

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 demonstруйте 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:

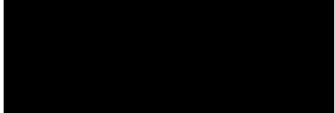
- [1] CERNY, T. \_ CHALUPA, V. \_ DONAHOO, M. Towards Smart User Interface Design. In Information Science and Applications (ICISA), 2012 International Conference on, s. 1\_6, May 2012. doi: 10.1109/ICISA.2012.6220929.
- [2] CERNY, T. et al. Aspect-driven, Data-reflective and Context-aware User Interfaces 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

  
prof. Ing. Filip Železný, Ph.D.  
vedoucí katedry



  
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

15. 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

<b>1</b>	<b>Úvod</b>	<b>1</b>
1.1	Motivace . . . . .	1
<b>2</b>	<b>Popis problému a specifikace cíle</b>	<b>3</b>
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
<b>3</b>	<b>Analýza</b>	<b>9</b>
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 . . . . .	14
3.2.3.6	Layout . . . . .	14
3.2.4	Tvorba komponent . . . . .	14
3.2.4.1	AFComponent . . . . .	15
3.2.4.2	AFField . . . . .	15
3.2.4.3	AFComponentBuilder . . . . .	15
3.2.4.4	RequestMaker a JSONParser . . . . .	17
3.2.4.5	FieldBuilder a WidgetBuilder . . . . .	17
3.2.4.6	Lokalizace . . . . .	17
3.2.5	Práce s vytvořenou komponentou . . . . .	17

3.3	Případy užití . . . . .	18
3.3.1	Případ užití: Validace formuláře . . . . .	18
3.3.2	Případ užití: Vygenerování odesílaných dat . . . . .	20
3.3.3	Případ užití: Odeslání dat na server . . . . .	20
3.3.4	Úprava vzhledu komponenty . . . . .	21
3.4	Práce na existujícím řešení . . . . .	21
3.5	Použité technologie . . . . .	22
3.5.1	Java a Android SDK . . . . .	22
3.5.2	C# a Windows Phone SDK . . . . .	22
3.5.3	AFSwinx a AFRest . . . . .	23
3.5.4	Ukázkové projekty . . . . .	23
3.5.4.1	AFServer . . . . .	23
3.5.4.2	Glassfish . . . . .	23
<b>4</b>	<b>Implementace</b>	<b>25</b>
4.1	Architektura . . . . .	25
4.1.1	Komponenty . . . . .	26
4.2	Komunikace serveru a klienta . . . . .	28
4.2.1	Generování komponent . . . . .	29
4.2.1.1	Naplnění komponenty daty . . . . .	30
4.2.1.2	Widget buildery . . . . .	30
4.2.1.3	Skiny . . . . .	30
4.2.1.4	Layouty . . . . .	31
4.2.1.5	Lokalizace . . . . .	32
4.2.2	Práce s komponentami . . . . .	32
4.2.2.1	Odeslání, reset a vyčištění formuláře . . . . .	33
4.2.2.2	Validace a Validátory . . . . .	34
4.3	Úprava AFSwinx a AFRest . . . . .	34
4.4	Porovnání přístupů . . . . .	36
<b>A</b>	<b>Seznam použitých zkratk</b>	<b>43</b>
<b>B</b>	<b>UML diagramy a obrázky</b>	<b>45</b>
<b>C</b>	<b>Obsah přiloženého CD</b>	<b>55</b>

# Seznam obrázků

3.1	Doménový model objektů obsahující metadata o komponentě . . . . .	13
3.2	Část doménového modelu systému realizující tvorbu komponenty . . . . .	16
3.3	Část diagram případů užití pro odeslání formuláře . . . . .	19
4.1	Třídy AFAndroid a AfWindowsPhone sloužící jako fasády pro ovládání fra- meworků . . . . .	27
B.1	Diagram aktivit popisující proces tvorby formuláře . . . . .	46
B.2	Diagram aktivit popisující práci s formulářem . . . . .	47
B.3	Model případů užití frameworku . . . . .	48
B.4	Diagram nasazení systému . . . . .	49
B.5	Diagram tříd zachycující komponenty . . . . .	50
B.6	Diagram tříd zachycující widget buildery . . . . .	51
B.7	Diagram tříd popisující strukturu uložení metadat . . . . .	52
B.8	Sekvenční diagram zobrazující proces tvorby formuláře . . . . .	53
B.9	Sekvenční diagram zobrazující proces odeslání formuláře . . . . .	54
C.1	Seznam přiloženého CD — příklad . . . . .	55



# Seznam tabulek

4.1	Podporované widget buildery . . . . .	31
4.2	Podporované validace . . . . .	35





# 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 zkratk, 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 většiny 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 identická 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, konkrétně pro Android a Windows Phone, což mi přišlo 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 první 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 různých 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 žádné specifické příkazy, jako v případě příkazové řádky, nebo hlasové povely jako v případě hlasového rozhraní. [17].

V softwarových systémech je nejběžnějším 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 uživateli 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 danému 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 stanicí, 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 [26]. 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 [28]. 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 [28].

Jedním z hlavních a kritických aspektů dobrého softwaru jeho udržitelnost, anglicky maintainability. Udržitelnost je schopnost systému se dále měnit a vyvíjet 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 [32]. Bohužel uživatelské rozhraní moc udržitelné 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 mechanismus, který

by automaticky zaručil, že je UI v souladu s modelem [28]. 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[18]. V tomto případě je jedním zařízením server, druhým klientská aplikace a interakcí je myšlen vzájemný přenos dat.

V mobilních aplikacích jsou velmi populární interpretací webových služeb RESTful Web Services využívající Representational 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 [14].

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[31], což jsou formáty obsahující data ve formě párů klíč - hodnota. Zmíněné formy dat vzikají serializací objektů [16], 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 [12]. Dalším způsobem je využití nějaké knihovny, která umí data rovnou deserializovat do objektu. Příkladem takové knihovny je Gson[10].

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 [16] do něj odeslaná data, je nutné upravit i příslušná místa v klientské aplikaci, která zpracování a odesílání uživatelského vstupu mají na starosti.

Demonstrujeme problém na příkladě. Máme server a na něm model například Tým, který obsahuje dva sloupce - název týmu a počet členů. Vytvoříme si klientskou 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 nahlédneme 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 klientské aplikaci, se musí vydat její nová verze. Bohužel v dnešní době je možnost aktualizací neprovést, a to hlavně na mobilních zařízeních, příkladem může být Google Play na Androidu [25]. 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 klientská 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 [22]. 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 AFSwinox společně s AFRest [34], 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 [6], který uvedené problémy řeší v Java EE aplikacích.

#### 2.1.4.1 PHP Database Form

PHP Database Form [15] 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 [5].

#### 2.1.4.2 AspectFaces

AspectFaces je framework, který se snaží o to, aby bylo UI generováno na základě modelu [27], 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 [34]. Framework používá RESTful webové služby pro získání definic komponent, díky kterým je schopen dynamicky postavit formulář či tabulku. Takové definice komponent vznikají za pomoci 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 jednotlivá 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[34]. Framework se snaží o zjednodušení tvorby uživatelských rozhraní hlavně z hlediska množství kódu a udržitelnosti. 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 poskytnout 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 [34]. Tento framework vytváří na straně serveru tzv. definice komponent, které komponentu popisují z hlediska vzhledu, rozložení i obsahu. Jedná se tedy o metadata [13], 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 informace:

- 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 úryvku 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>
  </connection>
  <endPoint>toms-cz.com</endPoint>
  <!-- ... obdobne jako metamodel -->
  <security-params>
    <!-- ... obdobne jako data -->
  </security-params>
</connectionRoot>
```

```
</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 [34] 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.

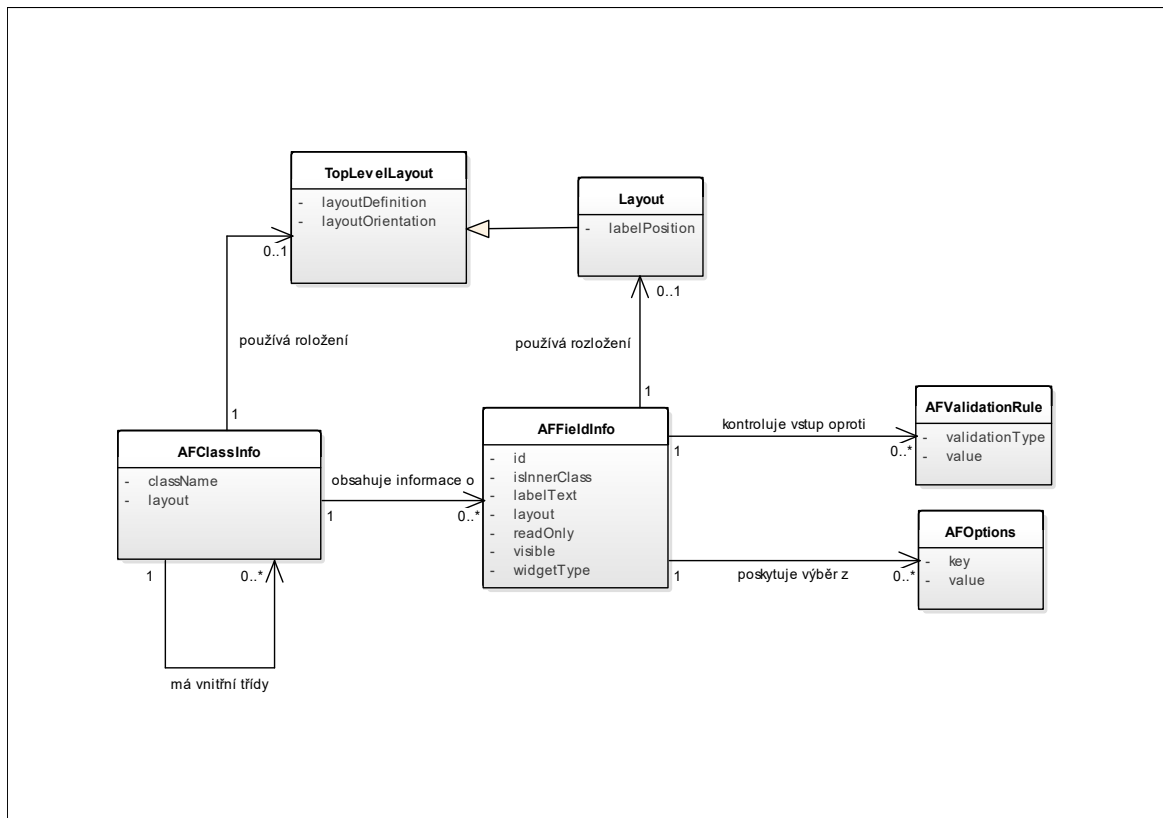
### 3.2.3 Reprezentace metadat ve frameworku

Získaná metadata je potřeba ve frameworku nějak rozumně udržovat. AFSwinx už pro to určitou strukturu definuje [34] a je tedy žádoucí ji opětovně využít. Tuto část systému, která uchová získané informace o komponentě, zachycuje následující doménový model, vytvořeného za pomoci 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 [24]. 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 vnitřních 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 metadatach 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 validačních 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 metadatach a označí ho jako classType, tedy vnitřní třídu, jejíž popis můžeme nalézt v metadatach v části s vnitřními třídami. V rámci zachování správného pořadí vykreslení polí je tedy nutné, aby klientský 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 mít příslušný validátor. 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.

### 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 3.1, 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 [34], 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 [24].

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 obrázku 3.2. Strukturu na obrázku následující část práce blíže specifikuje.

