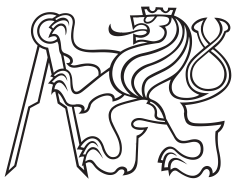
Diplomová práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačové grafiky a interakce

Implementace mobilního klienta pro ERP systém Helios na platformě Android

Bc. Pavel Nedoma
Studijní program: Otevřená informatika
Obor: Softwarové inženýrství

Květen 2014
Vedoucí práce: Ing. Pavel Náplava

Poděkování / Prohlášení

Tímto bych chtěl poděkovat rodičům za podporu při studiu. Dále bych chtěl poděkovat vedoucímu práce Ing. Pavlu Náplavovi za užitečné rady a připomínky při tvorbě diplomové práce.

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 Praze dne 9. 5. 2014

.....

Abstrakt / Abstract

Práce popisuje vývoj mobilního klienta ERP systému HELIOS Green na platformě Android. Hlavním cílem je vytvoření dostatečně obecného klienta, který je snadno konfigurovatelný pro použití v rámci jakékoliv agendy. Toho je dosaženo na základě analýzy systému HELIOS Green a jeho dekompozice na základní stavební prvky. Tyto prvky jsou následně implementovány s ohledem na minimalizaci omezení daných mobilní platformou.

Klíčová slova: HELIOS Green; ERP; mobilní klient; Android;

This thesis aims to describe development of the Android mobile client for the HELIOS Green ERP system. The main goal is to create a general mobile client that can be easily used to support various organization processes. This goal is reached by analysis HELIOS Green system and finding its base building blocks. These blocks are then implemented with a focus on minimizing the impact of mobile device limitations.

Keywords: HELIOS Green; ERP; mobile access; Android;

Title translation: Android Mobile Client for the HELIOS ERP System

Obsah /

Tabulky / Obrázky

Kapitola 1
Úvod

Moderní ERP systémy se staly nedílnou součástí správy a řízení organizací bez ohledu na jejich velikost či zaměření stejně tak, jako se mobilní zařízení stala nedílnou součástí našich životů. Tato práce se snaží o přiblížení těchto dvou světů a přináší výhody mobilního přístupu do ERP systému HELIOS Green.

Zkratka ERP (Enterprise Resource Planning) označuje komplexní informační systémy, jejichž úkolem je podpořit veškeré činnosti dané společnosti, ať již jde o výrobu, finance, CRM, řízení lidských zdrojů, marketing či účetnictví bez ohledu na pole působnosti. Mezi hlavní výhody ERP patří centralizace dat a sjednocení procesů společnosti, umožňující jejich přehledné monitorování a zefektivnění. Toto je předpokladem zvýšení potenciálu společnosti pro poskytování kvalitnější péče a služeb zákazníkovi a v konečném důsledku zlepšení konkurenceschopnosti na trhu. Klíčová je pro úspěšné provozování ERP systému počáteční analýza a implementace řešení na míru potřebám společnosti. [1]

Mezi hlavní výrobce ERP systémů patří společnosti: SAP, Oracle a Microsoft. Existuje ale mnoho dalších komerčních i open source řešení. Tato práce se zaměřuje na ERP systém HELIOS Green české společnosti Asseco Solutions, a.s. HELIOS Green je podobně jako ostatní zmíněná řešení modulární platformou, kde každá instalace u zákazníka může podporovat odlišné spektrum činností. Záleží tedy plně na zákazníkovi, které moduly se rozhodne zakoupit. Základní nabídka obsahuje například moduly pro:

- ekonomiku (účetnictví, fakturace),
- správu lidských zdrojů,
- logistiku,
- skladové hospodářství,
- elektronickou správu dokumentů,
- CRM,
- řízení podniku,
- výrobu (kalkulace, řízení),
- řízení servisní činnosti (reklamace, helpdesk),
- dopravu (kniha jízd),
- a mnoho dalších modulů pro specifické odvětví průmyslu (automobilová výroba, stavebnictví atd.).

Samozřejmá je možnost zakázkového dovývoje modulů zákazníkovi na míru jeho procesům a přesně podle jeho představ.

Kapitola 2
Popis problému, specifikace cíle

Cílem této práce je navrhnout a implementovat mobilního klienta pro ERP systém HELIOS Green na platformě Android. Zadání této práce vychází z požadavků společnosti ALDOR s.r.o., která je obchodním, vývojovým a implementačním partnerem společnosti Asseco Solutions, a.s. Součástí zadání jsou i tři agendy, které má výsledný produkt podporovat. Zároveň by měl být mobilní klient navržený s ohledem na snadné rozšíření o další agendy. Rozmanitost a rozsah v úvodu vyjmenovaných agend systému HELIOS Green naznačuje, že statická implementace všech podporovaných agend by vyžadovala nesmírné množství času a pravděpodobně by byla nad rámec možností dnešních mobilních zařízení. Agendy ze zadání budou tedy sloužit nejen jako testovací scénáře výsledné implementace, ale také jako podklady v průběhu analýzy systému.

Důraz je také kladen na uživatelské rozhraní mobilního klienta. Je důležité, aby aplikace respektovala uživatelské zvyklosti dané platformou a byla ovladatelná bez ohledu na velikost zařízení.

2.1 Existující řešení

Trh s ERP systémy je trhem velmi specifickým. Většina nabídek řešení je společnostem nabízena neveřejnými kanály prostřednictvím obchodních zástupců a to včetně možnosti vyzkoušet si mobilní klienty. Z veřejně dostupných systémů jsem měl možnost vyzkoušet mobilního klienta Microsoft Dynamics CRM pro platformu Android. V případě systému HELIOS Green řešení vyhovující zadání neexistuje. V současné době je jedinou mobilní aplikací pro tento systém produkt HELIOS Green Mini.

2.1.1 Microsoft Dynamics CRM

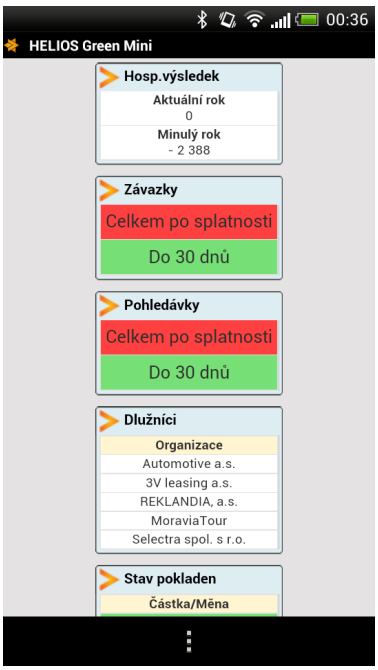
Microsoft Dynamics CRM je systém pro správu a podporu komunikace se zákazníky. Tento systém se primárně ovládá prostřednictvím webového rozhraní nebo z rozhraní aplikace Microsoft Outlook. Webové rozhraní je možné používat také na mobilním zařízení, nicméně k dispozici je i nativní klient pro Android. Klient je při používání plynulejší než webové rozhraní a umožňuje lépe využít možnosti zařízení. Příkladem je možnost přímo vytočit telefonní kontakt, nebo na emailovou adresu odeslat email. Ukázka formuláře otevřeného kontaktu je na obrázku 2.1. Ačkoliv je CRM pouze jednou z mnoha agend systému HELIOS Green, je aplikace Microsoft Dynamics CRM vhodným příkladem, jak může mobilní klient informačního systému vypadat.

2.1.2 HELIOS Green Mini

Tato aplikace společnosti Asseco Solutions existuje nejen pro platformu Android, ale i pro konkurenční iOS a pro stolní verzi Microsoft Windows (7 a novější). Úkolem aplikace je zobrazovat klíčové informace ERP systému prostřednictvím takzvaných „gadgetů“. Jedná se o malé komponenty, které je možné konfigurovat a vytvářet v prostředí klasického klienta systému HELIOS Green. Aplikace pak v pravidelných intervalech ze

serveru aktualizuje zobrazované hodnoty. Výhodou je jednoduchost a přehledná signalizace významných skutečností. Data lze ale pouze zobrazovat, nikoliv měnit. Ukázka rozhraní aplikace je na obrázku 2.2.

Obrázek 2.1. Aplikace Microsoft Dynamics CRM pro platformu Android.



Obrázek 2.2. Aplikace HELIOS Green Mini pro platformu Android.

