaci jednotný, je možné využít lokalizační část frameworku i nad jeho rámec k překladu například textů tlačítek či položek v menu.

3.2.5 Práce s vytvořenou komponentou

Komponentu nestačí jen vytvořit, ale cílem frameworku je také umožnit s ní další práci. Vývojáři by mělo být umožněno například odeslat formulář, zkontrolovat to, co uživatel zadal nebo upravit její vzhled. Práci s komponentou lze demostrovat na práci s formulářem. Pro tento účel byl vytvořen diagram aktivit, který popisuje tři hlavní možnosti práce s formulářem a to odesílání dat, resetování a vyčištění formuláře. Tento diagram lze nalézt v příloze B.2.

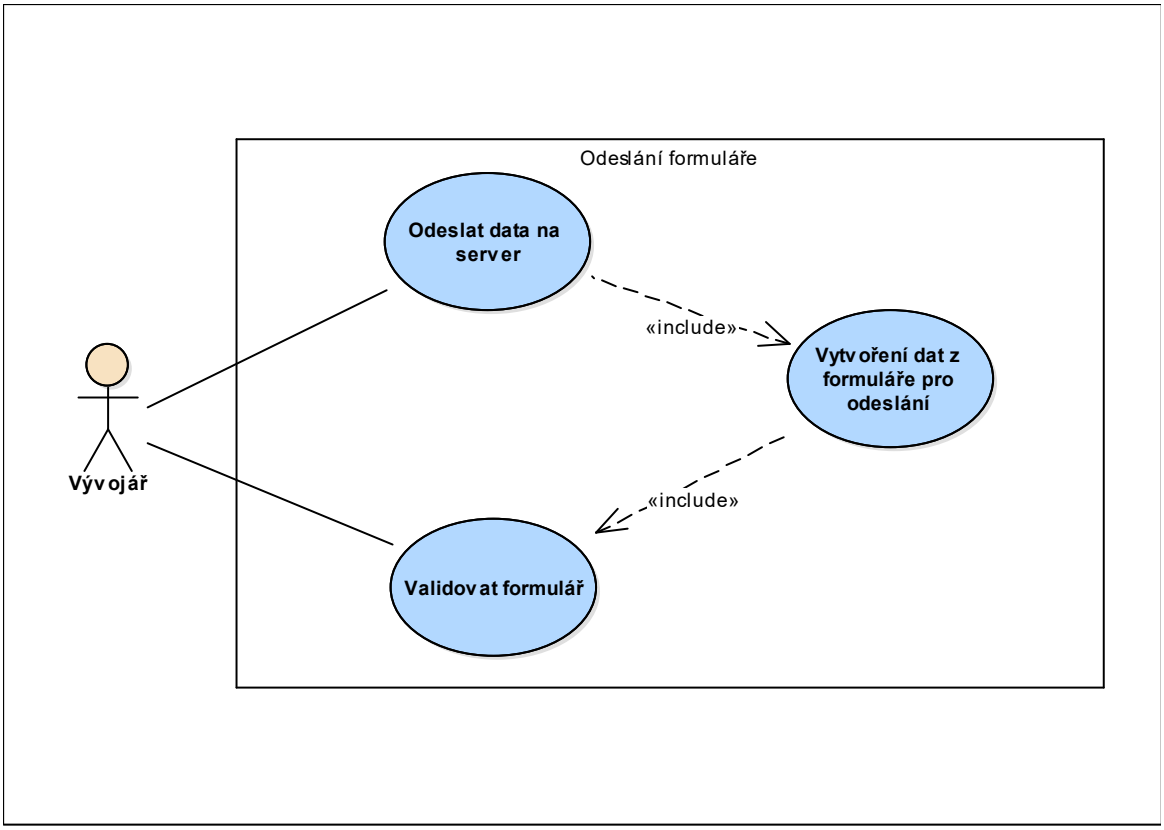
V případě odesílání dat diagram popisuje, že se data nejdříve musí validovat. Pokud jsou data nejsou validní, zobrazí se validační chyby a proces končí. Pokud však validní jsou, hraje roli ještě fakt, zda je nebo není nadefinovaný zdroj, kam se data mají odeslat. Pokud zdroj chybí, je o tomto informován uživatel a proces opět končí. Pokud zdroj nechybí a proces pokračuje dále, data se zformují do podoby, kterou server vyžaduje a odešlou se. Jelikož jsou data validní a v dobrém formátu, server nebude mít problém je přijmout a zpracovat. Výsledek zpracování je propagován vývojáři, který s ním nějak naloží. Tím proces odeslání končí.