#### 3.2.4.1 AFComponent

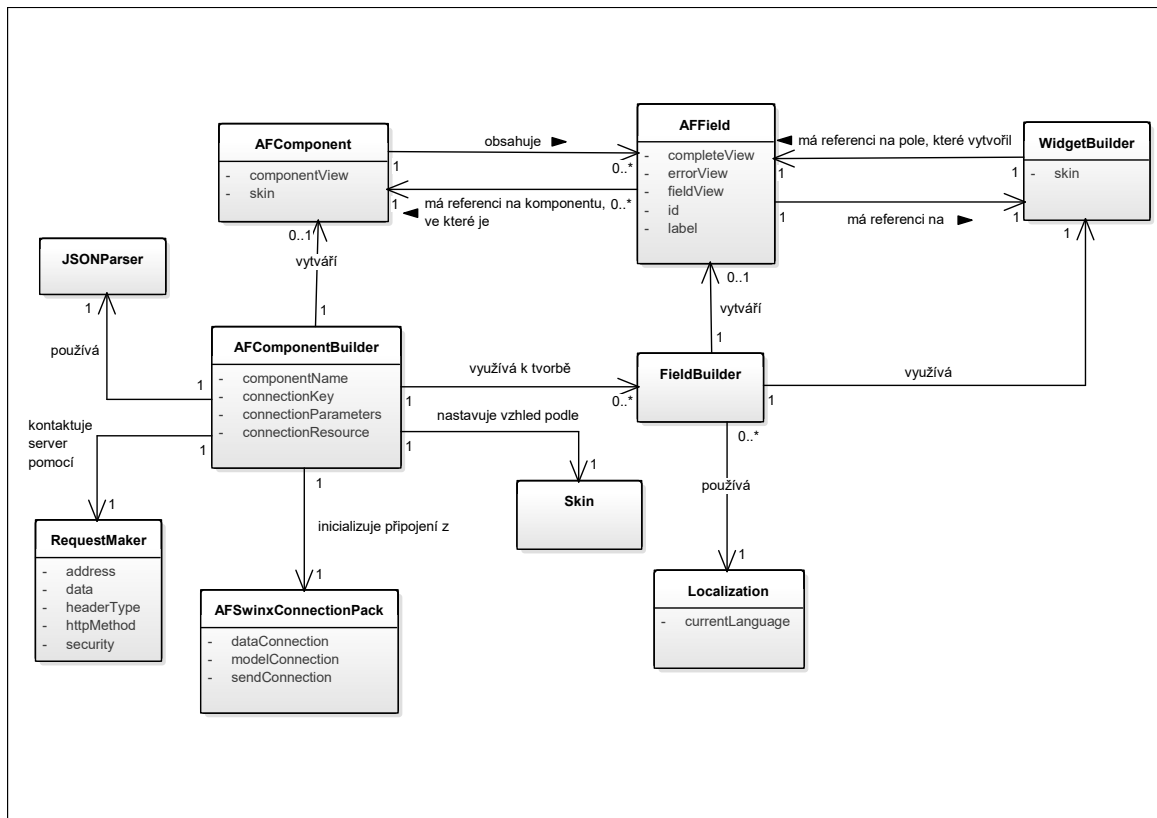
Tato třída zastřešuje vytvořenou komponentu. Jak bylo již zmíněno, frameworky podporují dva typy komponent, a to formulář a list. Grafická reprezentace komponenty je uložena v atributu `componentView`. Také obsahuje `skin`, podle kterého je vzhled komponenty nastaven. Komponenta obsahuje 0 až N tříd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

#### 3.2.4.2 AFField

`AFField` popisuje vytvořenou část komponenty, tedy její pole. Pole má svůj jednoznačný 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 popis 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



Obrázek 3.2: Část doménového modelu systému realizující tvorbu komponenty

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 3.2 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. Uživateli při specifikaci builderu může nastavit 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 mohou být například uživatelské jméno a heslo sloužící pro autorizaci 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 do `AFSwinxConnectionPack`, která je součástí referenčních frameworků `AFSwinx` a `AFRest` [34].



#### 3.2.4.4 RequestMaker a JSONParser

Definovaných připojení pak využije třída RequestMaker, zajišťující komunikaci buildru 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 získání 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 3.2 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 částí, která určuje widget komponenty se využívá WidgetBuilder. V metadatach 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. 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 demonstrovat 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.

Aby však vývojář mohl s komponentou pracovat, musí být schopen ji získat, nejlépe kdekoliv v programu. Proto se vytvořené komponenty musí někde skladovat a být k nim jednoduchý přístup. V případě Androidu je navržena třída AFAndroid, v případě Windows Phone zase třída AFWinPhone. Tyto třídy nabízí mimo získání komponent i možnost vytvoření obou typů builderů a uživateli tak zprostředkovává vlastně vše, co pro vytváření komponent potřebuje.

### 3.3 Případy užití

V této sekci se práce zaměřuje na případy užití, které zachycují návrh možností, které by měly oba mobilní frameworky podporovat. Ke zobrazení těchto možností slouží diagram případů užití, který lze nalézt v příloze B.3. Případy užití určují, jak lze používat systém nebo jeho dílčí části [24]. 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.

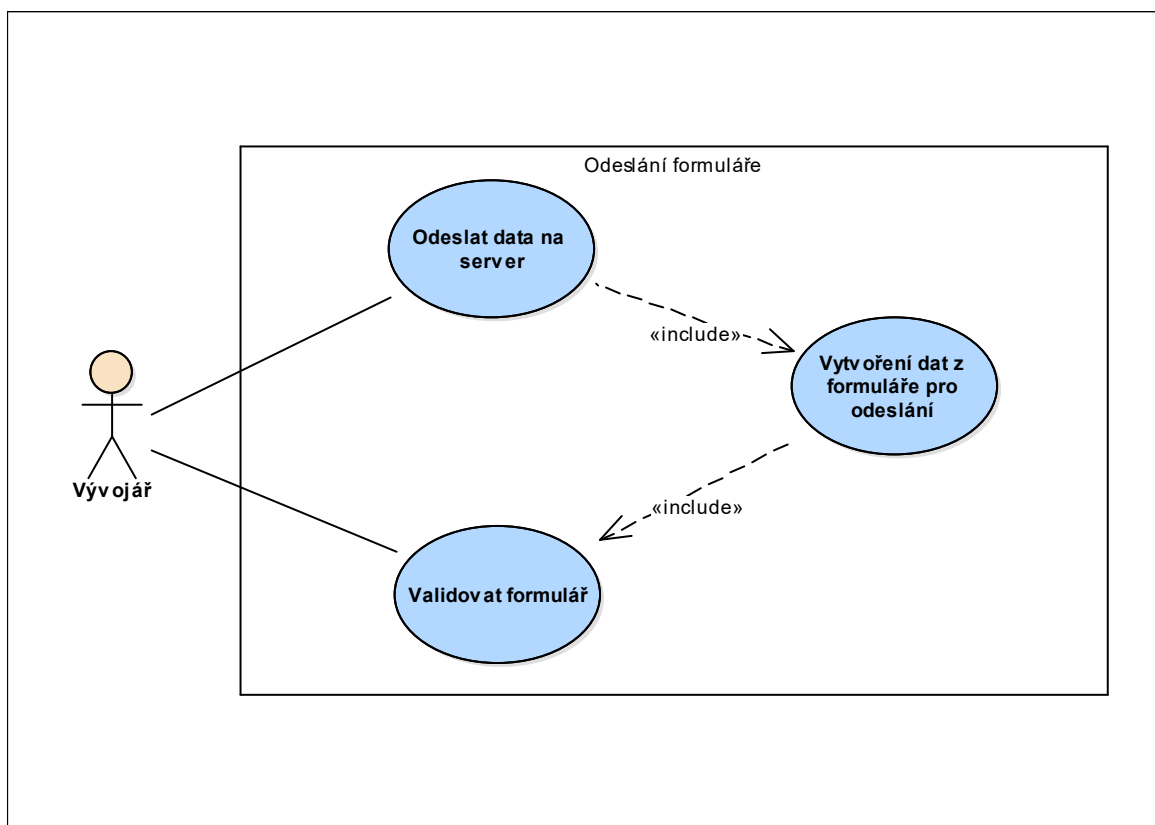
Jednou z hlavních funkcí, kterou musí frameworky podporovat, je odeslání formuláře. Tato funkce je blíže popsána obrázkem 3.3, na kterém jsou zobrazeny potřebné případy užití. Tento obrázek je podrobněji přiblížen popisem a scénářem každého případu užití, který se v něm vyskytuje. Pro stručnost je předpokládán 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.

#### 3.3.1 Případ užití: Validace formuláře

Na obrázku 3.3 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 explicitně, 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.



Obrázek 3.3: Část diagram případů užití pro odeslání formuláře

- Tento formulář musí být vytvořený 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 Příklad 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 [34]. Bylo by tedy dobré, navrhnout tento případ užití pro více možných formátů. V diagramu 3.3 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 vytvořený 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 Příklad užití: Odeslání dat na server

Posledním případem užití z obrázku 3.3 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 vytvořený 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 [4], 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 [21], 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 pomoci 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 dřívejší nebo stejné než druhý atribut určený v parametrech anotace. Anotace má tedy funkci validační pravidla. Tato informace se v metadatech objevuje prozatím pouze u informací o poli s typem widgetu,

jež má být reprezentován jako DatePicker. AspectFaces [6] 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 [34] 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 frameworkech pro mobilní platformy, ale také v již existujícím řešení AFSwinx pro platformu Java SE.

## 3.5 Použité technologie

V této části práce jsou popsány použité technologie. Kromě samotných frameworků jsou součástí práce i příslušné ukázkové projekty.

### 3.5.1 Java a Android SDK

Část frameworku, která je schopná vytvářet formuláře nebo listy a poté s nimi pracovat pro Android aplikace k tomu využívá právě těchto dvou technologií. Zdrojové kódy Android aplikací jsou psány v Javě a ty se pak přeloží pomocí Android SDK do APK souboru spustitelném na mobilním zařízení. Android SDK poskytuje Java knihovny pro dynamické vytváření prvků UI jako jsou komponenty, layouty, dialogy, posluchače akcí atp. Také poskytuje širokou škálu funkcí k práci se souborovým systémem, senzory, sítí, kamerou a mnoho dalšího [1]. Android je na spoustě různorodých zařízeních a vyskytuje se v mnoho různých verzích, proto je třeba zvážit, na jaká zařízení je aplikace mířena. Obecně platí, že čím nižší verze SDK, tím méně poskytovaných funkcí. Při vytváření projektu v IDE Android Studio se vývojář dozví, že pokryje 100% zařízení, pokud cílí svůj projekt alespoň na verzi 10, která odpovídá Android verzi 2.3.3 nebo 2.3.4. s kódem Gingerbread z Únoru 2011. Framework AFAndroid je nastaven na podporu SDK verze 11, neboť využívá pro kontaktování serveru třídu `AsyncTask`, která je dostupná až od této verze. Nicméně framework byl otestován kvůli absenci staršího zařízení až na SDK 18, tedy verzi Android 4.3.3 s kódem Jellybean, na kterém framework funguje bezchybně [3]. Při vývoji této části frameworku bylo využito IDE Android Studio 1.5.1.

### 3.5.2 C# a Windows Phone SDK

Další část práce se věnuje Windows Phone verzi frameworku. Aplikace na Windows Phone je obvykle psaná v jazyce C# [33] za pomoci knihoven z Windows Phone SDK, které obdobně jako Android SDK umožňují dynamické tvoření UI, reagovat na vzniklé události, pracovat s mobilním zařízením atd. Ačkoliv není Windows Phone tolik rozšířen jako Android, taktéž existuje více verzí operačního systému, mezi kterými je potřeba se rozhodnout. Tato práce se zaměřuje na Windows Phone 8.1, protože dle statistik z Března 2016 je nejpoužívanější Windows Phone verzí a starších verzí ubývá. Naopak přibývají uživatelé novější verze Windows

10 Mobile, která by ale neměla mít problém aplikace pro starší telefony spustit [23]. Při vývoji WP části frameworku bylo využito vývojové prostředí Visual Studio 2015 od společnosti Microsoft.

### 3.5.3 AFSwinx a AFRest

AFSwinx a AFRest, které byly již dříve popsány v sekci s existujícími řešeními, jsou napsány v Javě. Jsou k dispozici v podobě JAR souborů, které Android verze frameworku importuje a využívá. Windows Phone verze je použít nemůže, neboť se liší platformou a vše co bylo Android frameworkem využito, musí být do WP přepsáno. Použité části těchto frameworků byly popsány v dřívějších částech této kapitoly, jako například struktura pro uložení metadat na obrázku 3.1.

### 3.5.4 Ukázkové projekty

Ukázkové projekty demonstrují použití obou vytvořených frameworků. Pro splnění tohoto úkolu bylo zapotřebí ještě následujících technologií.

#### 3.5.4.1 AFServer

Jak bylo již zmíněno, oba frameworky AFAndroid i AFWinPhone interpretují určitý popis uživatelského rozhraní, který přichází ze serveru, který takový popis dokáže poskytnout. Tento server byl již vytvořen pro demonstraci funkčnosti frameworků AFSwinx a AFRest a je využit i pro ukázkové projekty v této práci. AFServer je Java EE aplikace, která využívá technologií jako RestEasy, která poskytuje API pro definici RESTful webových služeb nebo například in-memory databázi DerbyDB [34].

#### 3.5.4.2 Glassfish

Glassfish [9] je open-source aplikační server využívaný pro Java EE platformu. Zmiňovaný AFServer je odladěn pro použití právě na Glassfish ve verzi 3 a 4 a proto, když bylo potřeba server při vývoji ukázkových projektů použít, byl spuštěn přes tento aplikační server [34]. Server byl spouštěn při vývoji ukázkových projektů lokálně. Později byl server umístěn na doménu [www.toms-cz.com/AFServer](http://www.toms-cz.com/AFServer), aby mohl být testován i z reálných mobilních zařízení a ne pouze pomocí emulátorů a už nebylo potřeba explicitně Glassfish používat.





## Kapitola 4

# Implementace

### 4.1 Architektura

Jak už bylo popsáno, tato práce se zabývá tvorbou dvou klientských frameworků pro dvě různé mobilní platformy. Ke správnému fungování frameworků je předpokládána serverová část, která generuje data v určité struktuře a formátu, z nichž oba frameworky umí vytvořit a zobrazit koncovému uživateli prvky grafického uživatelského rozhraní. Použití frameworku na jednotlivých zařízeních zachycuje diagram nasazení na obrázku B.4. Diagram nasazení je určen k tomu, aby zobrazil jak je architektura softwaru namapovaná na architekturu hardwaru. Jedná se o diagram, který patří do implementační fáze, avšak často již vzniká určitá první verze ve fázi návrhové a poté se doplňuje [24].

Obrázek B.4 zobrazuje tři zařízení - server, Android klienta a Windows Phone klienta. Pro účely vývoje těchto frameworků byla na serveru nasazena Java EE aplikace AFServer, která za pomoci AFRest [34] využívající AspectFaces [6] generuje definice komponent z modelu, které pak upraví a poskytuje klientovi v požadovaném tvaru, který ji může získat pomocí http dotazu. Android klient interpretuje tuto specifikaci komponenty za pomoci frameworku AFAndroid, který využívá stejných částí jako serverová strana, což zajišťuje kompatibilitu objektů na serverové a klientské straně. Windows Phone klient interpretuje definici za pomoci AFWinPhone, který bohužel neumožňuje využívat stejných částí se serverem neboť běží na rozdílných platformách. Aby bylo zachováno stejné chování jako u Android klienta, byly tyto části znovu vytvořeny.