2.1.3 HELIOS Mobile

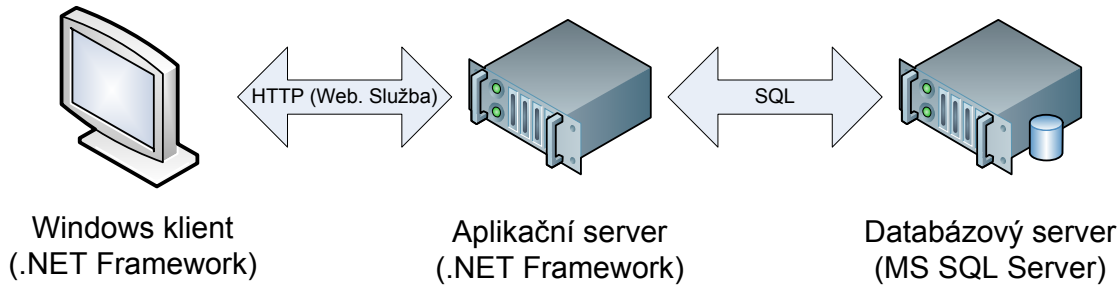
V tomto případě se nejedná o existující řešení, nýbrž o výsledek této diplomové práce. Během vytváření diplomové práce došlo k odkladu odevzdání jejího textu. Mobilní aplikace tak byla společností ALDOR uveřejněna již v prvním čtvrtletí roku 2014 pod názvem HELIOS Mobile.

Kapitola 3
ERP systém HELIOS Green

Cílem této kapitoly je shrnout základní strukturu a principy fungování ERP systému HELIOS Green (dále jen HeG). Po úvodním nastínění celkové architektury systému se zaměřím na logický model systému, tak jak je vnímaný uživatelem nebo správcem. Toto je důležité z hlediska zavedení terminologie používané dále v analýze, ale také pro sběr klíčových vlastností, které musí výsledný klient podporovat a musí být tedy zohledněny již v samotném návrhu. Dále se zaměřím na vnitřní reprezentaci a způsob uložení dat. Na základě získaných poznatků bude tvořen datový model klienta. Závěrem kapitoly prozkoumám prostředky a rozhraní poskytované systémem HeG pro komunikaci s aplikacemi třetích stran, které mohou být využity vyvíjeným klientem. Popis v této kapitole vychází z mých osobních zkušeností se systémem HeG a z informací dostupných na dokumentačním portálu systému HeG [2], kde je také možné nalézt detailnější informace a specifikace.

3.1 Architektura systému

Architektura systému HeG odpovídá klasické třívrstvé architektuře a je znázorněna na obrázku 3.1. Celý produkt je postaven na technologiích společnosti Microsoft. Nejnižší datová vrstva je databázový server na platformě Microsoft SQL Server a obsahuje veškerá uživatelská data a metadata aplikace. Prostřední vrstva aplikační logiky je tvořena aplikačním serverem, který může existovat ve více instancích za účelem zvýšení výkonu a obsloužení většího množství uživatelů. Prezentační vrstvu tvoří klient pro operační systém Microsoft Windows. Jedná se o tenkého klienta a většinová část uživatelského rozhraní se chová na základě logiky aplikačního serveru, se kterým klient komunikuje prostřednictvím webové služby. Bez aktivního spojení na server se stává klient nepoužitelným. Implementace aplikačního serveru i klienta je pomocí technologie .NET Framework.

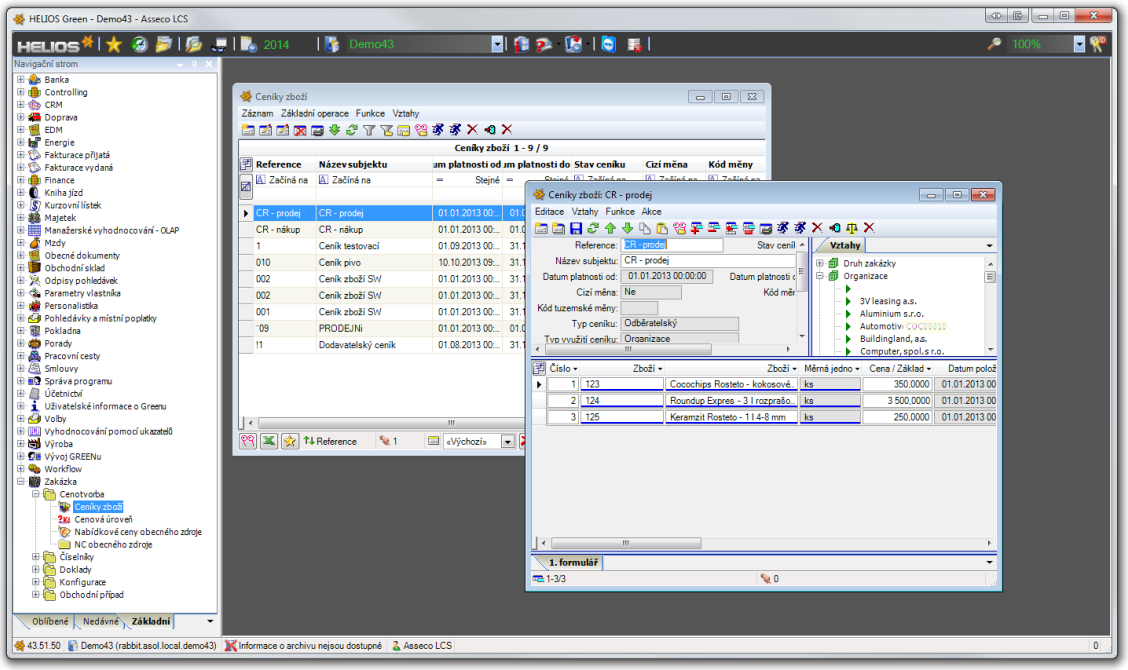


Obrázek 3.1. Architektura systému HELIOS Green.

3.2 Uživatelské rozhraní

Pro lepší představu o práci se systémem HeG a pro lepší začlenění následujícího oddílu do kontextu, provedu krátké shrnutí uživatelského rozhraní klienta. Aplikace klienta

má klasické MDI[1]) rozhraní, hlavní okno aplikace tedy slouží pro zobrazování dalších oken potomků s konkrétními daty. Pro přístup k datům slouží strom pořadačů v levé části hlavního okna. Po výběru pořadače (3.3.3) ze stromové struktury dojde k otevření přehledu (3.3.9) a následně může uživatel zvolit záznam, se kterým potřebuje pracovat. Ukázka popsané navigace je na obrázku 3.2.



Obrázek 3.2. Uživatelské rozhraní klienta systému HELIOS Green.

3.3 Součásti systému

V úvodu práce (1) jsem vyjmenoval některé z mnoha agend, které systém HeG nabízí. Navzdory dojmu komplexnosti, který tento seznam agend může navozovat, je možné v jádru systému identifikovat malou množinu stavebních prvků, ze kterých se jednotlivé agendy skládají. Nyní se na tyto stavební prvky jednotlivě zaměřím.

3.3.1 Třídy

Podobně jako u objektově orientovaných programovacích jazyků slouží v HeG třída jako předpis datové struktury záznamu. Třída definuje, jaké atributy a vztahy záznam obsahuje, jaká knihovna má na starosti vykonávání aplikační logiky při práci se záznamem, ale i místo (tabulku), na které jsou záznamy ukládány. Dále je možné na základě tříd řídit oprávnění uživatelů.

Je důležité zmínit, že třída deklaruje, zda záznam obsahuje položky. Vhodným příkladem záznamu s položkami z běžného života je záznam reprezentující fakturu. Fakturu lze rozdělit na dvě části. Na část hlavičkovou, kde jsou uvedeny základní informace jako je datum, číslo a údaje o subjektu, který fakturu vydal a také samozřejmě údaje subjektu, kterému je faktura určena. Druhou částí jsou pak samotné fakturované položky. Položky mají společný formát, kde každá obsahuje například: číslo řádky, název, počet, jednotkovou cenu, jednotku a celkovou cenu. Oproti tomu u záznamu typu zaměstnanec nám stačí uchovávat pouze „hlavičkové“ údaje a pro položky nemáme využití.

[1]) Multiple document interface – rozhraní pro práci s více dokumenty najednou.

3.3.2 Záznamy

Veškerá data, se kterými HeG pracuje, jsou uložena v záznamech. Záznam je tedy takovým základním stavebním prvkem celého systému. Pokud bych měl opět použít analogii objektově orientovaného programování, je záznam instancí nějaké třídy, obsahující konkrétní data – ať již hodnoty atributů nebo vztahy na jiné záznamy. Takový záznam může například reprezentovat zaměstnance, fakturu, kontaktní osobu, zkrátka jakoukoliv entitu, která v dané agendě figuruje.

3.3.3 Pořadače

Pořadače slouží pro vymezení množiny záznamů jedné třídy, které patří z určitého pohledu k sobě. V kontextu příkladu s fakturou lze pořadač přirovnat k šanonu, kdy společnost sdružuje faktury do šanonů například podle roku či měsíce, ve kterém byla faktura vydána. Každý záznam musí být umístěn v právě jednom pořadači. Podobně jako v případě třídy je možné využít pořadač k řízení oprávnění.