3.3 Případy užití

Návrh možností, které by měly oba mobilní frameworky podporovat, je vidět v diagramu případu užití, který lze nálezt v příloze B.3. Případy užití určují, jak lze používat systém nebo jeho dílčí části [13]. Případ užití je iniciován aktérem, jímž je v tomto případě hlavně vývojář, který bude framework používat. V této práci jsou některé případy užití definovány i pro samotný framework, které zobrazují, jaké akce musí framework provést pro splnění

úkolu, zadaným uživatelem. Definici posloupností jednotlivých akcí popisuje vždy příslušný scénář případu užití.

Jedním z hlavních případů užití je odeslání formuláře. Tuto část use case digramu systému, kterou lze vidět na následujícím obrázku, podrobněji popíšu. Pro stručnost je předpokládán hladký průběh scénáře, samozřejmě všude tam, kde se vyskytne IF, tedy podmínka, by měl být uveden i alternativní scénář, který řeší to, co se stane při nesplnění podmínky.



Obrázek 3.4: Diagram případů užití pro odeslání formuláře

3.3.1 Případ užití: Validace formuláře

Na obrázku 3.4 se tento případ jmenuje Validovat formulář a má zachycovat validaci uživatelského vstupu ve formuláři. V diagramu jde také vidět, že by uživatel měl být schopen užívat validaci formuláře i explcitně, tedy ne jen v rámci odeslání dat. Následující scénář tohoto případu užití popisuje postup, jak by měla validace formuláře probíhat.

Vstupní podmínky:

- Framework zná formulář, který chce validovat.
- Tento formulář musí být vytovřený za pomoci metadat.

Scénář případu užití:

1. Uživatel požádá systém o validaci daného formuláře.
2. Systém získá z formuláře všechna pole
3. WHILE existuje nezvalidované pole DO
   (a) Systém získá všechna pravidla, kterým pole podléhá
   (b) WHILE existuje nenavštívené pravidlo DO
      i. Systém získá pro dané pole a pravidlo příslušný validátor.
      ii. Systém provede validaci.
      iii. IF validace selhala THEN
         A. Systém přidá k chybovým hláškám určených pro zobrazení příslušnou hlášku o chybě validace.
   (c) IF alespoň jedna z validací na poli selhala THEN
      i. Systém zobrazí k danému poli validační chyby.

3.3.2 Případ užití: Vygenerování odesílaných dat

Tento případ užití popisuje akci frameworku, která musí být provedena, aby šlo úspěšně na server odeslat data. Framework nebude znát přímo objekt, který chce na serveru odesláním dat vytvořit nebo upravit. Co však zná, je jeho struktura, na základě které sice objekt nevytvoří, ale je schopen poskládat pro server přijatelná data, ze kterých si server, který objekt již zná, dokáže tento objekt vytvořit. Důležitý je formát dat, ve kterém mají být serveru tyto data zaslány. Referenční AFSwinx a AFRest podporují prozatím jen JSON formát, ale počítá se s přidáním dalších formátů [20]. Bylo by tedy dobré, navrhnout tento případ užití pro více možných formátů. V diagramu 3.4 je tento use case nazván Vytvoření dat z formuláře pro odeslání a popisuje ho níže zmíněný scénář.

Vstupní podmínky:

- Framework zná formulář, ze kterého chce sestavovat data pro odeslání.
- Tento formulář musí být vytovřený za pomoci metadat.
- Framework musí znát serverem akceptovaný formát dat.