Aby klienti mohli framework využívat, musí ho nejdříve vložit. V případě Androidu se framework kompiluje do AAR souboru, který lze do projektů přidat jako Gradle závislost. Gradle je systém pro build projektů a správu závislostí nejen v Androidu. V současné době ho lze využít i v C, C++, Java aplikacích [30]. Závislost může být lokální nebo se stahovat z online repozitáře. Jelikož není AFAndroid ještě k dispozici v žádném z online repozitářů, je nutné ho přidat lokálně. To lze udělat přidáním AAR souboru do složky lib nebo vytvořit z knihovny nový modul. Windows Phone framework je zase zkompileován do DLL souboru, který je v současné chvíli nutné také přidat lokálně jako knihovnu mezi reference.

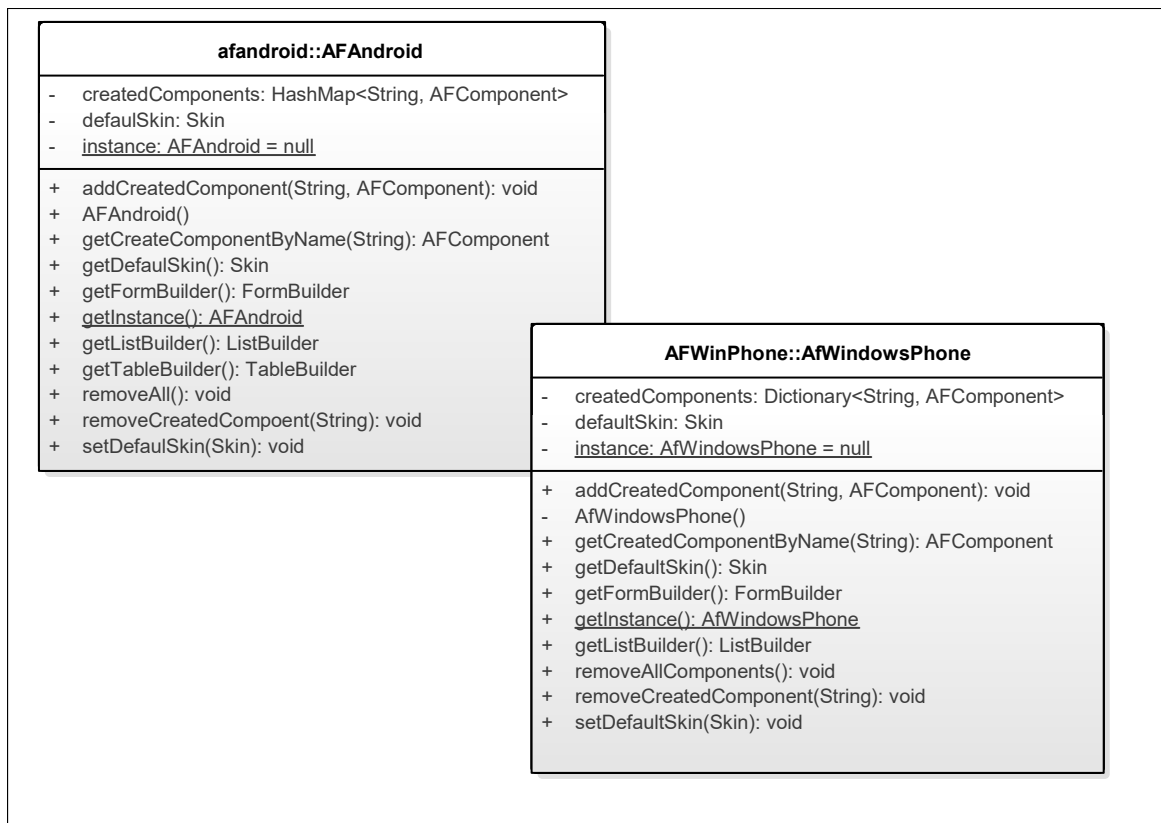
### 4.1.1 Komponenty

Oba frameworky jsou schopny generovat dva typy komponent - formulář a list. Obě tyto komponenty dědí od třídy `AFCComponent`, která představuje společnou část obou komponent. `AFCComponent` implementuje rozhraní třídy `AbstractComponent`, která definuje metody pro získání definice komponenty, naplnění daty, generování dat pro odeslání a jejich odeslání na server včetně validace. Tyto metody musí `AFCComponent` nebo některá třída, která od něj dědí, nuceně implementovat. Společná část komponent konkrétně implementuje metody pro získání definice ze serveru a získání dat, kterými se má komponenta naplnit, protože tyto části jsou jak pro formulář, tak pro list identické. Způsob reprezentace komponenty a vložení získaných dat, pak řeší jednotlivé komponenty každá jiným způsobem. Stejně tak list nedisponuje stejnou funkcionalitou jako formulář, je jen pro čtení a proto nepodporuje metody pro generování a následné odesílání dat či jejich validaci. Struktura části Android frameworku zachycující komponenty je na obrázku [B.5](#) popsána diagramem tříd. Diagram pro tuto část ve Windows Phone frameworku je až na syntaktické odlišnosti shodný.

Obě komponenty obsahují metodu, která získá jejich grafickou reprezentaci, kterou pak může vývojář vložit do libovolné části uživatelského rozhraní, neboť tato metoda má návratový typ u Android verze `View` a u WP verze `FrameworkElement`, což jsou jedny ze základních prvků GUI na těchto platformách a lze je vložit do jakéhokoliv jiného elementu. V AFSwinx [\[34\]](#) komponenty rovnou dědily od třídy `JPanel` a tudíž metodu pro získání grafické reprezentace nepotřebovaly. Tento přístup však vyústil v to, že se metody poskytované komponentou mísily s nepřehledným množstvím metod příslušících třídě `JPanel`, což způsobovalo dle uživatelského testu nepřehlednost a zhoršenou orientaci v poskytovaných metodách, proto byl zvolen výše zmíněný přístup.

Vytvářené komponenty se ve frameworku určitým způsobem ukládají, aby bylo možné je získat a pracovat s nimi v celém programu, ne jen v místě, kde byly vytvořeny. Konkrétně se komponenty skladují ve třídách `AFAndroid` pro Android a `AfWindowsPhone` pro WP, zobrazené na obrázku [4.1](#). Tyto třídy jsou implementovány jako singleton, což je návrhový vzor, který se využívá, když je potřeba mít pouze jednu instanci této třídy, ke které lze přistupovat z více míst [\[29\]](#). Singleton často bývá součástí jiných návrhových vzorů jako je například Facade neboli fasáda. Fasáda se používá v případě, že programátor chce poskytnout jednoduché rozhraní pro ovládání složitějšího systému a tím klienty odstínit od vnitřní implementace systému schovaného pod fasádou [\[29\]](#). `AFAndroid` a `AFWinPhone` tedy neslouží pouze jako sklad vytvořených komponent ale také jako fasády pro ovládání frameworků. Nabízí získání vytvořené komponenty, jejich smazání, získání builderů pro tvorbu komponent a nastavení základního skinu, který se použije při sestavování komponent, není-li v při inicializaci builderu specifikováno jinak. Proces sestavení komponenty je složitější proces několika funkcí, které na sebe musí navazovat ve správném pořadí, proto kdyby tyto třídy neexistovaly, nebylo by použití frameworků vůbec snadné.

Příklad vytvoření komponenty v Android frameworku, konkrétně formuláře, za použití fasády `AFAndroid` je v ukázce kódu [4.1](#). Kód ukazuje, že je třeba za použití fasády získat builder, který je pak nutné nainicializovat. Inicializace zahrnuje určení zdroje, ze kterého se má definice komponenty získat, definuje `InputStream` s načteným souborem s definicí zdrojů a klíč, pod kterým se získá konkrétní připojení. Soubor byl popsán v rámci analýzy a jeho struktura je ukázána v části kódu [3.1](#). Vývojář si komponentu také při inicializaci



Obrázek 4.1: Třídy AFAndroid a AfWindowsPhone sloužící jako fasády pro ovládání frameworků

builderu pojmenuje, tedy přiřadí ji jednoznačný textový řetězec, pod kterým se komponenta uloží do vytvořených komponent a na základě tohoto řetězce ji lze odtud skrz fasádu opět získat. Existuje ještě přetížená metoda pro inicializaci builderu, která je umožňuje přidat dodatečná nastavení pro připojení. Po inicializaci builderu lze nadefinovat ještě skin, který se při vytváření komponenty použije. Jelikož je komponenta vytvářena dynamicky včetně jejích vlastností a vzhledu v rámci metody, která je pro vývojáře zapouzdřená, jediným způsobem jak pohodlně upravit její vzhled je nastavit skin před zahájením její tvorby. Jelikož má vývojář k některým částem komponenty přístup i po jejím vytvoření, lze vzhled komponenty upravit i později, vyžaduje to však znalosti tvorby GUI pro danou platformu, což už je složitější záležitost. Po nastavení skinu lze komponentu vytvořit a dále pak nad ní provádět akce, které poskytuje. Pro Windows Phone je kód identický, pouze se používá WP fasáda AfWindowsPhone a místo InputStreamu stačí nadefinovat jméno souboru se zdroji, respektive cesta k němu.

Pokud se něco při vytváření nepovede, například se nelze připojit ke zdroji na serveru, je vyhazována výjimka. Vývojář může výjimku libovolně zpracovat, například zobrazit dialog s chybou.

Listing 4.1: Ukázka tvorby formuláře

```
InputStream connectionResource = getResources().openRawResource(R.raw.connection);
```

```
try {
    AForm form =
        AFAndroid.getInstance().getFormBuilder()
            .initBuilder(getActivity(), LOGIN_FORM_NAME, connectionResource,
                LOGIN_FORM_CONNECTION_KEY)
            .setSkin(new LoginSkin(getActivity()))
            .createComponent();
} catch (Exception e) {
    zpracovani_vyjimky, ktera_mohla_nastat_pri_vytvareni_komponenty
}
}
```

## 4.2 Komunikace serveru a klienta

Jak už bylo zmíněno v analýze, server poskytuje zdroje, ze kterých lze získat definice komponent, data nebo na ně lze odeslat uživatelský vstup. K nadefinování těchto zdrojů je potřeba vytvořit XML soubor, který je popsán v ukázce kódu 3.1. Tento soubor se zpracuje pomocí příslušného XML parseru a výsledkem je třída AFSwinxConnectionPack, která má v sobě uloženy jednotlivé části daného připojení, konkrétně, kde je uložena definice komponenty, data a kam lze odeslat uživatelský vstup [34]. V průběhu tvorby komponenty se tyto části připojení využijí k získání informací, které jsou na definovaných zdrojích, na něž připojení odkazuje, uloženy. Získání informací je provedeno pomocí HTTP požadavku, který vytváří a odesílá třída RequestMaker. Způsob provedení tohoto požadavku je na obou platformách dosti rozdílný, avšak něco mají společného a to, že musí být asynchronní. V mobilních aplikacích se requesty musí provádět asynchronně na jiném vlákne než na hlavním, neboť hlavní vlákno spravuje celé UI aplikace a to by v momentě, kdy se požadavek provádí, zamrzlo, což je nežádoucí. HTTP požadavek v případě, že se získává definice komponenty nebo data, vrací textový řetězec, který obsahuje JSON objekt převedení na textovou reprezentaci, v případě odesílání na server se akce pouze provede.

V Androidu se standardně HTTP requesty řeší v rámci objektu typu AsyncTask [7], který umožňuje provést akci v pozadí a výsledky akce předat hlavnímu UI vláknu. Tím se vývojáři Android aplikací vyhnou přímé práci s vlákny. Třída RequestMaker od AsyncTask dědí a přetěžuje metodu pro práci v pozadí, kde se celý požadavek vytvoří, odešle a získá se odpověď. Tvorbu a práci s požadavkem má na starosti třída HttpURLConnection, před Android 6.0 by bylo možné ještě použít Apache HttpClient, který byl s příchodem nové verze Androidu smazán. Lze ho po určitém nastavení nadále používat ale ukazuje se, že HttpURLConnection je efektivnější z hlediska využití sítě a spotřeby energie [11].

Specifikace požadavku, tj. http metoda, url adresa, data, content-type a případně bezpečnostní omezení, se předávají třídě v konstruktoru. Během procesu se může stát výjimka a jelikož metoda, která pracuje na pozadí nemůže zpropagovat výjimku o úroveň výš, je nutné výjimky řešit v rámci zmíněné metody. Je ale žádoucí ponechat řešení těchto výjimek na vývojáři, proto byla zavedena alternativa, ve které metoda běžící na pozadí vrací Object, který buď může být textový řetězec s daty ze serveru nebo vzniklá výjimka. O úroveň výše tento fakt kontroluje a v případě, že se jedná o výjimku, je vyhozena a zpropagována až k vývojáři. Po vytvoření objektu RequestMaker je třeba ho spustit. K tomu slouží metoda executeOnExecutor, která umožňuje spustit v jednu dobu více objektů typu AsyncTask.