3.3.4 Atributy

Pro definici jednotlivých datových polí třídy slouží atribut. Základním parametrem atributu je datový typ. Podporovány jsou následující datové typy:

- datum,
- datum a čas,
- číslo,
- řetězec,
- logický typ,
- pořadač,
- poznámka,
- binární data.

Jedná se o datový typ v rámci aplikační logiky HeG a nejedná se přímo o databázový datový typ (viz 3.4.4). Kromě datového typu umožňuje atribut nastavit editační styl.

Editační styl má dvě základní funkce. Pro časové a číselné datové typy může definovat formátovací řetězce. Druhou funkcí je možnost pro řetězce a čísla nastavit výčet povolených hodnot.[1]) Výčet je definován množinou párů *datová hodnota – vizuální hodnota*. Klasickým příkladem je editační styl pro logické hodnoty [*A – Ano*; *N – Ne*]. V případě, že uživatel zvolí „Ano“, bude systém interně uchovávat písmeno „A“. Atribut také definuje svůj název (popisek), pro který systém HeG umožňuje zadefinovat překlady do jiných jazyků. Dále je možné nastavit, zda je atribut povinný, v takovém případě systém nedovolí záznam uložit, dokud nedojde k vyplnění hodnoty všech povinných atributů.

3.3.5 Vztahy

Pro reprezentaci vazeb mezi záznamy slouží vztahy. Umožňují nám například propojit již zmiňovanou fakturu s konkrétním záznamem odběratelské organizace a na položce faktury například odkazovat přímo na karty konkrétního zboží. V HeG rozlišujeme dva typy vztahů: statický vztah a dynamický vztah. Dříve, než se budu věnovat jednotlivým typům vztahů, se zaměřím na jejich společné vlastnosti.

[1]) Kromě editačního stylu, je možné výčet také nastavit pomocí uživatelsky definovaného záznamu Valuace atributů.

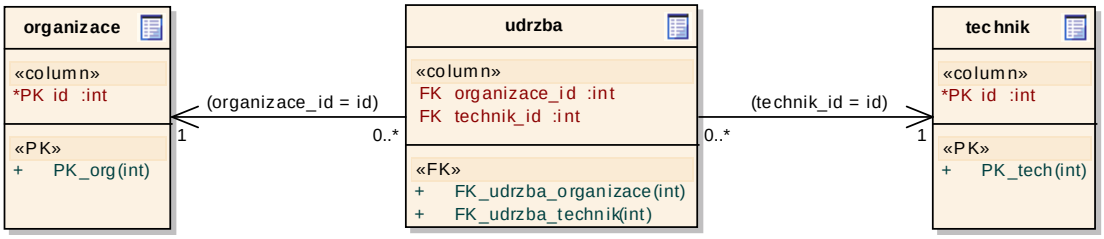
Podobně, jako tomu je v případě atributů, i u vztahů je možné vyžadovat povinné vyplnění (parcialitu) a nastavit popisek včetně lokalizace. Navíc je v některých případech vhodné vyplnit i popisek pro případ interpretace vztahu z druhé strany. Důležitou položkou definice vztahu je vymezení záznamů, které je možné do vztahu navazovat. Každý vztah tedy definuje množinu pořadačů, ze kterých lze záznamy vybírat. Kromě zmíněné parciality je možné omezit maximální kardinalitu vztahu.[1])

3.3.6 Statické vztahy

Tento typ vztahu má svou podstatou velmi blízko k atributu (3.3.4). Statický vztah je možné zobecnit na speciální případ atributu, který jako hodnotu uchovává identifikátor vztaženého záznamu – tedy cizí klíč. Jako příklad uvedu záznam typu kontaktní osoba, který se váže vztahem na záznam organizace. Z principu ukládání vyplývá i maximální kardinalita statických vztahů, která je *1:N*. Každý záznam na levé straně vztahu (kontaktní osoba) může odkazovat maximálně na jeden záznam napravo (organizaci), ale při pohledu na vztah zprava nemusí být maximální počet navázaných kontaktních osob pro jednu organizaci omezen.

3.3.7 Dynamické vztahy

Dynamické vztahy se používají především v případech, kdy nám nestačí omezení kardinality u statických vztahů. Zde máme k dispozici kardinalitu *M:N*, záznam na levé straně může být ve vztahu s libovolným počtem záznamů a to samé platí pro záznamy na pravé straně vztahu. Například můžeme jako společnost poskytující údržbu počítačových systémů chtít evidovat, který technik má na starosti jakou organizaci. Větší organizace mohou vyžadovat nasazení většího počtů techniků a zároveň nechceme mít techniky starající se pouze o jednu organizaci, proto v tomto případě potřebujeme dynamický vztah. Pro ukládání dynamických vztahů si již nevystačíme s hodnotami uloženými přímo na zúčastněných záznamech, ale je zde nutnost mít zvláštní úložiště (tabulku). Toto je pro modelování vztahů s kardinalitou *M:N* běžné. [3] Obrázek 3.3 ukazuje, jak by mohl být popisovaný vztah vymodelován v databázi. Způsob, jakým systém HeG řeší dynamické vztahy, konkrétně rozeberu v části 3.4.



Obrázek 3.3. Příklad vymodelovaného databázového vztahu kardinality M:N s pomocnou tabulkou.

3.3.8 Formulářové šablony

V předchozích odstavcích jsem představil základní prvky, které dohromady utvářejí třídu. Pomineme-li popisky, tak žádný prvek nedefinoval, jakým způsobem má systém záznamy prezentovat uživateli. Tento úkol mají na starosti formulářové šablony. Formulářové šablony jsou tvořeny až třemi XML soubory.[2])

[1]) Maximální počet záznamů na dané straně vztahu.

[2]) Nejedná se nutně o soubory na disku serveru, kromě výchozí šablony se nacházejí formuláře přímo v databázi aplikace.

První ze souborů obsahuje vizuální rozložení formuláře. Rozložení je tvořeno pomocí jedné či více tabulek, do kterých se postupně umisťují jednotlivé prvky. Umisťovanými prvky jsou atributy (3.3.4) a statické vztahy (3.3.6). Následuje malý příklad části XML vizuálního rozložení.

```
1  <tab TableColumns="3" TabPage>
2     <column Name="reference_subjektu" Width="120" Label="fm(MSG001)" />
3     <column Name="datum_prijeti" Label="Datum přijetí"></column>
4     <column Name="priorita" Width="59" Align="Right"></column>
5     <column Name="nazev_subjektu" Width="100%" ColSpan="3"></column>
6  </tab>
7  <tab TableColumns="2">
8     <column Name="smer_zakazky" Width="90" ReadOnly="True"></column>
9     <column Name="typ_dokladu" Width="90" ReadOnly="True"></column>
10    <column Name="typ_zakazky" Width="90" ReadOnly="True"></column>
11    <column Name="odezva" Width="50" Align="Right"></column>
12    <column Name="druh_zakazky_refer" Width="90"></column>
13    <column Name="druh_zakazky_nazev" Width="100%"
14     LabelPos="None" Align="Left"></column>
15 </tab>
```

Tato část obsahuje dvě tabulky (ř. 1 a 7) s rozdílným počtem sloupců. Do těchto tabulek se postupně umisťují prvky zleva doprava a shora dolů. Šablona identifikuje prvky pomocí vlastnosti *Name*, která obsahuje název sloupce z definice prvku nebo z datového XML formulářové šablony. Některé prvky mají nastavenou vlastnost *ColsSpan*, která říká kolik sloupců má prvek vyplnit (ř. 5). Dále je možné zamknout prvky pouze pro čtení, změnit jejich editační styl, změnit zarovnání hodnoty, nastavit editor[1]) nebo například skrýt popisek. Popisek je definován pomocí vlastnosti *Label*, pokud není vlastnost uvedena, použije se popisek z definice prvku. V ukázce jsou dvě použití této vlastnosti. Na třetím řádku je do vlastnosti nastaven přímo text, který formulář zobrazí. Druhý řádek ale obsahuje odkaz do překladového XML souboru.