Scénář případu užití:

1. Uživatel chce poskládat z formuláře data k odeslání.
2. «include» Validovat formulář
3. IF validace proběhla úspěšně THEN
   (a) Systém získá z formuláře všechna pole
   (b) WHILE existuje pole, které nebylo zahrnuto v odesílaných datech DO

i. Systém získá builder, který pole postavil.
ii. Systém požádá builder o data, která se v poli nachází.
iii. Systém určí název proměnné a třídu, do které patří, a nastaví jí data.
iv. Systém podle formátu dat, které server očekává, rozhodne, v jakém formátu data zaslat a na tento formát data převede.

3.3.3 Případ užití: Odeslání dat na server

Posledním případem užití z obrázku 3.4 je use case Odeslat data na server. Ten zahrnuje předchozí případ, což jde ostatně vidět v níže uvedeném scénáři.

Vstupní podmínky:

- Framework zná formulář, ze kterého chce sestavovat data pro odeslání.
- Tento formulář musí být vytovřený za pomoci metadat.
- Framework musí znát zdroj, na který mají být data odeslána a všechny potřebné informace, které zdroj vyžaduje.

Scénář případu užití:

1. Uživatel chce odeslat data z formuláře na server.
2. «include» Vytvoření dat z formuláře pro odeslání
3. IF bylo vytvoření dat z formuláře úspěšné THEN
   (a) Systém odešle data na specifikovaný zdroj
   (b) Systém informuje uživatele o výsledku akce

3.3.4 Úprava vzhledu komponenty

Důležitým aspektem grafického uživatelského rozhraní je také jeho vzhled, neboť špatně navržený vzhled GUI může vést v některých případech i k ohrožení použitelnosti aplikace. V Android aplikacích se vzhled definuje buď v XML šablonách pro daný view, to v případě, že je GUI vytvořeno staticky [1], nebo pomocí Javy v případě dynamického přístupu.

Na Windows Phone platformě jsou obdobou XML šablon XAML soubory, které z XML vychází, a případě dynamického vytváření, slouží k úpravě metody napsané v C#. Ve Windows Phone aplikacích se preferuje neměnit barvy komponent, protože ke změně barev slouží barevná témata, která mají výhodu v tom, že jsou konzistentní na všech zařízeních [11], ale ovlivňují mimo GUI operačního systému také GUI aplikací. Může se tedy stát, že pokud vývojář nastaví například barvu textu na bílou a aktuální téma, které zařízení používá, má bílé pozadí, nepůjdou texty jednoduše vidět. Windows Phone verze frameworku bude tedy muset zohlednit i tyto situace.

Jelikož se komponenty tvoří dynamicky v rámci frameworku a uživatel k procesu tvorby nemá explicitně přístup, je nutné poskytnout mu možnost nadefinovat vzhled komponenty

předem. K tomuto účelu frameworky disponují skiny, které má uživatel možnost nastavit builderu komponenty. V příloze B.3 se tento případ užití nazývá Nastavit skin. V případě, že by uživatel frameworku skin nenadefinoval, existuje základní vzhled. Od tohoto základního skinu může uživatel ve svých skinech dědit a překrýt pouze části, které mu nevyhovují a chtěl by je jinak. Po sestavení komponenty už není jednoduše možné za pomocí frameworku tento vzhled změnit. Framework sice poskytuje možnost získat reprezentaci komponent i jejich částí, ale změna vzhledu částí vyžaduje znalost dynamické tvorby UI na jednotlivých platformách.