WindowPhone verze používá k tvorbě a manipulaci s požadavkem třídu `HttpClient` [20] v rámci asynchronní metody, která request provádí. V tomto případě problém s propagací případné výjimky až k vývojáři. Nastavení požadavku se provádí přes konstruktor obdobně jako v Android verzi.

### 4.2.1 Generování komponent

Na základě definice komponenty, která se ze serveru získá výše uvedeným způsobem, lze generovat komponenty. Typ komponenty, která se vygeneruje závisí na zvoleném builderu. Každý builder je reprezentován třídou, která dědí od abstraktní třídy `AFCComponentBuilder`. Tato třída definuje společné vlastnosti všech builderů a vynucuje implementaci metody pro vytvoření komponenty jako takové a vytvoření její grafické reprezentace. Proces vytvoření komponenty v Android frameworku je popsán na sekvenčním diagramu v příloze B.8. Proces tvorby ve WP verzi je obdobný s tím rozdílem, že není potřeba znát k vytváření GUI prvků kontext, proto nemusí být do argumentů funkcí předávána aktivita, ve které chceme komponentu vytvořit. Z obrázku je patrné, že po inicializaci builderu a případném nastavení skinu začíná proces tvorby formuláře na jehož konci uživatel získá již hotový formulář. Nejprve dojde k získání metamodelu komponenty ze serveru. Poté začne tvorba jednotlivých polí formuláře, k tomu je zapotřebí odpověď s metamodelem rozparsovat, výsledkem čehož bude třída `AFCClassInfo` obsahující informace o celé komponentě. Tato struktura byla v analytické části popsána na obrázku 3.1, skutečná struktura v tomto případě pro Windows Phone framework je na obrázku B.7. `AFCClassInfo` obsahuje 0 až N objektů třídy `FieldInfo`, tedy objektu držící informace o tvořeném poli. Přes tyto objekty framework iteruje a postupně z nich vytváří objekty třídy `AFField`, které už obsahují i grafickou reprezentaci pole.

Při iterování skrze popisy polí je důležité kontrolovat atribut, který určuje zda pole zastupuje v modelu na serveru primitivní nebo složený datový typ. V případě, že se jedná o složený datový typ, vyhledá se k němu v `AFCClassInfo` příslušná definice vnitřní třídy, na kterou se pak funkce, která tvoří jednotlivá pole, rekurzivně zavolá a na místo pole označeného jako vnitřní třída budou vytvořena pole, která jsou nadefinovaná v příslušné vnitřní třídě.

Součástí tvorby pole není jen nadefinování vlastností a tvorba samotné grafické reprezentace. Při tvorbě formulářových polí je potřeba vytvořit tři elementy, ze kterých se pole skládá - label, místo pro zobrazení validačních chyb a aktivní prvek. To zajišťuje třída `FieldBuilder`. Vytvořit label a místo pro validační chyby je jednoduché, aktivní prvek je mírně komplikovanější. Součástí třídy `FieldInfo` je i atribut `widgetType`, který určuje, jakým typem aktivního prvku má být pole reprezentováno. Na základě tohoto atributu se získává pomocí třídy `WidgetBuilderFactory` příslušný builder, který umí přímo příslušný aktivní prvek vytvořit. Tento builder poskytuje také funkcionalitu pro získání a nastavení dat vytvořenému aktivnímu prvku. Pro jednodušší přístup k této funkcionalitě, má pole formuláře na builder, který mu vytvořil část s aktivním prvkem, referenci. Po vytvoření všech tří částí se zkompletují do celkové grafické reprezentace, která se poli taktéž nastaví. Takovéto pole se vrátí builderu komponenty, který si jeho existenci zaznamená. Po zaznamenání všech polí, které mají ve formuláři být, se jednotlivé grafické reprezentace polí seskupují do celkového vzhledu komponenty, který se pak komponentě nastaví a vývojář ho potom může z komponenty získat.

#### 4.2.1.1 Naplnění komponenty daty

Pokud má být komponenta naplněna daty, je nutné získat jednotlivé buildery, které vytvořily jednotlivá pole. Data se vkládají přímo z odpovědi serveru, která je obsahuje. V datech se objevuje vždy identifikátor daného pole a hodnota, která má být vložena. Dle tohoto identifikátoru je framework shopen vyhledat příslušné pole a z něj získat builder, který postavil jeho aktivní část. Jak už bylo řečeno, builder disponuje funkcí schopnou do pole, které vytvořil, vložit data, což také provede. Takto se to provede u všech polí, která se v přijatých datech vyskytují. Tím je proces tvorby formuláře popsán na obrázku [B.8](#) ukončen a vývojáři je předána vytvořená komponenta.

#### 4.2.1.2 Widget buildery

Bylo zmíněno, že části formulářových polí, do kterých lze zadat uživatelský vstup, se vytváří pomocí příslušných builderů, jež lze získat pomocí třídy `WidgetBuilderFactory`. Tato třída využívá návrhový vzor `Factory`. Ten se využívá, když je potřeba vytvořit objekt a zároveň zastínit logiku vytváření objektu klientovi [8]. Tento návrhový vzor byl ve frameworku použitý proto, aby se v případě, že by bylo potřeba přidat nový builder, nezasahovalo do logiky vytváření polí, ale pouze do třídy `WidgetBuilderFactory`. Jednotlivé widget buildery lze vidět na obrázku [B.6](#) a jejich seznam je v tabulce [4.1](#). Každý builder dědí od základního builderu definovaného abstraktní třídou `BasicBuilder` implementující rozhraní `AbstractWidgetBuilder`. `AbstractWidgetBuilder` definuje metody pro vytvoření grafické reprezentace widgetu, nastavení dat do widgetu a získání dat z widgetu. Tyto metody musí být pak každým novým builderem implementovány. Ukázka kódu [4.2](#) zobrazuje tvorbu grafické reprezentace textového pole v Android frameworku.

Listing 4.2: Ukázka tvorby grafické reprezentace textového pole

```
@Override
public View buildFieldView(Activity activity) {
    EditText text = new EditText(activity);
    text.setTextColor(getSkin().getFieldColor());
    text.setTypeface(getSkin().getFieldFont());
    addInputType(text, getField().getFieldInfo().getWidgetType());
    if (getField().getFieldInfo().getReadOnly()) {
        text.setInputType(InputType.TYPE_NULL);
        text.setTextColor(Color.LTGRAY);
    }
    return text;
}
```

V ukázce [4.2](#) lze vidět, že na vstupní pole je aplikován skin, který upravuje jeho vzhled. Konkrétně v případě textového pole se nastavuje barva jeho textu a typ písma.

#### 4.2.1.3 Skiny

Skiny neupravují pouze vzhled aktivního prvku, ale také vzhled labelu, validačních chyb nebo i celé komponenty. Lze nastavit grafické vlastnosti jako je barva textu, pozadí, velikost textu a jeho font, ale také rozměry celé komponenty. Ve frameworku existuje rozhraní `Skin`, které definuje všechny nastavitelné vlastnosti. Už bylo zmíněno, že skiny se musí nastavit

Tabulka 4.1: Podporované widget buildery

Builder	Typ widgetu	Popis
DateWidgetBuilder	Calendar	Používá se při reprezentaci atributů typu datum. Umožní uživateli zobrazit date picker, pomocí kterého lze vybrat datum.
DropDownWidgetBuilder	dropDownMenu	Menu, ze kterého lze vybrat jednu z několika voleb.
CheckBoxWidgetBuilder	checkBox	Zaškrťovací políčko reprezentující hodnotu true nebo false podle toho, zda je nebo není zaškrtnuto.
TextWidgetBuilder	textField	Builder pro textové pole. Lze mu nastavit typ vstupu, podle kterého se změní klávesnice pro zadávání znaků například na číselnou, tedy používá si i pro widget typy NUMBERFIELD a NUMBERDOUBLEFIELD. V Android frameworku lze použít i pro pole s heslem.
PasswordWidgetBuilder	password	Pole pro hesla. Text, který uživatel dovnitř napíše je převeden na zástupné znaky. Tento builder se vyskytuje pouze ve WP verzi frameworku, Android verze k tomuto účelu používá TextWidgetBuilder
OptionBuilder	option	Vytvoří skupinu radiobuttonů, z které lze vybrat jednu hodnotu. Pokud nejsou definovány možnosti vytvoří se 2 radiobuttony s hodnotami ano a ne

před tvořením komponenty jejímu builderu. Toto nastavení je ale volitelné a tak existuje třída DefaultSkin, která toto rozhraní implementuje a definuje základní vzhled, který se použije nespécifikuje-li vývojář jinak. Vývojář pravděpodobně nebude chtít předělat úplně celý vzhled, ale pouze některé části skinu. K tomuto mu stačí dědit od třídy DefaultSkin a pouze přetížít metody udávající vlastnosti, které chce změnit.

#### 4.2.1.4 Layouty

Při vytváření grafické reprezentace komponenty, server definuje, jak se její jednotlivé části v rámci komponenty uspořádají. V případě formuláře je nutné uspořádat pole tak, aby byl formulář přehledný a dobře použitelný. V případě listu jde hlavně o přehlednost, neboť jednotlivé položky listu by byly při větším obsahu moc vysoké a nevzhledné. Server definuje osu, podle které jsou komponenty vykreslovány, počet sloupců a pozici labelu [34]. Layout lze určit buď celé komponentě, ten na obrázku B.7 reprezentován třídou TopLevelLayout nebo pouze části komponenty tj. poli, který je na obrázku reprezentován třídou Layout. V případě layoutu komponenty je pouze definována osa a počet sloupců, layout jednotlivých

polí přidává pozici labelu. Hodnoty, kterých jednotlivé vlastnosti nabývají jsou popsány pomocí následujících výčtových typů, které lze vidět i na obrázku [B.7](#).

- `LayoutDefinitions`, který určuje počet sloupců a nabývá hodnot `ONECOLUMNLAYOUT` a `TWOCOLUMNSLAYOUT`, tedy jednosloupcové a dvousloupcové rozložení.
- `LayoutOrientation`, který určuje osu, ve směru které se části komponenty vykreslují. Nabývá hodnot `AXISX` a `AXISY`, první určuje směr vykreslování podle osy X, druhá podle osy Y.
- `LabelPosition`, který určuje pozici labelu vzhledem k aktivnímu prvku. Nabývá hodnot `BEFORE`, `AFTER` a `NONE`. První znamená, že bude label před aktivním prvkem, druhá za aktivním prvkem a třetí znamená, že label nebude vůbec zobrazen.

Layouty jsou v jednotlivých klientských frameworkcích interpretovány různě. Zatímco v případě Android frameworku je layout realizován pomocí seskupení `LinearLayout`ů [\[2\]](#), WP verze používá k tvorbě layoutu `Grid` [\[19\]](#). Důvodem není nic zvláštního, k oběma třídám existují na druhé platformě ekvivalenty, pomocí kterých lze dosáhnout stejných výsledků, jde spíše o to, ukázat více možných způsobů implementace.

Důležitým aspektem je také pořadí jednotlivých částí komponenty. To definuje server a v metadatech se informace o polích objeví už v daném pořadí, které je třeba zachovat. Proto bylo nutné použít k uskladnění kolekci, která zachovává pořadí svých prvků, takovou je například `List` [\[34\]](#).

#### 4.2.1.5 Lokalizace

Jelikož se mohou v metadatech objevit i texty pro překlad, bylo třeba naimplementovat třídu `Localization`, která se stará o překlady těchto textů. Třída disponuje samostatnou funkcí pro překlad a funkcí pro změnu jazyku. V Androidu se typicky lokalizační texty umísťují do souboru `strings.xml` ve složce `values`. Pro Windows Phone je to obdobné. Zde se texty vkládají do souborů `Resources.resw`, které jsou umístěny ve složce s názvem jazyku ve složce `Strings`. S tímto umístěním textů oba frameworky počítají, u Android verze je sice možnost nadefinovat jiný balíček, ve kterém by se texty měly hledat, ale stále se předpokládá umístění ve `values/strings.xml`. Pokud není překlad nalezen, je vrácena hodnota klíče, který se hledal. Třída `Localization` se dá využít i pro překlad jiných textů i mimo framework a to i v rámci XML šablon, ve kterých je vytvořeno statické UI. Při startu aplikace se použije jazyk zařízení. Pokud by pro daný jazyk neexistovaly předklady, je vhodné mít defaultní soubory s texty, které se v tomto případě použijí.

#### 4.2.2 Práce s komponentami

Vytvořená komponenta disponuje různými funkcemi, které umožňují s ní dále pracovat. Pro formulář jsou to metody pro odeslání, validaci, resetování nebo vyčištění formuláře. `List` je komponenta, která by měla pouze zobrazovat data, proto s ní nejde provést ani moc akcí. Obsahuje však důležitou funkci, pomocí které lze získat obsah jedné položky listu, který lze pak využít například pro naplnění formuláře.



#### 4.2.2.1 Odeslání, reset a vyčištění formuláře

Nejdůležitější funkcí je však odeslání formuláře, který je zobrazen na sekvenčním diagramu B.9. K tomuto účelu má formulář metodu `sendData`. V rámci této metody se nejdříve zjistí, zda vývojář pro komponentu nadefinoval v XML souboru s připojeními část označenou uzlem `<send>`, který definuje, kam se budou data odesílat a jehož struktura lze vidět v ukázce kódu 3.1. Komponenty mají atribut `connectionPack` typu `AFSwinxConnectionPack`, který obsahuje všechna definovaná připojení. V případě, že vývojář specifikoval uzel `<send>` se v tomto atributu formuláře objeví i připojení na zdroj, který přijímá uživatelský vstup.