Překladové XML pro zprávy je druhým ze tří souborů a obsahuje řetězce popisků. Každý popisek může definovat překlady popisků. Následující ukázka obsahuje jeden popisek přeložený pro české, slovenské a anglické prostředí aplikace s identifikátorem „MSG002“. Vizuální části šablony by pak na tento popisek odkázala takto: *Label*=`fm(MSG002)`.

```
1  <mess Code="MSG002">
2     <text LangID="CZ" Value="Základní údaje"></text>
3     <text LangID="SK" Value="Základné údaje"></text>
4     <text LangID="EN" Value="Basic data"></text>
5  </mess>
```

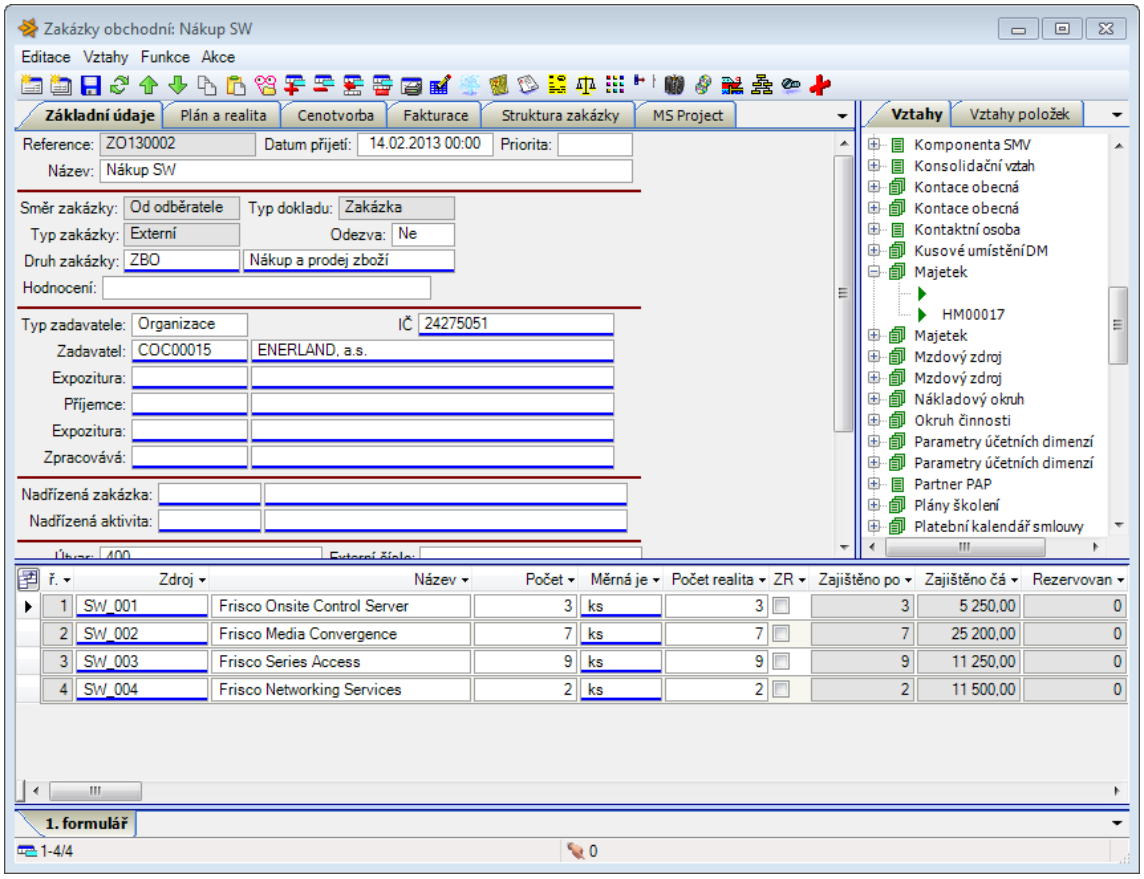
Posledním XML souborem formulářové šablony je soubor definující datovou část. Tento soubor umožňuje nadefinovat SQL dotaz, kterým bude systém získávat data. Dále také definuje výčet prvků, které se mají ukládat do databáze.[2])

Ukázka formuláře na obrázku 3.4 používá šablonu, ze které je převzata část u popisu XML dokumentu pro vizuální rozložení. Nyní popíšu, kde se na formuláři nacházejí jednotlivé prvky, které jsem popsal v předchozích odstavcích. Jedná se o formulář třídy

[1]) Mezi nejčastěji používané patří: základní editační políčko, víceřádkové editační políčko, zaškrtávací box a pole pro výběr jedné z přednastavených hodnot.

[2]) V některých případech se na formulář umisťují statistické údaje, které jsou počítány na základě jiných atributů, případně atributů jiných záznamů. Takové hodnoty není žádoucí uchovávat.

obsahující položky, které jsou umístěny ve formě tabulky ve spodní části okna, kde je možné je přímo editovat. Hlavní a největší panel zobrazuje data hlavičky. Statické vztahy (3.3.6) jsou na formuláři reprezentovány políčky s modrým podtržením. Zbylá políčka představují atributy (3.3.4). Na položkách je umístěn atribut „ZR“, který má nastavený jako editor zaškrtávací políčko. Dynamické vztahy (3.3.7) jsou umístěny ve formě stromové struktury v pravé části formuláře.



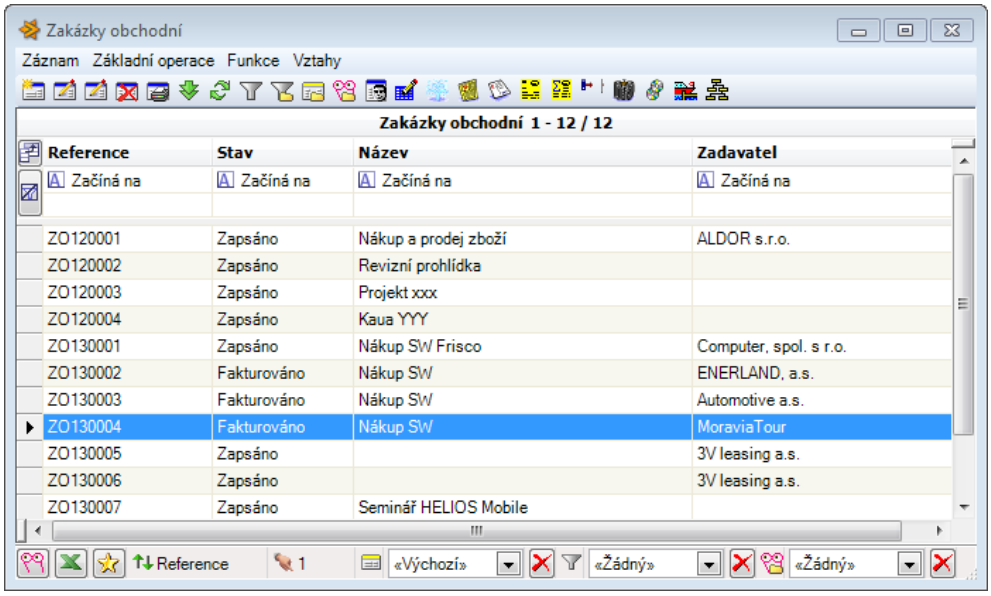
Obrázek 3.4. Formulář záznamu třídy obsahující položky.

3.3.9 Přehledové šablony

Podobně jako formulářové šablony definují, jak má systém zobrazovat data záznamu, tak přehledové šablony definují způsob zobrazení dat pořadače (3.3.3).

Obrázek 3.5 ukazuje klasické zobrazení pořadače v aplikaci. Přehledová šablona definuje, které prvky třídy se v přehledu zobrazí, tak aby mohl uživatel přehledně identifikovat záznam, který chce otevřít. Kromě prvků používaných formulářovou šablonou je ale možné v přehledu zobrazit data z položek záznamu a dokonce i ze záznamů navázaných v dynamických vztazích. Záznam je pak logicky v přehledu reprezentován více řádky.[1]) Pomocí přehledové šablony je také možné nadefinovat implicitní řazení tabulky přehledu, nastavit popisky sloupečků a jejich šířku. V okně pořadače lze provádět základní filtraci pomocí řádkových filtrů v záhlaví tabulky.

[1]) Data hlavičky jsou např. zobrazena na každém řádku, který zobrazuje data položky.



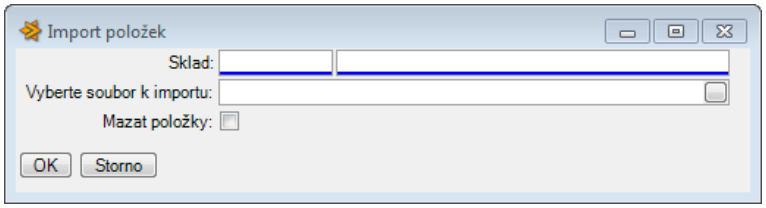
Obrázek 3.5. Okno pořadače s nastavenou přehledovou šablonou.

3.3.10 Velké filtry

Pro pokročilejší hledání záznamů slouží tzv. velké filtry. Umožňují nadefinovat složitější podmínky a filtrovat i pomocí hodnot, které nejsou zahrnuty v přehledové šabloně. Výhodou je i možnost uložit filtr pro pozdější použití. Editace a vytváření velkého filtru probíhá uživatelsky přívětivým způsobem, kdy není nutná znalost SQL, pokročilý uživatel má ale možnost SQL použít pro tvorbu komplexních filtračních podmínek. Obecně jsou velké filtry v systému HeG hojně využívány nejen pro filtraci přehledových šablon, ale i jako nástroj pro jemnější přiřazení uživatelských práv, či jako parametr funkce pro vymezení množiny vstupujících záznamů.