3.4 Práce na existujícím řešení

Cílem práce je také přidat nějakou hodnotu do stávající verze frameworku AFSwinx a AFRest. Pro tento účel bylo rozhodnuto přidat novou anotaci, kterou zohlední dříve zmiňovaná inspekce na serveru při tvorbě metadat. Cílem anotace je sdělit klientovi, že se má anotací označený atribut v modelu na serveru porovnat druhým atributem, který je určen v parametrech anotace, ve smyslu menší nebo rovno než. Anotace se týká hlavně datumů a řeší, zda anotací označený atribut typu Datum obsahuje datum dřívější nebo stejné než druhý atribut určený v parametrech anotace. Tato informace se v metadatech objevuje prozatím pouze u informací o poli s typem widgetu CALENDAR, tedy u těch, jež mají být reprezentovány jako DatePicker. Kontrétně je informace přidána do sekce rules, jedná se tedy o validační pravidlo. AspectFaces [3] ve své dokumentaci popisuje, že pokud chce uživatel přidat novou anotaci, musí vytvořit tzv. anotační deskriptor, který pak musí zaregistrovat v konfiguračním souboru aspectfaces-config.xml uloženém ve složce WEB-INF. Tím se zajistí, že si inspekce anotace všimne a promítne ji do metadat ve formě XML. AFSwinx a AFRest tyto XML soubory ještě převádí do platformově nezávislé podoby [20] a v tomto procesu je nutné taktéž provést změny. Po vytvoření anotace je nutné přidat ji na daná místa na serveru a je žádoucí vytvořit validátory nejen v obou vytvářených frameworcích pro mobilní platformy, ale také v již existujícím řešení AFSwinx pro platformu Java SE.

Literatura

[1] *Styles and Themes* [online]. [cit. 26. 03. 2016]. Dostupné z: <http://developer.android.com/guide/topics/ui/themes.html>.

[2] *Apache Cordova* [online]. [cit. 23. 03. 2016]. Dostupné z: <https://cordova.apache.org/>.

[3] *AspectFaces framework* [online]. [cit. 23. 03. 2016]. Dostupné z: <http://www.aspectfaces.com/overview>.

[4] *Gson User Guide* [online]. [cit. 01. 04. 2016]. Dostupné z: <https://github.com/google/gson/blob/master/UserGuide.md>.

[5] *Introducing JSON* [online]. [cit. 01. 04. 2016]. Dostupné z: <http://www.json.org/>.

[6] *What Are RESTful Web Services?* [online]. [cit. 22. 03. 2016]. Dostupné z: <https://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html>.

[7] *PHP Database Form* [online]. [cit. 26. 03. 2016]. Dostupné z: <http://phpdatabaseform.com/>.

[8] *Serialization* [online]. [cit. 01. 04. 2016]. Dostupné z: <https://en.wikipedia.org/wiki/Serialization>.

[9] *Typy uživatelských rozhraní a jejich specifika/old – Wikisofia* [online]. [cit. 22. 03. 2016]. Dostupné z: <https://wikisofia.cz/index.php/Typy_uživatelských_rozhraní_a_jejich_specifika/old>.

[10] *Web service* [online]. [cit. 22. 03. 2016]. Dostupné z: <https://en.wikipedia.org/wiki/Web_service>.

[11] *Themes for Windows Phone* [online]. [cit. 26. 03. 2016]. Dostupné z: <https://msdn.microsoft.com/en-us/library/windows/apps/ff402557(v=vs.105).aspx>.

[12] ABLESON, F. *Build dynamic user interfaces with Android and XML* [online]. [cit. 23. 03. 2016]. Dostupné z: <http://www.ibm.com/developerworks/xml/tutorials/x-andddyntut>.

[13] ARLOW, J. – NEUSTADT, I. *UML 2 a unifikovaný proces vývoje aplikací: Objektově orientovaná analýza a návrh prakticky.* : Computer Press a.s., Brno, 2nd edition, 2008. In Czech. ISBN 978-90-251-1503-9.

[14] BARTLETT, M. *Enable or Disable Auto-Update Apps in Google Play for Android* [online]. [cit. 23. 03. 2016]. Dostupné z: <http://www.technipages.com/auto-update-apps-setting-google-play>.

[15] CERNY, T. – CHALUPA, V. – DONAHOO, M. Towards Smart User Interface Design. In *Information Science and Applications (ICISA), 2012 International Conference on*, s. 1–6, May 2012. doi: 10.1109/ICISA.2012.6220929.

[16] CERNY, T. et al. Aspect-driven, Data-reflective and Context-aware User Interfaces Design. *SIGAPP Appl. Comput. Rev.* December 2013, 13, 4, s. 53–66. ISSN 1559-6915. doi: 10.1145/2577554.2577561. Dostupné z: <http://doi.acm.org/10.1145/2577554.2577561>.