Pokud připojení nadefinované není vyhodí se výjimka, která je propagována k vývojáři, který s ní dle svého uvážení naloží. Pokud připojení existuje, vygenerují se z formuláře data ve formátu, které server dokáže zpracovat. Klient nezná přesně objekty, které se odeslání na serveru vytvoří či upraví, zná ale jejich strukturu, která stačí k tomu, aby byla vytvořena data, která server dokáže přijmout. V rámci generování dat proběhne nejdříve jejich validace, která je blíže popsána v sekci níže. Pokud data nejsou validní vrací metoda `sendData` hodnotu `false`, která určuje, že se odesílání nezdařilo. Samotný proces generování dat byl převzat z frameworku `AFSwinx`. Ten využívá pro znovupostavení dat objekt typu `BaseRestBuilder` disponující metodou `reserialize`. [34].

`BaseRestBuilder` je implementován prozatím jen třídou `JSONBuilder`, která dokáže z formuláře vytvořit JSON soubor a použije se v případě, že server požaduje JSON formát. Další formáty nejsou prozatím podporovány. Metoda `reserialize`, kterou builder dat používá, již očekává vyparsovaná data z komponenty uložená v objektu typu `AFDataHolder`. Vytvoření těchto dat provádí komponenta sama. Uživatelský vstup získává z aktivní prvků, ke kterým má skrz kolekci polí typu `AFField` přístup. Data se získávají z aktivního prvku pomocí widget builderu, který aktivní prvek vytvořil a na který má `AFField` referenci. Každý `AFField` má svůj unikátní identifikátor, který lze využít pro zjištění pozice proměnné v objektu na serveru, který má odeslání formuláře ovlivnit. Pokud identifikátor obsahuje tečku, znamená to, že dané pole je částí reprezentace nepřimitivního datového typu, tedy vnitřní třídy a k té je třeba hodnotu proměnné přidat. Pokud v identifikátoru tečka není, znamená to, že jsme na správném místě a do mapy `AFDataHolder` se přidá nová proměnná s hodnotou [34]. Android verze využívá již naimplementovaný JSON Builder, který využívá k sestavení dat framework GSON [10]. Ve Windows Phone verzi musel být builder dat přepsán a využívá Windows knihovnu pro práci s JSON soubory. Po vytvoření dat stačí data odeslat na definovaný zdroj na serveru pomocí třídy `RequestMaker`, která byla popsána výše. Jak už bylo zmíněno, v rámci procesu odeslání může dojít k výjimkám, které jsou předány vývojáři ke zpracování.

Odeslání formuláře je tedy jednoduché, stačí definovat zdroj v XML souboru a pak například jako reakci na stisknutí tlačítka zavolat metodu `sendData`. Dále je nutné, aby aplikace nespadla, ošetřit výjimky, které mohou při odesílání nastat. V případě, že nenastane žádná výjimka, data jsou validní a povede se je z formuláře vygenerovat, může vývojář navázat na odeslání další akcí. Vývojáři je tedy celý proces odeslání schován a nemusí se jím vůbec zabývat, díky čemuž je použití frameworku snadné. Nevýhodou však je, že je vývojář omezen na použití nabízených funkcí frameworku a nemůže do procesu odeslání nijak zasahovat [34].

Dalšími funkcemi, které formulář nabízí je resetování a vyčištění formuláře. Reset formuláře provede obnovení aktuálních dat ve formuláři. Pokud je tedy formulář naplněn nějakými daty a uživatel data změní, lze se k původním nezměněným datům před odesláním formuláře

vrátit. Vyčištění formuláře pak nastaví aktivní prvky formuláře na prázdné.

#### 4.2.2.2 Validace a Validátory

Formulář se před odesláním dat na server validuje. Pro každé pole ve formuláři může být nadefinováno 0 až  $N$  validačních pravidel, které vznikají na základě datových nebo bussiness omezení. Hlavním účelem validací je zajistit, aby byla tato omezení splněna a nedošlo při odeslání k chybě. Chyba by například nastala, pokud server očekává pouze celočíselnou hodnotu a přijde mu místo ní textový řetězec. Na Android platformě existuje vůči některým chybám určitá prevence. Příkladem je pole, které vzniklo z atributu typu integer, jež má být reprezentováno typem widgetu Numberfield. V takovém případě framework použije widget Textfield a určí, že lze zadat pouze čísla, změní se tedy nabízená klávesnice, na které uživatel může zadat opravdu jen čísla od 0 do 9 a navíc do pole nelze zkopírovat nic jiného než čísla. WP také změní typ klávesnice, už však dovoluje do pole zkopírovat jiné znaky a nemá jako Android různé klávesnice pro celé a desetinné číslo.

Ke každému pravidlu existuje validátor, který umí pravidlo prověřit. Tabulka 4.2 zobrazuje seznam podporovaných validátorů, ke kterým pravidlům patří a jejich stručný popis. Proces validování formuláře byl již zobrazen na obrázku B.9. Nejprve se projdou všechna pole formuláře, což jsou objekty typu AField. AField obsahuje pravidla, což jsou objekty typu AFValidationRule, které je pro každé pole nutno proiterovat, vyhledat k nim příslušný validátor a provést validaci. Získání příslušného validátoru probíhá pomocí tovární třídy ValidatorFactory, podobně jako získání konkrétního widget builderu, který byl popsán dříve. Validátory implementují rozhraní AFValidator, které disponuje metodou validate, která vrací boolean určující, zda validace proběhla úspěšně či ne. Této metodě je předáván StringBuilder, obsahující chybové hlášky pro dané pole, tedy každé pole má svůj StringBuilder. Pokud alespoň jedna z validací selže, vytvoří se label, do kterého se nastaví hlášky ze StringBuilderu a tento label se uživateli zobrazí. Vývojář má možnost upravit chybové hlášky, které se zobrazují uživateli. V tabulce 4.2 lze v popisu nalézt klíče jednotlivých chybových hlášek, které je potřeba vývojářem přeložit.

### 4.3 Úprava AFSwinx a AFRest

V rámci úpravy existujících frameworků AFSwinx a AFRest [34] bylo požadováno umožnit porovnání dvou polí s datumy ve smyslu menší rovno než, které se použije v případě, že mají v aplikaci existovat dvě pole s datumy, z nichž jedno může obsahovat pouze dřívější nebo stejné datum než datum v druhém poli. Proto bylo vytvořeno validační pravidlo LESSTHAN, které lze najít v tabulce 4.2. K vytvoření tohoto pravidla bylo nutné upravit nejen zmíněné frameworky, ale také serverovou část aplikace. Pro realizaci pravidla bylo rozhodnuto vytvořit novou anotaci @UILessThan, která lze přidat k atributům entity typu datum na serveru. V anotaci je nutno definovat identifikátor druhého pole, se kterým se má pole označené anotací porovnávat. K vytvoření anotace bylo dle dokumentace AspectFaces [6] potřeba vytvořit v AFRest nový anotační deskriptor a zaregistrovat ho v konfiguračním souboru aspectfaces-config.xml na serveru. Tím se zajistí, že si anotace všimne inspekce modelu. Aby se však anotace projevila i v metadatech, která přichází ze serveru, navíc ještě v

Tabulka 4.2: Podporované validace

Validátor	Pravidlo	Popis
RequiredValidator	REQUIRED	Použije se, pokud má být pole povinné. Pokud je pole prázdné, vypisuje hlášku, která vzniká přeložením klíče validation.required.
MaxValueValidator	MAX	Používá se na číselná pole, které mají svou hodnotu omezenou shora nějakým číslem. Pokud hodnota překročí definované číslo, vypisuje hlášku, která vzniká přeložením klíče validation.maxval.
MinValueValidator	MIN	Používá se na číselná pole, které mají svou hodnotu omezenou zdole nějakým číslem. Pokud je hodnota menší než definované číslo, vypisuje hlášku, která vzniká přeložením klíče validation.minval.
MaxCharsValidator	MAXLENGTH	Použije se, pokud počet znaků v poli překročí danou hodnotu. Pokud dojde k překročení, vypisuje hlášku, která vzniká přeložením klíče validation.maxchars.
NumberValidator	Pole vzniklo z atributu typu integer nebo double	Kontroluje, zda jsou vloženy celočíselné hodnoty, pro integer nebo desetinná čísla pro double. V případě, že má být v poli celé číslo a je tam něco jiného, vypisuje se hláška, která vzniká přeložením klíče validation.integer. V případě double je to klíč validation.double. Tento validátor bylo nutné použít pouze ve WP frameworku, neboť Android disponuje prevencí, která byla popsána výše.
LessThanValidator	LESSTHAN	Porovnává dvě pole s daty. Pokud v poli, které obsahuje toto pravidlo není menší hodnota než v poli, jehož identifikátor je definován hodnotou tohoto pravidla, vypisuje se hláška, která vzniká přeložením klíče validation.less-than.

části s validačními pravidly, bylo nutné ještě upravit šablonu date.xml, která se používá při generování metadat o atributu typu datum. Upravená šablona je v ukázce kódu 4.3.

Listing 4.3: Upravená šablona date.xml

```
<widget>
  <widgetType>calendar </widgetType>
  <fieldName>$field$ </fieldName>
```

```
<label>$label$ </label>
<readonly>false </readonly>
<validations>
    <required>$required$ </required>
    <lessthan>$lessthan$ </lessthan>
</validations>
<fieldLayout>
    <layoutOrientation>$layoutOrientation$ </layoutOrientation>
    <labelPosition>$labelPosition$ </labelPosition>
    <layout>$layout$ </layout>
</fieldLayout>
</widget>
```

Další úprava se týkala výčtového typu frameworku AFRest, který obsahuje podporované validace. K validaci bylo nutné vytvořit validátory ve všech existujících klientských frameworkích, tedy nejen v Android a WP verzi, ale také v AFSwinx, jež interpretuje UI pro Swing aplikace [34]. Nakonec byla anotace umístěna k atributům v serverových entitách, kterých se to týkalo, příklad použití lze vidět v ukázce kódu 4.4.

Listing 4.4: Ukázka použití anotace @UILessThan

```
@Entity
public class AbsenceInstance {
    ...
    @Temporal( value = TemporalType.DATE)
    private Date startDate;
    @Temporal( value = TemporalType.DATE)
    private Date endDate;
    ...
    @UILessThan( value="endDate")
    public Date getStartDate() {
        return startDate;
    }
    ...
}
```

## 4.4 Porovnání přístupů

V Android a WindowsPhone aplikacích lze GUI vytvářet buď za pomoci kódu v příslušném programovacím jazyce nebo staticky za pomoci XML šablon. V obou případech platí, že musí vývojář každý prvek GUI nadefinovat, naplnit daty a pokud má být na prvek navázána nějaká akce, musí naimplementovat i ji. V případě formuláře tedy musí obvykle vytvořit dané aktivní prvky s popisky, element pro zobrazení validačních chyb a tlačítko pro odeslání formuláře. K tomu musí naprogramovat logiku validací, získání dat z formuláře a pokud má být formulář předvyplněn daty, tak i způsob jeho naplnění. Jestliže jsou navíc data získávána ze serveru, je nutné naimplementovat připojení na server, jak se data získají a reprezentují a způsob jejich odeslání na server. To vše znamená pro vývojáře spoustu kódu, který musí napsat, zvlášť když se komponenta objevuje na více místech v aplikaci.

Pokud jsou komponenty vytvářeny frameworkem, většina částí, které musí klasicky programátor řešit, za něj provádí framework. Při tvorbě komponent stačí vývojáři nadefinovat zdroje, ze kterých se má formulář vytvořit, naplnit daty nebo kam se má odeslat a pomoci

frameworku komponentu vytvořit a vložit do GUI. Za pomoci frameworku se vygenerují aktivní prvky i s jejich popisky podle toho, co udává server. Server také určí validace k nimž framework vytvoří validátory a řeší i zobrazení případných validačních chyb. V případě formuláře definuje i jak se z něj získají data a jak se mají odeslat na server. Připojení na server také není potřeba implementovat.

To vše vede k výrazné redukci kódu, což je nepochybně výhodou. Další výhodou je, že je GUI definováno serverem a reaguje tedy automaticky na změny například na serveru. Nevýhodou však je, že je vývojář omezen na funkce, které framework nabízí a využití frameworku požaduje mít server, který poskytuje definice komponent. Tedy při nasazení na existující aplikaci nestačí jen použít framework na straně klienta, ale je nutné výrazně upravit serverovou část. V případě klasického přístupu není nutné při tvorbě klienta server měnit a vývojář klienta pouze serveru přizpůsobuje. Také je nespornou výhodou, že má vývojář na komponenty plnou kontrolu a daleko větší možnosti jak s ní naložit, například co se úpravy vzhledu týče.