3.3.11 Funkce

Pojem Funkce v systému HeG označuje operaci prováděnou nad jedním[1]) nebo více záznamy. V případě jednoho položkového záznamu může být prováděna přímo nad vybranými položkami. Operace se pak sestává z vykonání nějakého kódu v knihovně či modulu, ze zavolání uložené databázové procedury nebo z provedení jednoduché uživatelem definované změny hodnot. Výsledkem volání funkce tedy může být téměř cokoliv: modifikace vstupujících záznamů, vytvoření záznamů nových nebo libovolná jiná akce s ohledem na možnost spouštění aplikačního kódu knihovny. Důležité je zmínit, že funkce může na vstupu vyžadovat i jiné udaje než jen záznamy. Pro tento účel zobrazuje funkce parametrické okno (obrázek 3.6), které je tvořeno podobným způsobem jako formulářové šablony (3.3.8).



Obrázek 3.6. Parametrické okno funkce.

[1]) Funkce může být často na vstupních záznamech nezávislá, například funkce, která má na starosti jejich generování.

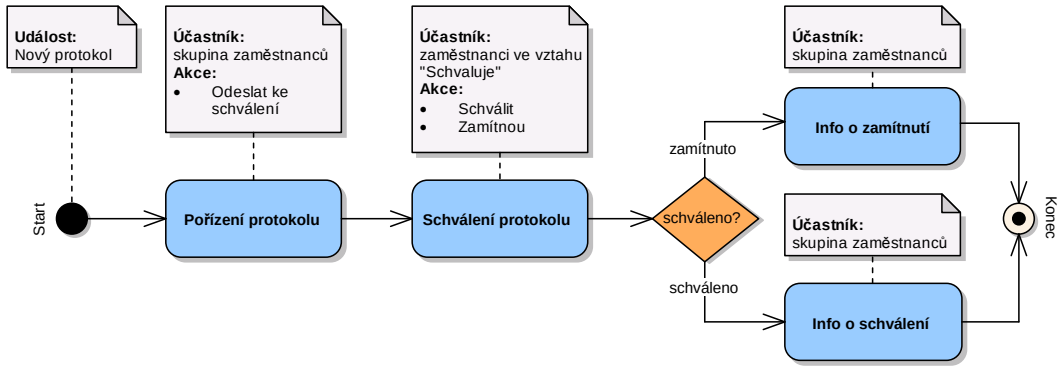
3.3.12 Uživatelé

Dalším neopomenutelným stavebním prvkem každého informačního systému, tedy včetně systému HeG, je uživatel. Uživatele jsem v předchozích odstavcích zmínil ve spojitosti s oprávněními na práci s jednotlivými komponentami. Kromě samotných přihlašovacích údajů, navázaných nejčastěji na záznam typu Zaměstnanec, je uživatel definován množinou oprávnění, podle kterých systém zobrazuje a umožňuje modifikovat data.

Důležité je také zmínit, že maximální počet současně přihlášených uživatelů je svázán licencí, kterou si daná organizace zakoupí. Maximální počet uživatelů je dvojího typu, systém rozlišuje takzvané interaktivní přihlášení (pomocí klasického klienta) a přihlášení neinteraktivní. Do druhé kategorie spadají dávkově spouštěné úlohy na serveru a právě, pro nás důležitá, přihlášení prostřednictvím aplikací třetích stran. (3.6) Počet licencovaných souběžných přihlášení této druhé kategorie bývá z pravidla nižší (v řádu jednotek).

3.3.13 Workflow

Pojem Workflow se dá doslova přeložit jako „tok práce“, což celkem přesně vystihuje účel této funkcionality v systému. Jejím úkolem je popsat a realizovat procesy při zpracování záznamů. Realizace pak spočívá v postupném přidělování úkolů jednotlivým uživatelům podle definovaného procesu, aby došlo ke zpracování záznamu v souladu s pravidly organizace. Samotný proces si lze představit jako stavový diagram, kterým záznam prochází. Hlavními součástmi procesu jsou: aktivita, akce a uzel.



Obrázek 3.7. Workflow proces - Schvalování zařazovacího protokolu.

Aktivita označuje činnost, která má být nad dokumentem vykonána. Vykonání může probíhat automaticky spuštěním funkce, která je v definici aktivity přiřazená, nebo – pro tuto analýzu důležitějším – manuálním výběrem akce z menu na formuláři záznamu.

Akce může zahrnovat spuštění funkce (3.3.11) a to včetně zobrazení okna pro zadání parametrů. V rámci aktivity může být spuštěn i jiný proces jako podproces, nebo odeslán informační email účastníkům aktivity.[1])

Posledním důležitým prvkem procesu je uzel. Uzel slouží k zahájení procesu (pouze jeden) a k jeho ukončení (i více než jeden). V ostatních případech slouží uzel k řízení toku procesu, kdy na základě výsledku předchozí aktivity rozhoduje, která aktivita bude následovat.

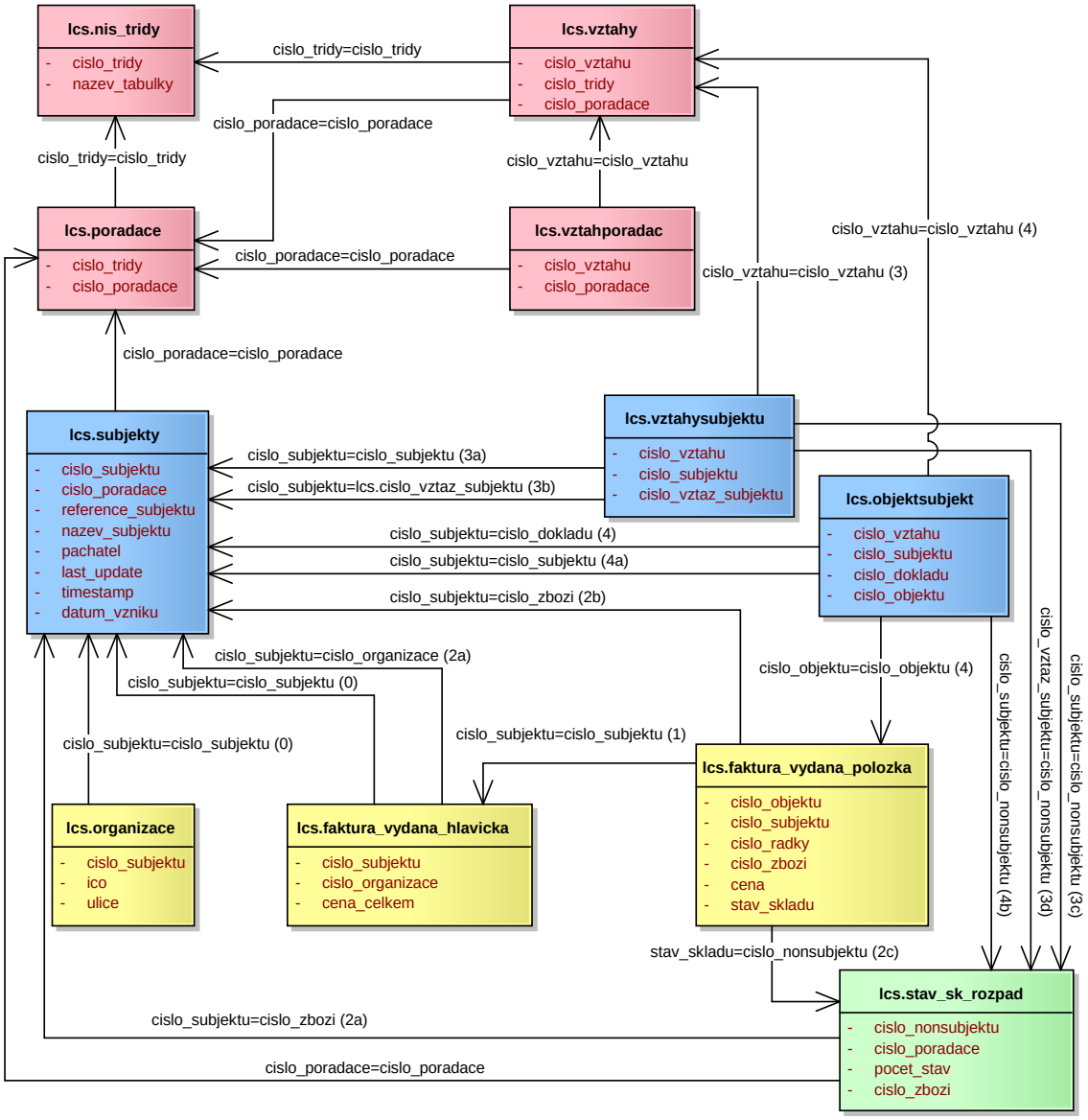
Obrázek 3.7 obsahuje diagram reálného Workflow procesu schvalování zařazovacího protokolu. Modře jsou na obrázku vyznačeny aktivity, u každé je uvedeno, kteří uživa-

[1]) Například informace pro technika, že mu byl přidělen nový úkol.

telé se aktivity účasní a zda jsou k dispozici nějaké manuální akce. Poslední dvě aktivity bez manuálních akcí slouží k odeslání informačního emailu zainteresované skupině zaměstnanců. Příklad obsahuje tři uzly. Jeden počáteční, který je spouštěný událostí založení nového protokolu. Dále jeden řídící (oranžová barva), který vybírá na základě uživatelem zvolené akce příslušnou aktivitu. Proces uzavírá třetí uzel, který je koncový.