[17] CERNY, T. – DONAHOO, M. J. – SONG, E. Towards Effective Adaptive User Interfaces Design. In *Proceedings of the 2013 Research in Adaptive and Convergent Systems*, RACS '13, s. 373–380, New York, NY, USA, 2013. ACM. doi: 10.1145/2513228.2513278. Dostupné z: <http://doi.acm.org/10.1145/2513228.2513278>. ISBN 978-1-4503-2348-2.

[18] JENKOV, J. *Web Service Message Formats* [online]. [cit. 23. 03. 2016]. Dostupné z: <http://tutorials.jenkov.com/web-services/message-formats.html>.

[19] SOMMERVILLE, I. *Software engineering.* : Addison-Wesley, 9 edition, 2011. ISBN 978-0-13-703515-1.

[20] TOMášEK, M. Aspektově orientovaný vývoj uživatelských rozhraní pro Java SE aplikace. Master's thesis, České vysoké učení technické v Praze Fakulta Elektrotechnická, 1 2015.

Příloha A

Seznam použitých zkratek

CUI Character User Interface

GUI Graphical User Interface

UI User Interface

REST Representational State Transfer

URI Uniform Resource Identifier

HTTP Hypertext Transfer Protocol

JSON JavaScript Object Notation

XML Extensible Markup Language

XAML Extensible Application Markup Language

Java SE Java Platform Standart Edition