# Literatura

- [1] *Introduction to Android* [online]. [cit. 09.04.2016]. Dostupné z: <<http://developer.android.com/guide/index.html>>.
- [2] *Linear Layout* [online]. [cit. 13.04.2016]. Dostupné z: <<http://developer.android.com/guide/topics/ui/layout/linear.html>>.
- [3] *Platform Versions* [online]. [cit. 09.04.2016]. Dostupné z: <<http://developer.android.com/about/dashboards/index.html>>.
- [4] *Styles and Themes* [online]. [cit. 26.03.2016]. Dostupné z: <<http://developer.android.com/guide/topics/ui/themes.html>>.
- [5] *Apache Cordova* [online]. [cit. 23.03.2016]. Dostupné z: <<https://cordova.apache.org/>>.
- [6] *AspectFaces framework* [online]. [cit. 23.03.2016]. Dostupné z: <<http://www.aspectfaces.com/overview>>.
- [7] *AsyncTask* [online]. [cit. 12.04.2016]. Dostupné z: <<http://developer.android.com/reference/android/os/AsyncTask.html>>.
- [8] *Factory pattern* [online]. [cit. 13.04.2016]. Dostupné z: <<http://www.oodesign.com/factory-pattern.html>>.
- [9] *GlassFish Server Open Source Edition Quick Start Guide, Release 4.0* [online]. Dostupné z: <<https://glassfish.java.net/docs/4.0/quick-start-guide.pdf>>.
- [10] *Gson User Guide* [online]. [cit. 01.04.2016]. Dostupné z: <<https://github.com/google/gson/blob/master/UserGuide.md>>.
- [11] *Android 6.0 Changes* [online]. [cit. 12.04.2016]. Dostupné z: <<http://developer.android.com/about/versions/marshmallow/android-6.0-changes.html>>.
- [12] *Introducing JSON* [online]. [cit. 01.04.2016]. Dostupné z: <<http://www.json.org/>>.
- [13] *Metadata* [online]. [cit. 09.04.2016]. Dostupné z: <<https://en.wikipedia.org/wiki/Metadata>>.
- [14] *What Are RESTful Web Services?* [online]. [cit. 22.03.2016]. Dostupné z: <<https://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html>>.

- [15] *PHP Database Form* [online]. [cit. 26.03.2016]. Dostupné z: <<http://phpdatabaseform.com/>>.
- [16] *Serialization* [online]. [cit. 01.04.2016]. Dostupné z: <<https://en.wikipedia.org/wiki/Serialization>>.
- [17] *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](https://wikisofia.cz/index.php/Typy_uživatelských_rozhraní_a_jejich_specifika/old)>.
- [18] *Web service* [online]. [cit. 22.03.2016]. Dostupné z: <[https://en.wikipedia.org/wiki/Web\\_service](https://en.wikipedia.org/wiki/Web_service)>.
- [19] [online]. [cit. 13.04.2016]. Dostupné z: <<https://msdn.microsoft.com/library/windows/apps/windows.ui.xaml.controls.grid.aspx>>.
- [20] *HttpClient class* [online]. [cit. 12.04.2016]. Dostupné z: <<https://msdn.microsoft.com/library/windows/apps/windows.web.http.httpclient.aspx>>.
- [21] *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](https://msdn.microsoft.com/en-us/library/windows/apps/ff402557(v=vs.105).aspx)>.
- [22] 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>>.
- [23] ADDUPLEX. *AddDuplex Windows Phone Statistics Report - January, 2016* [online]. Dostupné z: <<http://www.slideshare.net/adduplex/adduplex-windows-phone-statistics-report-march-2016>>.
- [24] 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.
- [25] 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>>.
- [26] 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.
- [27] 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>>.
- [28] 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.
- [29] GAMMA, E. et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA : Addison-Wesley, 1995. ISBN 978-0-201-63361-0.
- [30] HANS DOCKTER, A. M. *Gradle User Guide* [online]. [cit. 11.04.2016]. Dostupné z: <<https://docs.gradle.org/current/userguide/userguide.html>>.
- [31] JENKOV, J. *Web Service Message Formats* [online]. [cit. 23.03.2016]. Dostupné z: <<http://tutorials.jenkov.com/web-services/message-formats.html>>.
- [32] SOMMERVILLE, I. *Software engineering*. : Addison-Wesley, 9 edition, 2011. ISBN 978-0-13-703515-1.
- [33] TAKA, D. *An Introduction to Windows Phone 8.1 Development* [online]. [cit. 26.03.2016]. Dostupné z: <<http://www.sitepoint.com/introduction-windows-phone-8-1-development>>.
- [34] 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

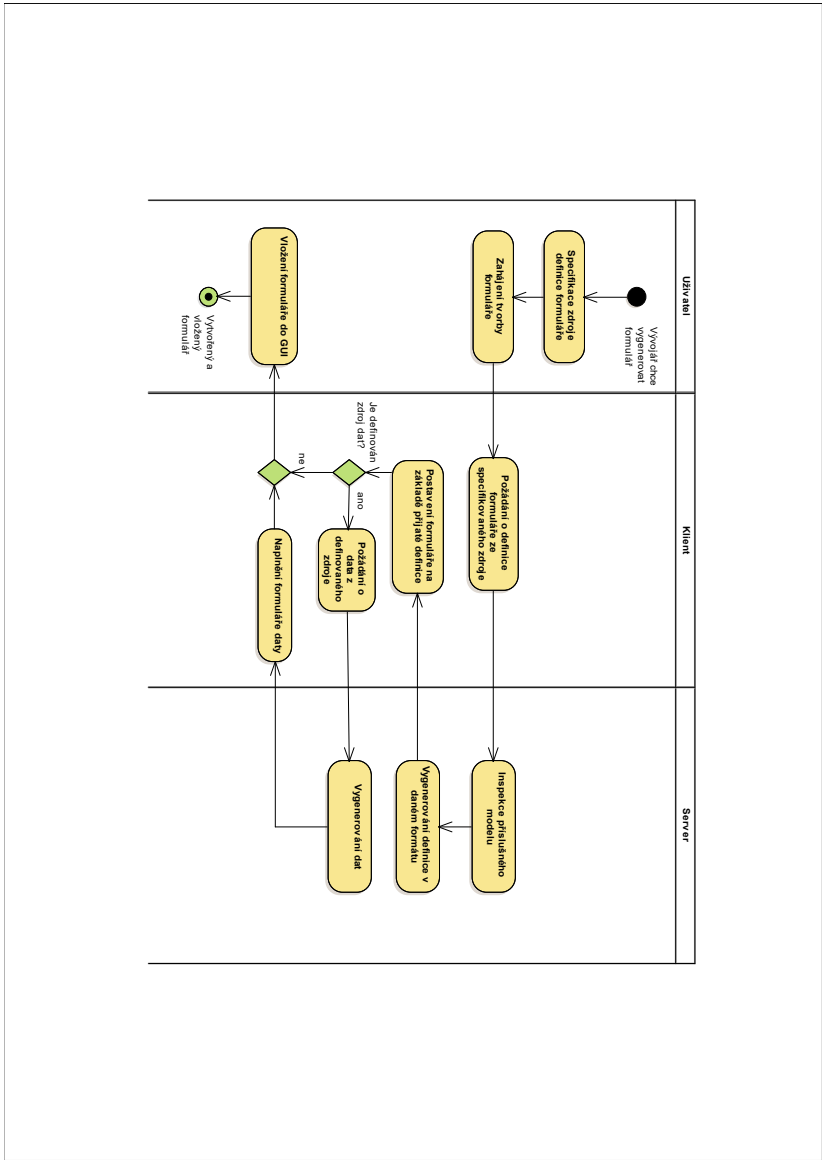
<b>CUI</b>	Character User Interface
<b>GUI</b>	Graphical User Interface
<b>UI</b>	User Interface
<b>REST</b>	Representational State Transfer
<b>URI</b>	Uniform Resource Identifier
<b>HTTP</b>	Hypertext Transfer Protocol
<b>JSON</b>	JavaScript Object Notation
<b>XML</b>	Extensible Markup Language
<b>XAML</b>	Extensible Application Markup Language
<b>Java SE</b>	Java Platform Standart Edition
<b>Java EE</b>	Java Platform Enterprise Edition
<b>URL</b>	Uniform Resource Locator
<b>PHP</b>	Personal Home Page
<b>HTML</b>	HyperText Markup Language
<b>SQL</b>	Structured Query Language
<b>CSS</b>	Cascading Style Sheets
<b>JPA</b>	Java Persistence API
<b>LGPL</b>	GNU Lesser General Public License
<b>UML</b>	Unified Modeling Language
<b>C#</b>	C Sharp
<b>WP</b>	Windows Phone