3.4 Datový model

Předcházející oddíl se zaměřil na popis jednotlivých součástí systému tak, jak figurují v aplikační logice serveru a v uživatelském rozhraní klienta. V tomto oddílu se zaměřím na fyzický datový model systému a způsob ukládání uživatelských dat na datové vrstvě systému. Obrázek 3.8 ukazuje zjednodušený[1]) diagram datového modelu systému HeG.



Obrázek 3.8. Fyzický datový model systému HELIOS Green.

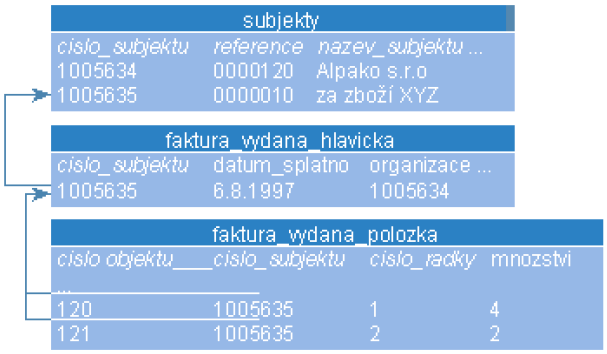
[1]) Zjednodušení spočívá v odstranění přebytečných datových polí pro zvýšení přehlednosti. Klíčové principy zůstaly zachovány.

Tabulky obsahující metadata jsou na diagramu označeny červenou barvou. Do této kategorie spadá většina součástí popsaných v předcházejícím oddílu. Máme zde tabulku *lcs.nis_tridy* pro třídy (3.3.1), *lcs.poradace* pro pořadače (3.3.3), které jsou navázány na tabulky definující vztahy (3.3.5) – základní *lcs.vztahy*, a tabulka *lcs.vztahporadac*, která vymezuje množinu pořadačů, ze kterých lze vybírat záznamy do vztahu. Žlutě jsou označeny tabulky příkladů subjektových tříd, zeleně pak příklad nonsubjektové třídy. Modrou barvou jsou označeny tabulky společné pro všechny třídy, zvláště pro ty subjektové. Datový model systému HeG rozlišuje dva typy tříd, subjektové a nonsubjektové.

3.4.1 Subjektové třídy

Specifickým znakem pro subjektové třídy je, že každá třída má garantovanou minimální množinu společných atributů a statických vztahů. Tyto vlastnosti se nacházejí v tabulce *lcs.subjekty*. Tento krok je z hlediska návrhu systému logický, protože u většiny záznamů chceme evidovat jejich název, reference[1]), datum vytvoření a poslední změny, kdo záznam vytvořil. K těmto společným datům je vhodné mít rychlý přístup, proto jsou vyčleněna do zvláštní tabulky. Rychlý přístup k těmto údajům je žádoucí hlavně z důvodu, že reference a název jsou zobrazovány, pokud záznam figuruje ve vztahu. Při načítání záznamu, který se odkazuje na větší množství množství jiných záznamů, pak dotazování do jedné subjektové tabulky (namísto mnoha specifických tabulek) představuje nižší zátěž pro systém.

Specifická data třídy (například u organizace chceme evidovat její IČO, adresu atd.) jsou ve vlastní tabulce třídy (v tomto případě *lcs.organizace*) a na společnou tabulku se vážou pomocí cizího klíče (vazby s číslem 0 u žlutých tabulek na obrázku 3.8). Z této vazby si můžeme odvodit, že všechny subjektové třídy sdílí společnou řadu identifikátoru (*cislo_subjektu*), žádné dva záznamy dvou libovolných subjektových tříd nemají stejný identifikátor.



Obrázek 3.9. Diagram uložení záznamu dokladové (položkové) třídy. Obrázek převzat z [2].

Subjektové třídy mohou být, na rozdíl od tříd nonsubjektových, takzvanými „dokladovými“ třídami. Tento druh tříd jsem zmínil již v oddílu o třídách (3.3.1), jedná se o třídy, které umožňují, aby měly záznamy uloženy řádky položek. Pro data položek má každá dokladová třída specifickou tabulku, v diagramu se jedná o tabulky *lcs.faktura_vydana_hlavicka* pro hlavní data faktur vydaných a *lcs.faktura_vydana_polozka* pro data řádek faktur vydaných. Reálná data jsou ilustrována atributem *cena*, který udává cenu na každém řádku faktury zvlášť, a atributem

[1]) Jedná se o „druhý název“, často reference obsahuje evidenční číslo, číslo faktury, kód produktu, či jiný lidsky čitelný a dostatečně unikátní identifikátor záznamu. Může být automaticky generována na základě konfigurace pořadače.

cena_celkem na hlavičce, který je součtem cen přes všechny řádky. Identifikátorem položek je *cislo_objektu*, které je unikátní v rámci každé tabulky položkových dat. Vazba mezi daty položek a hlavičkou je tvořena pomocí *cislo_subjektu* (na obrázku s číslem 1). Kromě identifikátoru a vazby na hlavičku musí každá položka obsahovat atribut s číslem řádky, které může být uživatelem editováno. Obrázek 3.9 obsahuje příklad uložení záznamu dokladové třídy se dvěma položkami.

3.4.2 Nonsubjektové třídy

Na rozdíl od subjektových tříd mají nonsubjektové třídy uložena veškerá data záznamů v jedné tabulce a nejsou zde podporovány položky. Příkladem budiž zelená tabulka *lcs.stav_sk_rozpad* pro data záznamů tříd „rozpad stavu skladu“. Záznamy tohoto typu se v praxi používají k detailní evidenci pohybů na skladě. Mohou být použity k hlídání exspirace, sledování jednotlivých šarží a dalších údajů. Rozpady vznikají téměř při každém pohybu na skladě, existuje jich tedy několikanásobně větší počet, než je počet produktů, se kterými organizace pracuje.

Rozpad stavu skladu je typickým příkladem situace, kdy je vhodné použít nonsubjektovou třídu. Práce se záznamy nonsubjektových tříd je rychlejší, pracuje se pouze s jedinou tabulkou. Dále má tato tabulka vlastní identifikátor *cislo_nonsubjektu*, u kterého je požadována unikátnost pouze v rámci tabulky. To nám omezuje plýtvání hlavními identifikátory (*cislo_subjektu*) u tříd, kde předpokládáme časté mazání a vytváření záznamů. Jak jsem již poznamenal, nonsubjektové třídy nemají mezi sebou žádnou minimální množinu společných atributů a statických vztahů, kromě sloupce identifikátoru musí tabulka obsahovat *cislo_poradace*. Speciálním případem jsou atributy název a reference, v metadatech konkrétních atributů nonsubjektů můžeme zvolit, že daný atribut zastupuje název a/nebo referenci. Systém pak zobrazuje hodnoty těchto atributů v případě, že je záznam nonsubjektové třídy navázán do vztahu.

3.4.3 Vztahy

Následuje rozšíření informací zmíněných v popisu metadat vztahů (3.3.5). Statické vztahy jsou řešeny prostým uložením identifikátoru navázaného záznamu do příslušného sloupce vztahu v tabulce. Na obrázku 3.8 jsou uvedeny tři příklady statického vztahu. První (2a) demonstruje statický vztah z faktury vydané na záznam organizace, logicky přes tabulku společných hodnot subjektových tříd (*lcs.subjekty*). Stejně je tomu tak u dat z položek faktury, které odkazují na karty zboží[1]) (2b). Pro statické vztahy na nonsubjekty (2c) platí stejný systém, jen logicky ukazují přímo do příslušné tabulky nonsubjektové třídy.