Java EE Java Platform Enterprise Edition

URL Uniform Resource Locator

PHP Personal Home Page

HTML HyperText Markup Language

SQL Structured Query Language

CSS Cascading Style Sheets

JPA Java Persistence API

LGPL GNU Lesser General Public License

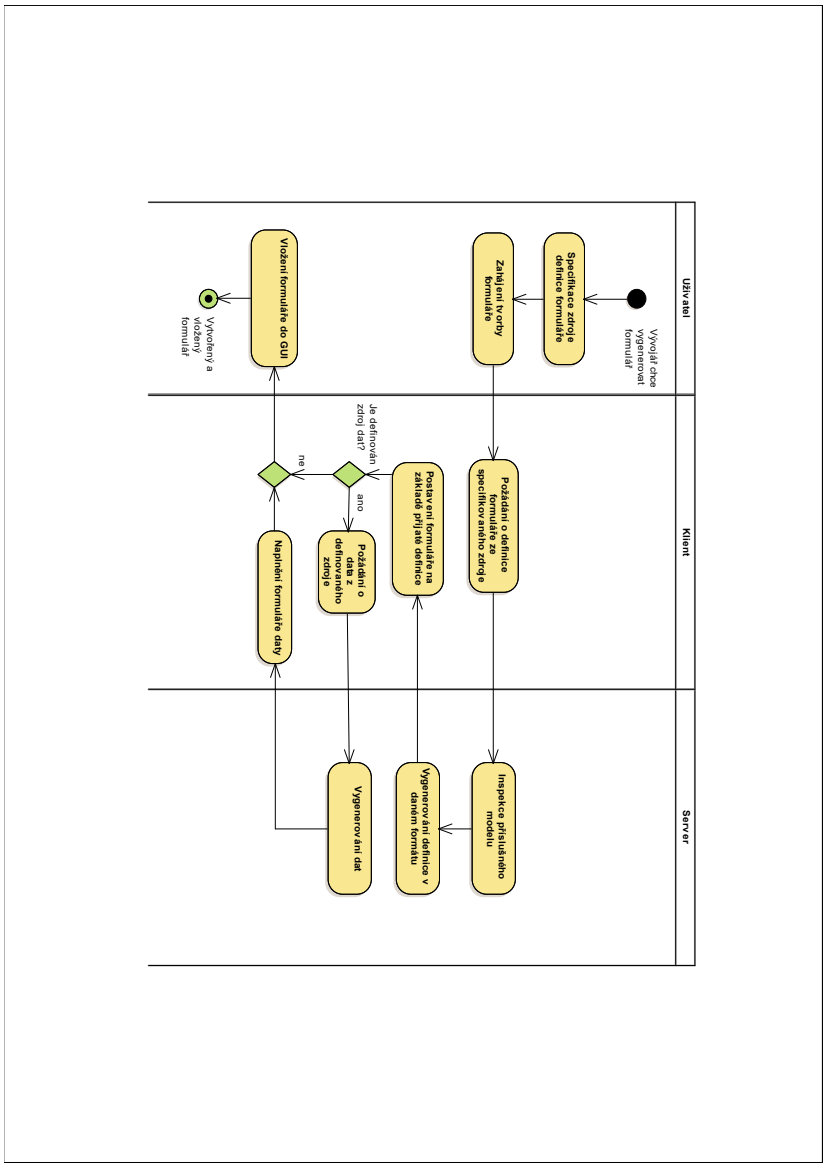
UML Unified Modeling Language

C# C Sharp

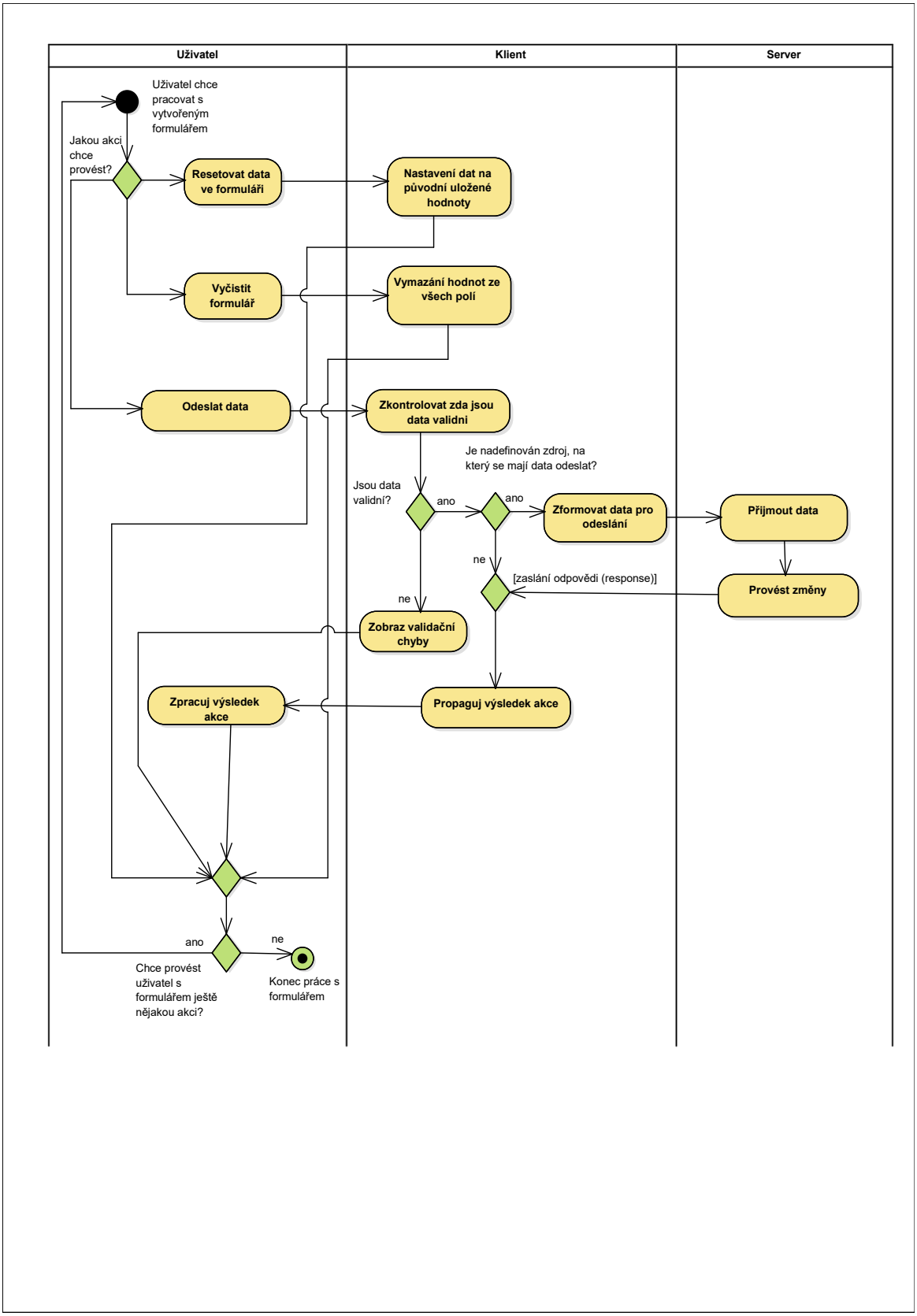
Příloha B

UML diagramy a obrázky

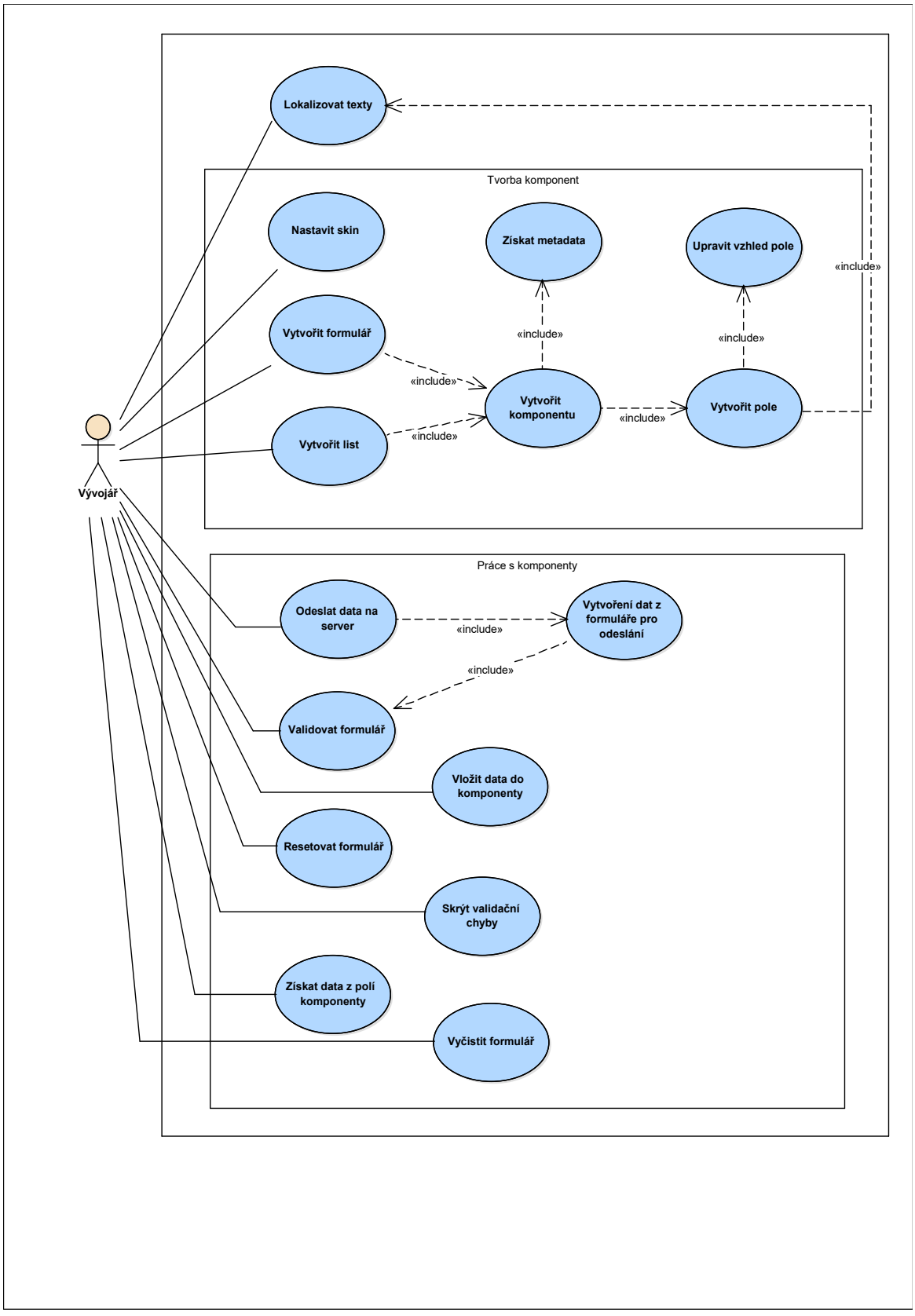
V této sekci naleznete použité UML diagramy a velké obrázky, na které bylo v textu odkazováno.