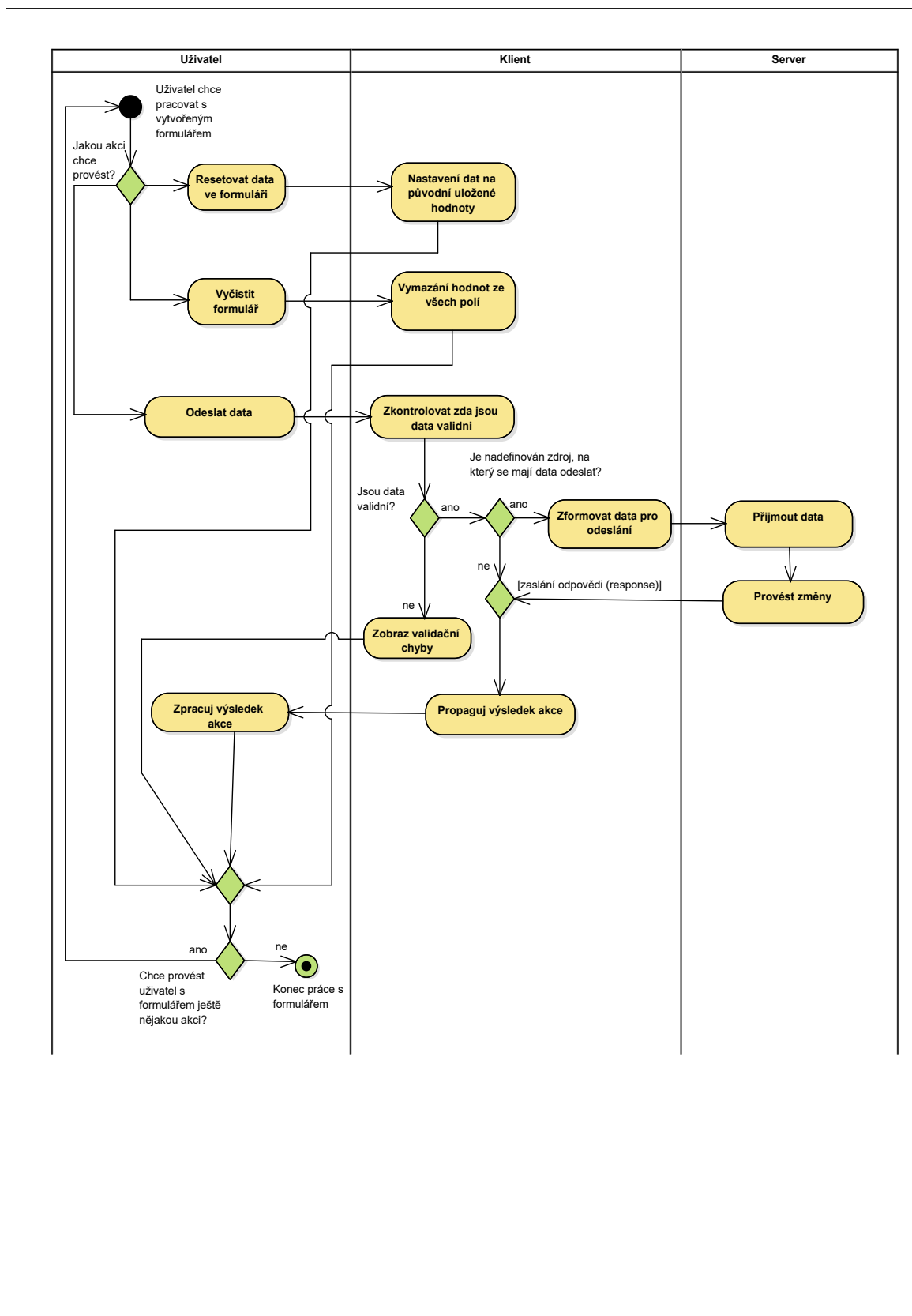
## Příloha B

# UML diagramy a obrázky

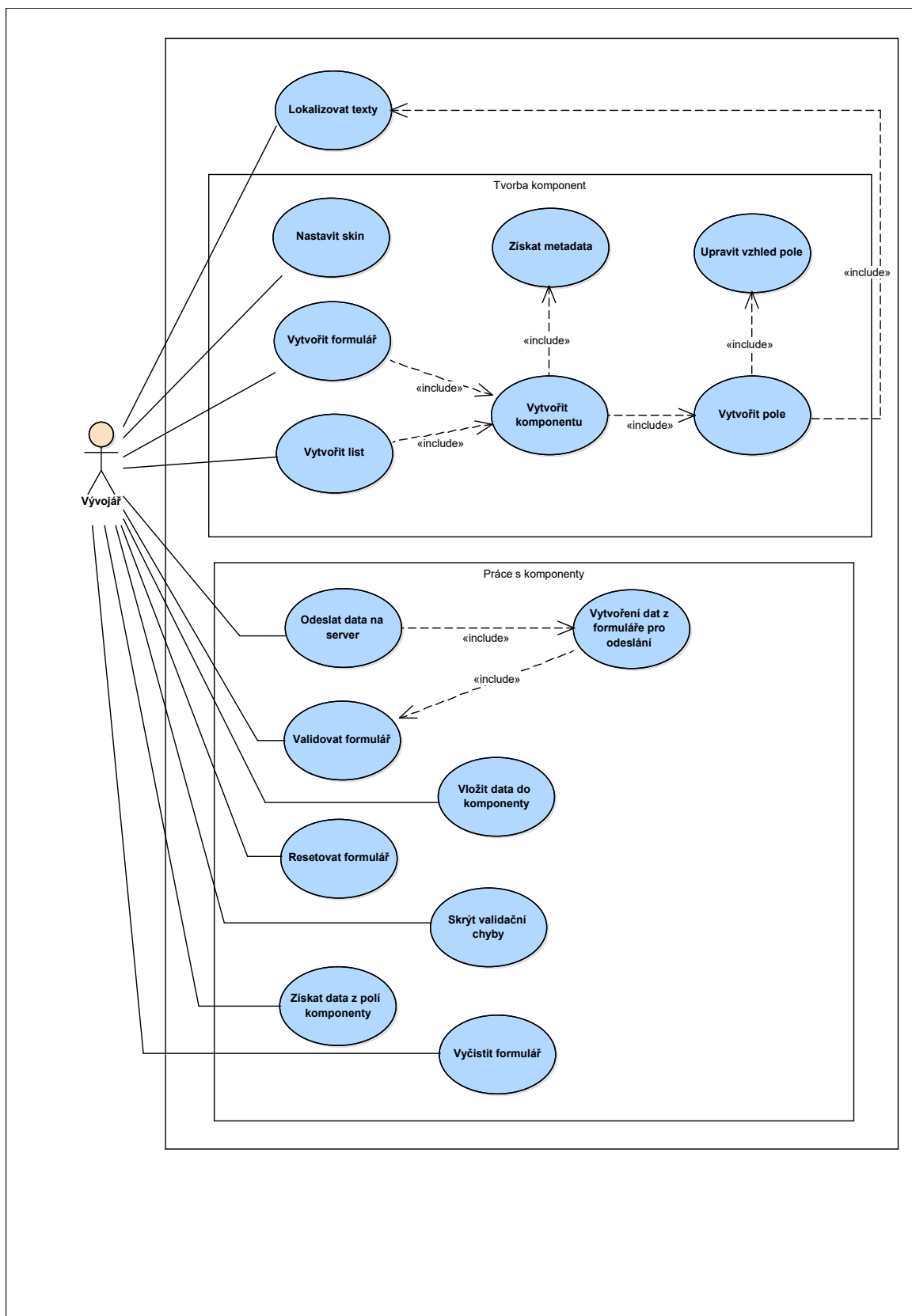
V této sekci naleznete použité UML diagramy a velké obrázky, na které bylo v textu odkazováno.