Integrita vztahů není hlídána na úrovni datové vrstvy, ale až na úrovni vrstvy aplikační logiky. Jedním z důvodů je rozdílná parcialita a kardinalita v závislosti na metadatech vztahů, kterou může dále správce měnit za běhu. Dalším důvodem je takzvaná typovost vztahů, kdy nám metadata říkají, ve které tabulce hledat navázaný záznam, zda v tabulce *lcs.subjekty*, nebo (v případě nonsubjektových tříd) v jejich konkrétní tabulce. Z této skutečnosti, spolu s faktem, že v datech záznamu je uložený pouze identifikátor, vyplývá, že vztah může ukazovat na libovolný počet subjektových tříd nebo pořadačů, nebo pouze na jednu konkrétní nonsubjektovou třídu a její pořadače. Toto se vztahuje i na způsob ukládání dynamických vztahů. Vztah může přesto v metadatech povolovat konfiguraci v rozporu s předchozí větou, aplikační logika (3.5) ale musí dohlédnout, aby nedošlo k uložení neplatné kombinace do databáze.

[1]) Tabulka specifických dat pro kartu zboží není v diagramu zahrnuta.

Pro ukládání dynamických vztahů používá systém HeG zbývající dvě (dosud nezmiňované) modré tabulky z obrázku 3.8: *lcs.vztahysubjektu* a *lcs.objektsubjekt*. Obě tabulky jsou si velmi podobné, definují levou stranu vztahu, pravou stranu vztahu a identifikátor metadat vztahu. Princip vztahu mezi levou a pravou stranou je obdobný jako v případě na obrázku 3.3. Tabulka *lcs.objektsubjekt* je určena výhradně pro vztahy s položkou dokladové třídy na levé straně a není možné v uživatelském rozhraní systému interpretovat vztah z pravé strany. Podporovány jsou následující kombinace levé a pravé strany[1]):

- subjekt (3a) – subjekt (3b),
- subjekt (3a) – nonsubjekt (3d),
- nonsubjekt (3c) – subjekt (3b),
- nonsubjekt (3c) – nonsubjekt (3d),
- položka (4) – subjekt (4a),
- položka (4) – nonsubjekt (4b).

3.4.4 Datové typy

Datový typ identifikátoru a tedy i vztahových sloupců je 32bit celé číslo. Datové typy pro atributy volí autoři třídy tak, aby došlo ke splnění požadavků na přesnost u číselných hodnot a také s ohledem na tabulkou využívané místo – volí se tedy rozumné délky řetězcových polí.[2]) Mezi nejčastěji využívané datové typy používané v databázi (MS SQL Server) patří: smallint, int, numeric (decimal), char, varchar a datetime. V případě aplikační logiky a vymezení se na jazyk C#, pomocí kterého je systém HeG implementován, je dostačující výčet: int, decimal, string a DateTime.

3.5 Aplikační logika

Dosud jsem pouze rozebíral systém HeG z hlediska struktury dat a oblast „chování“ systému jsem zmínil jen okrajově. Za každou rozsáhlejší implementovanou agendou stojí ale mnoho řádek kódu v knihovnách aplikačního serveru. Rozšíření standardní funkcionality se, jak již vyplývá z architektury systému (3.1), implementují na platformě Microsoft .NET Framework. Nejčastěji se aplikační logika rozšiřuje na dvou místech – rozšířením událostí záznamu a výkonným kódem pro funkce.

Pro každou třídu (3.3.1) může její autor definovat .NET třídu v knihovně (tzv. instanci), ve které je možné reagovat na různé události záznamu. Události vnikají při mnoha operacích se záznamem, ať již se jedná o filtrování vhodných záznamů při vyplňování jeho vztahů, nebo vytváření, načítání či ukládání záznamu. Na většinu může autor reagovat před, ale i po jejich vykonání systémem, a je tak možné například provádět pokročilejší kontroly správnosti zadávaných údajů. V případě událostí týkajících se nově vyplněného vztahu je k dispozici informace o třídě záznamu. Toto se používá například u záznamů objednávek, když je vztah na zboží definován tak, že umožňuje navázat jak subjektový záznam (kmenová karta) tak i nonsubjektový záznam (rozpad stavu skladu), což je v rozporu s typovým uložením vztahů popisovaným v 3.4.3). Proto v události po navázání záznamu dochází ke kontrole a v případě záznamu typu rozpad stavu skladu k dohledání příslušné kmenové karty, kterou je rozpad ve vztahu nahrazen.

V případě kódu funkcí má autor rovněž nástroje pro kontrolu zadávaných parametrů, jako je tomu v případě formulářů záznamů. Samotné tělo funkce pak může vykonávat

[1]) Čísla v závorkách odpovídají vazbám na obrázku 3.8.
[2]) Je například zbytečné, aby pole pro telefonní číslo mělo délku 100 znaků.

jakoukoliv funkcionalitu v rámci frameworku systému HeG (práce se záznamy), kde na rozdíl od databázových procedur může korektně reagovat aplikační logika. Samozřejmě je možné využít knihovny mimo systém HeG, komunikovat se softwarem a webovými službami třetích stran.

3.6 Webová služba ServiceGate

V oddílu 3.1 jsem uvedl, že tenký klient komunikuje s aplikačním serverem prostřednictvím webové služby. Služba, kterou používá, ale není veřejně zdokumentovaná. V případě, že bych se pokusil službu analyzovat za účelem použití pro mého mobilního klienta, nemám žádné záruky, že se služba radikálně nezmění s příchodem dalších verzí. Proto se nyní zaměřím na oficiálně poskytovanou a zdokumentovanou webovou službu poskytovanou aplikačním serverem – rozhraní ServiceGate.

3.6.1 Protokol SOAP

Webová služba ServiceGate používá jako základní vrstvu komunikace protokol SOAP. Tento bezstavový protokol představuje framework pro výměnu zpráv ve formátu XML. Protokol je snadno rozšiřitelný a je nezávislý na programovacím jazyku implementace. Umožňuje vzdálené volání operací na serveru výměnou těchto zpráv prostřednictvím široké palety transportních protokolů (HTTP, SMTP, TCP, UDP a další). Nejčastěji používaným transportním protokolem je HTTP, díky kterému poskytuje spolehlivý přenos dat a s ohledem na jeho rozšířenost také dobrou prostupnost firewally. [4]

3.6.2 Rozhraní služby

Samotné rozhraní služby pracuje na principu zpracování dávkových příkazů ve formě XML zpráv, které jsou zabaleny jako textový parametr SOAP operace. Výsledek služba vrací rovněž ve formě XML textu obsaženém v odpovídající struktuře SOAP zprávy. Aby byl zachován princip fungování systému HeG, kdy se uživatel na začátku práce přihlašuje a po celou dobu jeho práce existuje na serveru jeho relace, služba ServiceGate obchází bezstavovost protokolu SOAP pomocí operací pro přihlášení, odhlášení a volání všech operací s identifikátorem relace. Celé rozhraní webové služby se skládá z těchto čtyř operací:

- LogOn – operace provede přihlášení podle poskytnutých údajů v roli příslušného uživatele na daný databázový profil. Výsledkem operace je identifikátor relace, pokud bylo přihlášení úspěšné.
- ProcessXml – operace slouží k vykonání dávky v rámci relace na základě předané XML sekvence. Výsledkem operace je příslušná odpověď ve formátu XML.
- LogOff – odhlášení přihlášené uživatelské relace.
- KeepAlive – informace pro server, že má prodloužit platnost relace. Standardně je relace platná 300 sekund od poslední zavolané operace, tato hodnota se může lišit v závislosti na konfiguraci systému.

XML zprávy přijímané operací ProcessXml pak nabízejí širokou paletu operací pro práci na systémové úrovni jednotlivých součástí, bez ohledu na konkrétní agendu tak, jak jsem je představil v 3.3. Seznam podporovaných zpráv je v tabulce 3.1.

Důležitým aspektem webové služby ServiceGate je respektování aplikační logiky. Většina uvedených operací vrací nebo na vstupu přijímá XML strukturu reprezentující záznam včetně všech vztahů a atributů, která zároveň při svém vzniku a nebo při propisování zpět do systému spouští náležité události (v některých případech volitelně) a jejich obslužný kód v aplikační logice.

Požadavek	Popis
NEW	Pro požadovanou třídu a pořadač vrátí nový záznam s předvyplněnými hodnotami.
INSERTUPDATE	Slouží k založení nebo aktualizaci záznamů.
RETRIEVE	Vrátí celé záznamy načtené z databáze.
DELETE	Smaže záznam a vrátí výsledek smazání.
MERGE	Sloučí více záznamů do jednoho a vrátí výsledek.
BROWSE	Na vstupu očekává identifikátor přehledové šablony (3.3.9), číslo pořadače a případně velký filtr. Vrátí řádky odpovídající standardnímu použití přehledové šablony a velkého filtru v klientské aplikaci. Rozsah řádků (od, do) je možné vymezit.
RUN	Spustí funkci (3.3.11) nad zvolenými záznamy a vrátí výsledek.
WORKFLOW	Spustí workflow akci (3.3.13) nad zvolenými záznamy a vrátí aktuální seznam workflow.
PRINT	Provede tisk na aplikačním serveru.
QUERYSELECT	Umožňuje na databázi zavolat SQL dotaz nebo spustit proceduru. Výsledkem může být množina záznamů (dotaz vrátil jejich identifikátory), nebo surová data odpovědi na dotaz ve formě tabulky.