Obrázek B.1: Diagram aktivit popisující proces tvorby formuláře



Obrázek B.2: Diagram aktivit popisující práci s formulářem



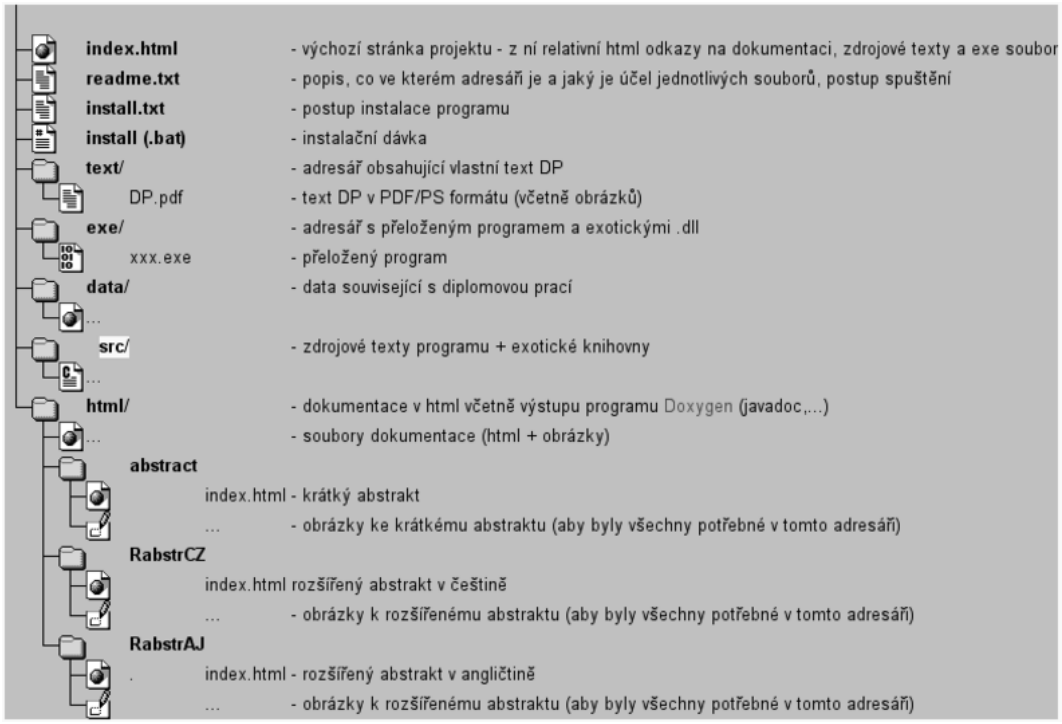
Obrázek B.3: Model případů užití frameworku

Příloha C

Obsah přiloženého CD

Tato příloha je povinná pro každou práci. Každá práce musí totiž obsahovat přiložené CD. Viz dále.

Může vypadat například takto. Váš seznam samozřejmě bude odpovídat typu vaší práce. (viz [**?**]):



Obrázek C.1: Seznam přiloženého CD — příklad

Na GNU/Linuxu si strukturu přiloženého CD můžete snadno vyrobit příkazem:

```
$ tree . >tree.txt
```

Ve vzniklém souboru pak stačí pouze doplnit komentáře.

Z **README.TXT** (případne index.html apod.) musí být rovněž zřejmé, jak programy instalovat, spouštět a jaké požadavky mají tyto programy na hardware.

Adresář **text** musí obsahovat soubor s vlastním textem práce v PDF nebo PS formátu, který bude později použit pro prezentaci diplomové práce na WWW.