Obrázek B.1: Diagram aktivit popisující proces tvorby formuláře

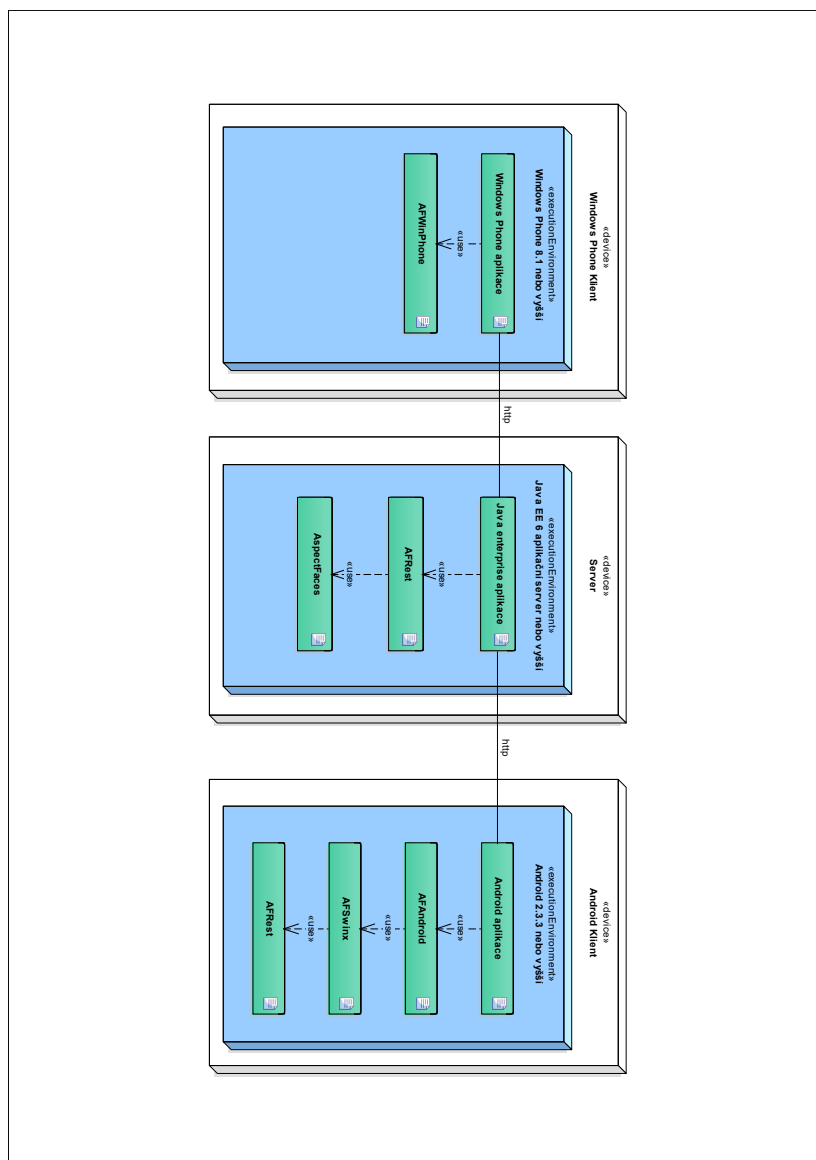


Obrázek B.2: Diagram aktivit popisující práci s formulářem

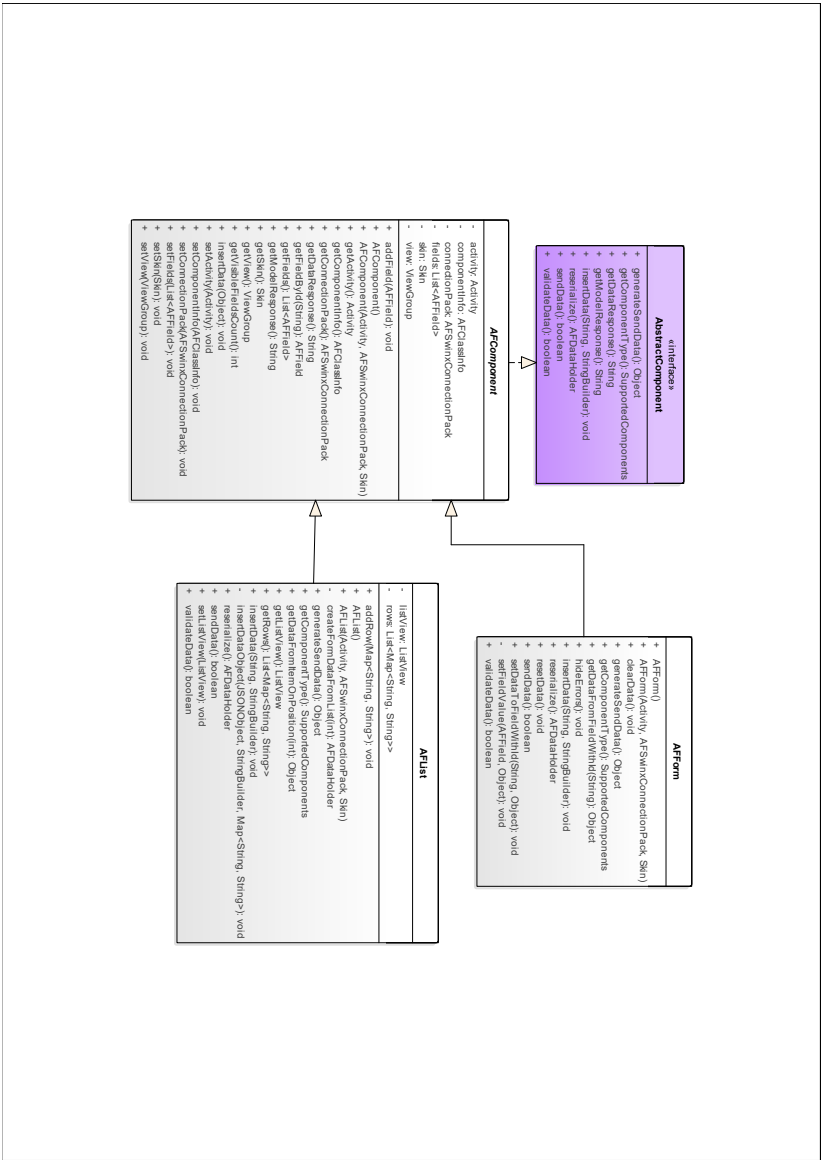


Obrázek B.3: Model případů užití frameworku

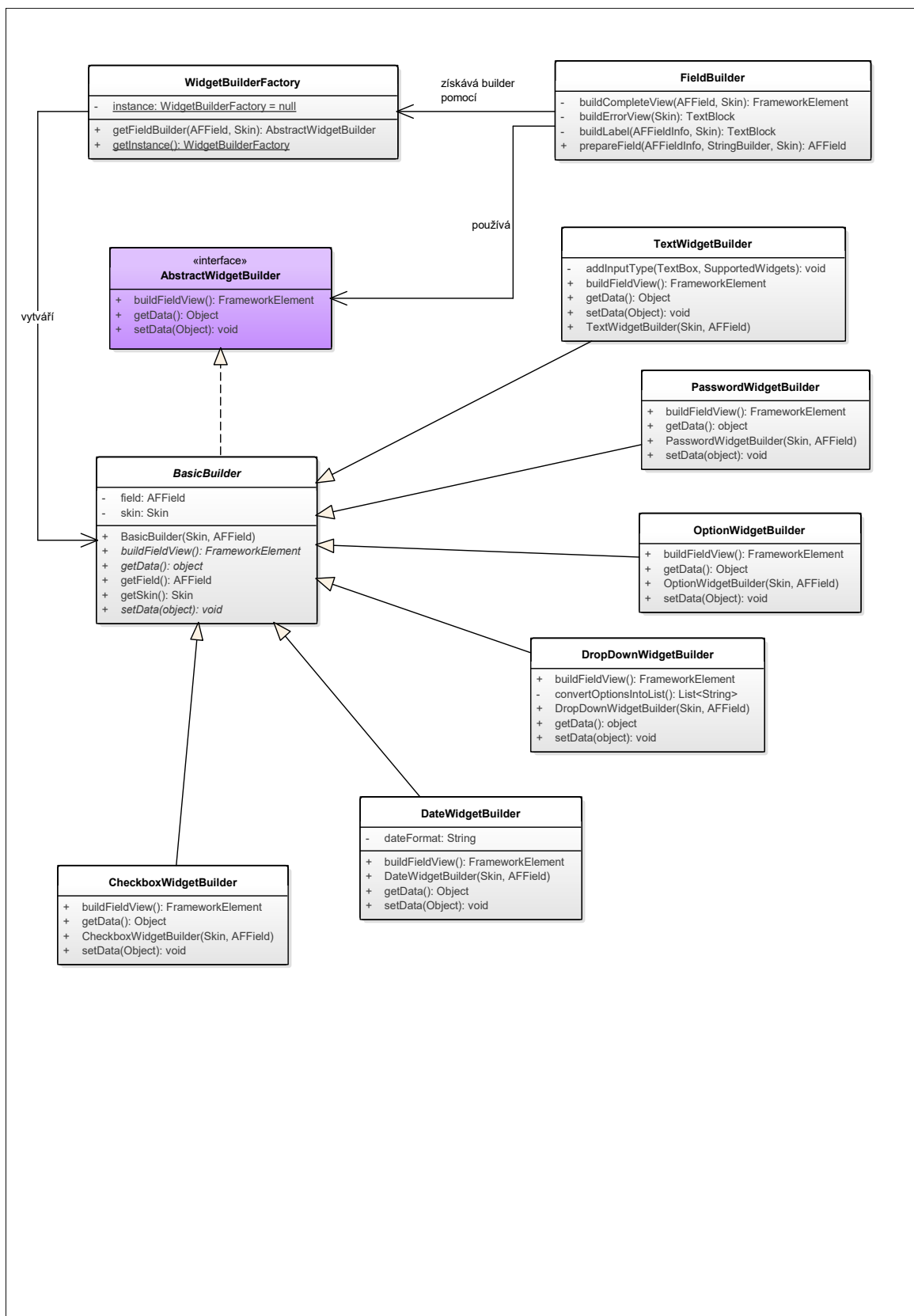




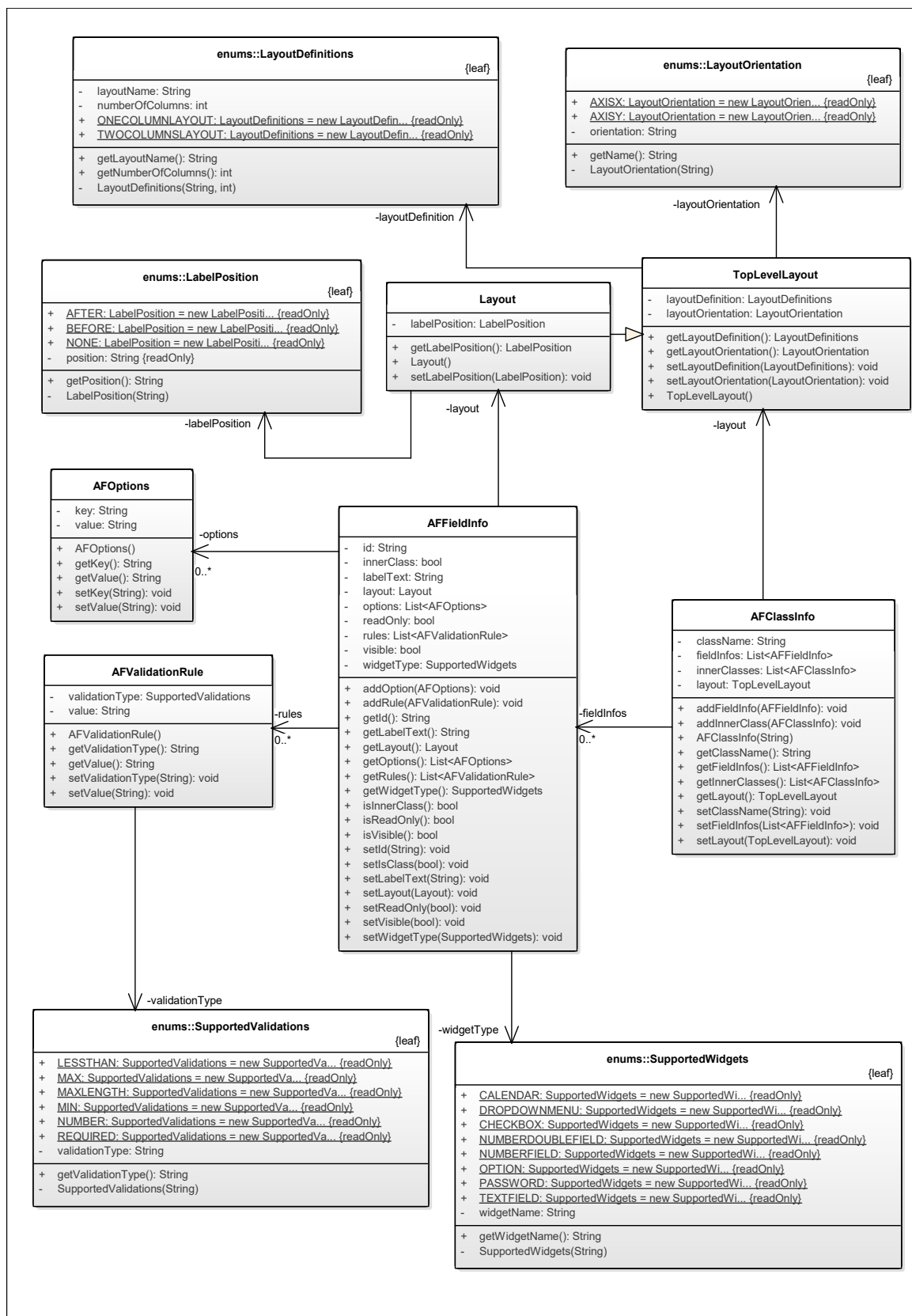
Obrázek B.4: Diagram nasazení systému



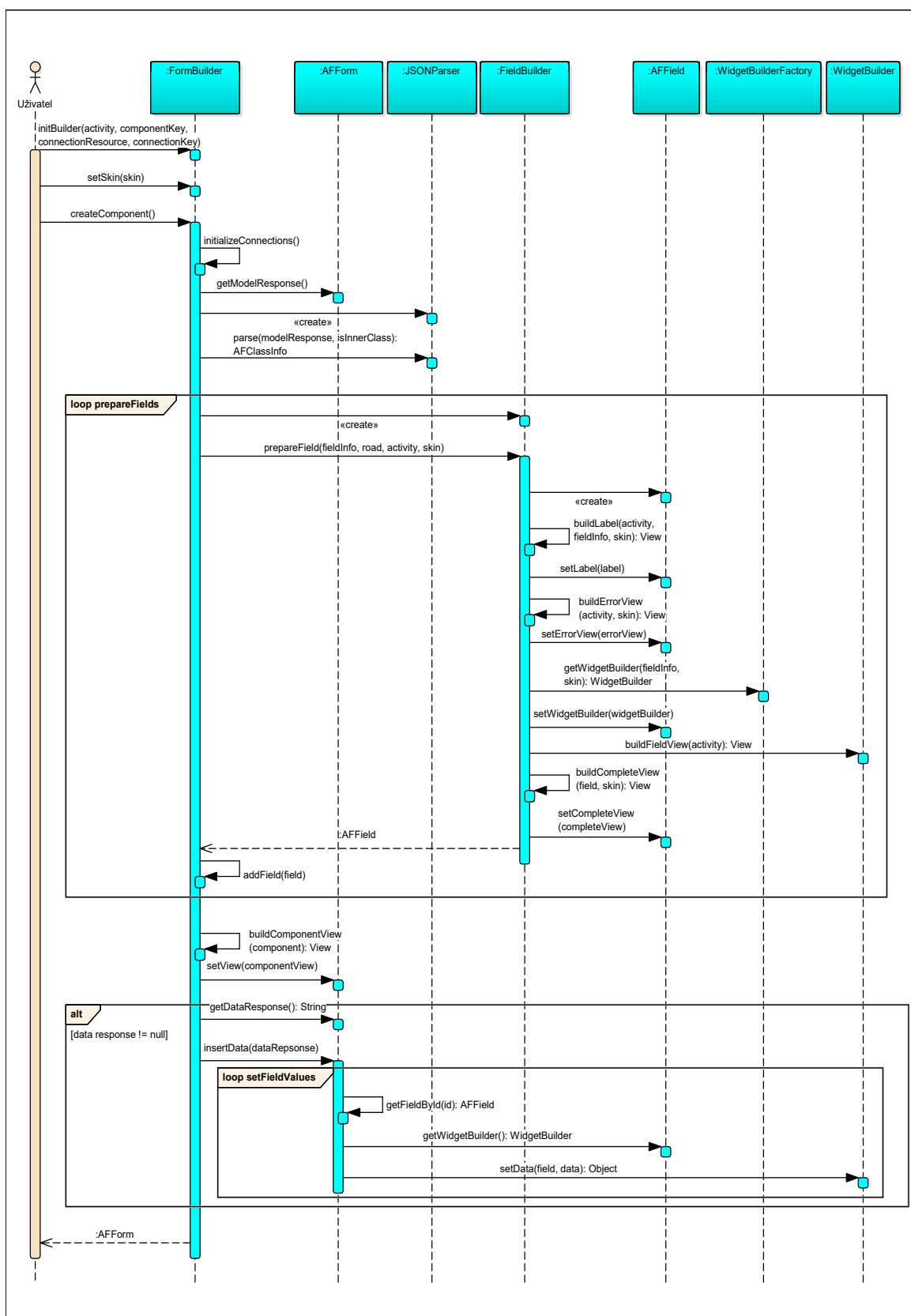
Obrázek B.5: Diagram tříd zachycující komponenty



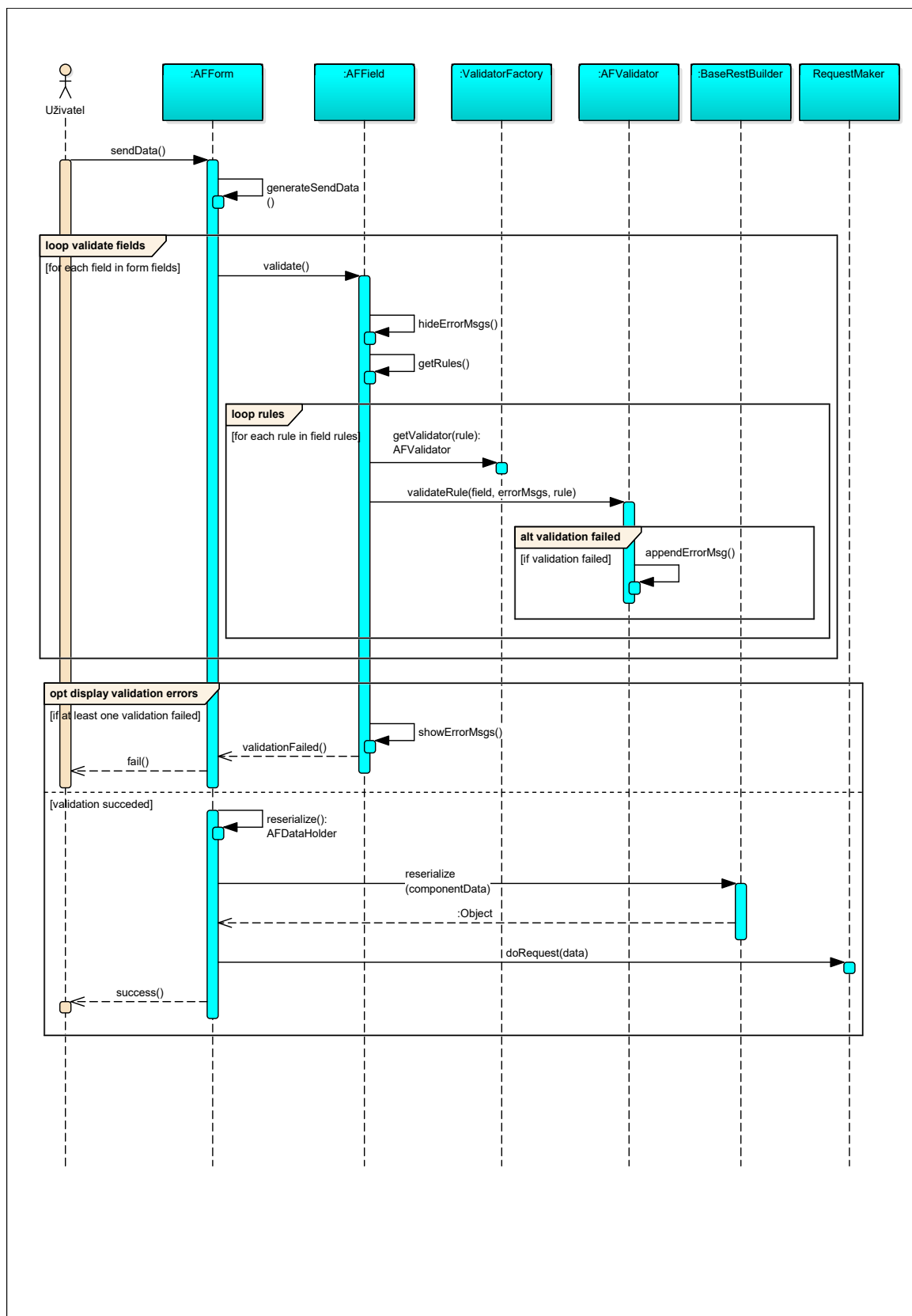
Obrázek B.6: Diagram tříd zachycující widget buildery



Obrázek B.7: Diagram tříd popisující strukturu uložení metadat



Obrázek B.8: Sekvenční diagram zobrazující proces tvorby formuláře



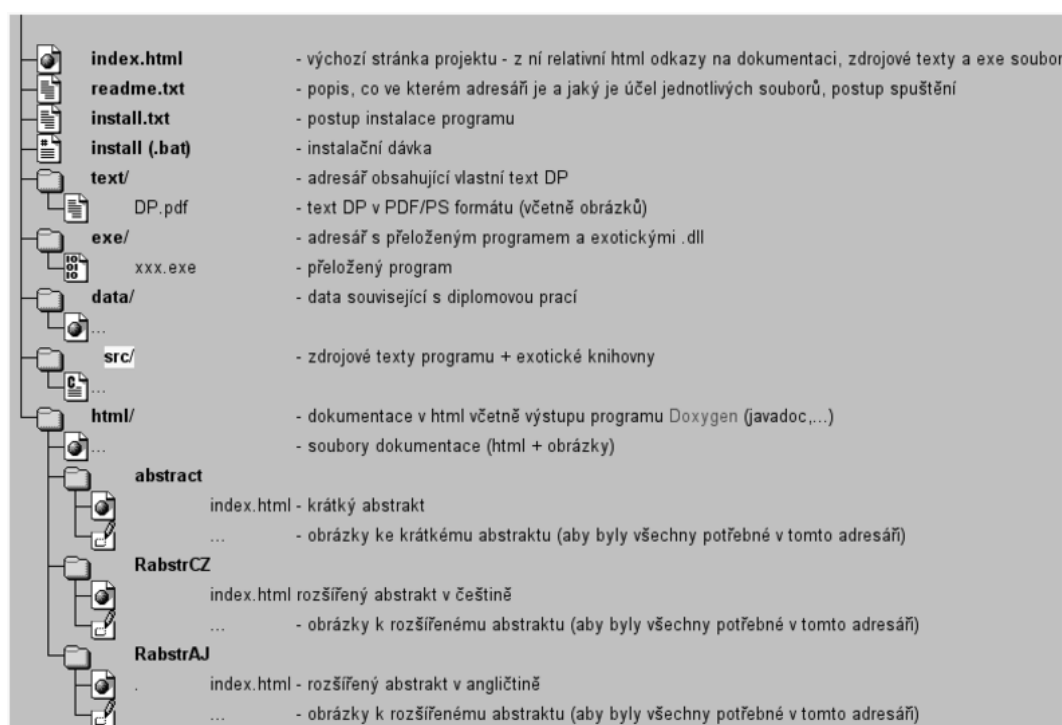
Obrázek B.9: Sekvenční diagram zobrazující proces odeslání formuláře

## 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** (případně 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.