Tabulka 3.1. Systémové XML zprávy pro operaci ProcessXml webové služby ServiceGate.

3.7 Rozšiřitelnost existujících řešení

Tato kapitola se primárně zaměřovala na situaci, kdy máme hotové řešení, ať již přímo v základní instalaci, či vyvinuté naší společnosti na míru. Existují ale případy, kdy je pro společnost existující implementace agend téměř vyhovující, ale pro plné podpoření existujících procesů chybí důležité atributy, vztahy nebo funkce. Systém HeG obsahuje několik nástrojů pro řešení takových situací.

Pokud je potřeba třídu rozšířit o atribut nebo statický vztah, je vhodným prostředkem vytvoření tzv. UDA tabulky. Tato tabulka obsahuje dodatečné sloupce a pomocí cizího klíče se váže na hlavní tabulku třídy. Jakmile je tabulka nastavena v metadatech třídy a dojde k vytvoření potřebných záznamů s metadaty pro atributy a statické vztahy, je možné s nimi v systému pracovat v podstatě rovnocenným[1]) způsobem jako s atributy a statickými vztahy obsaženými ve třídě již v základu.

V případě dynamických vztahů a funkcí je situace jednoduší, zde stačí založit příslušné a záznamy metadat a v případě funkce poskytnout výkonnou část (knihovnu, SQL proceduru, definovanou změnu).

Speciálním případem je rozšíření kódu reagujícího na události záznamu. Například chceme v návaznosti na vytvoření nového záznamu dané třídy dohledat a náležitě pozměnit nějaký jiný existující záznam. V takovém případě je vhodné použít mechanismus zvaný „ClassExtender“. Jedná se o .NET třídu, která při správném zaregistrování v metadatech třídy může reagovat na události záznamu. Pak je možné v události po uložení záznamu implementovat uvedený příklad.

[1]) Hlavním rozdílem je, že UDA atributy a statické vztahy musí být adresovány plným databázovým jménem sloupce, aby se zabránilo kolizím názvů s existujícími sloupci.

Důležité je, že zmíněná rozšíření se po nasazení stávají transparentní součástí konfigurace systému a ten se stará, aby byly korektně obslouženy, například při práci přes rozhraní ServiceGate.

Kapitola 4
Analýza a návrh řešení

4.1 Rozbor jednotlivých agend

V kapitole 3 jsem popsal klíčově vlastnosti systému HELIOS Green. Také jsem uvedl, že veškeré agendy podpořené systémem lze dekomponovat na základní stavební prvky. Nyní se zaměřím na jednotlivé agendy zmíněné v zadání této práce, abych mohl identifikovat minimální požadavky pro implementaci mobilního klienta. Následující oddíly popisují scénáře tak, jak byly na základě mých doplňujících dotazů specifikovány společností ALDOR.

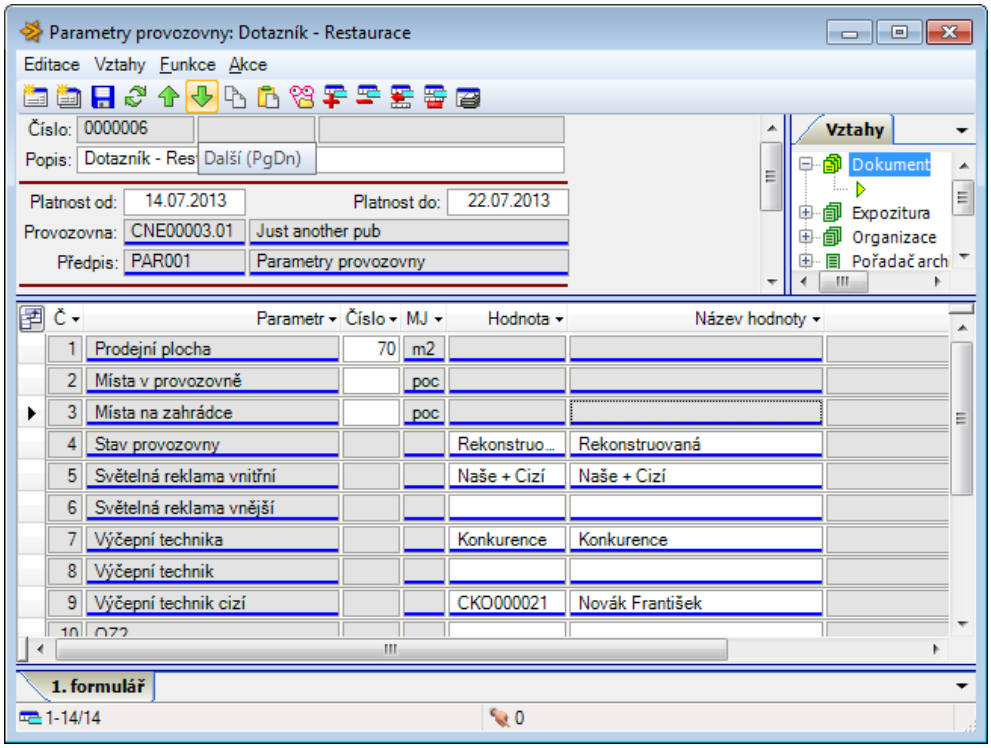
4.1.1 Vytváření objednávek (cenotvorba)

V případe obchodníků pracujících v terénu je hlavním úkolem vytvářet objednávky na základě přání zákazníka. Při vytváření objednávky se neberou vždy ceny z pevně daných ceníků, ale často dochází k úpravě celkové ceny objednávky na základě obsahu vytvářené objednávky (například při objednání většího počtu kusů) nebo na základě historie objednávek zákazníka (věrnostní program). Tento přepočet probíhá během události uložení změn záznamu. Třída objednávky je dokladová a obsahuje tedy položky. Samozřejmostí je nutnost navazovat statické vztahy a to včetně záznamů nonsubjektových tříd (situace popsaná v 3.5).

Kromě vytváření objednávek se od obchodníků očekává, že budou aktualizovat informace o zákaznících vyplňováním speciálních dotazníků. Data dotazníků pak poslouží pro lepší přehled o zákaznících a lepší zacílení marketingových akcí. Vyplnění těchto dotazníků probíhá na základě pozorování samotného obchodníka nebo přímo dotazováním zákazníka. Ukázka záznamu dotazníku je na obrázku 4.1. Na první pohled se jedná o dokladovou třídu, ale zároveň je vidět, že některé položky mají atribut „Číslo“ povolený pro zápis spolu s vyplněným vztahem měrné jednotky („MJ“) a jiné mají pro zápis povolený statický vztah „Hodnota“. Dále se liší množina záznamů, které je na položku do statického vztahu „Hodnota“ možné nastavit, například řádek číslo 9 očekává navázání záznamu typu Kontaktní osoba, zatímco řádek číslo 4 povoluje výčet hodnot týkajících se stavu provozovny. Chování položek reflektuje definici navázanou ve vztahu „Prametr“, ta konfiguruje, jaký typ hodnoty se na položku vyplňuje, případně jaké záznamy je povoleno navazovat.

Pro tento scénář je nutné, aby mobilní aplikace podporovala:

- položkové třídy,
- statické vztahy,
- navazování záznamů nonsubjektových tříd,
- větší objemy dat (katalog produktů),
- respektování aplikační logiky spouštěné na serveru (přepočet ceny objednávky),
- funkcionalitu dotazníků (specifické implementace chování záznamu pro konkrétní třídy).



Obrázek 4.1. Záznam dotazníku vyplňovaný obchodníkem.

4.1.2 Vykazování práce techniků v rámci servisních zásahů

Třída záznamu, který technik během/po výjezdu musí vyplnit se nazývá „Servisní práce“ a slouží pro evidenci času stráveného prací, její sazbu a další údaje, které pak slouží jako podklad pro fakturaci. Je vhodné minimalizovat úkony, které technik musí se záznamem o servisním zásahu provádět. (Nutnost vyplňování vtahů a atributů.) Dále je záznam součástí workflow procesu, v němž některé akce vykonávají funkci, která vyžaduje zadání parametrů. Kromě „Servisní práce“ technik pracuje i s jinými druhy záznamů, například vytváří záznamy o zápůjčkách, případně pořizuje fotodokumentaci závad. Na některých záznamech se pracuje s dynamickými vztahy. Technici se také často vyskytují na místech s nekvalitním, nebo žádným mobilním signálem, typickým příkladem jsou podzemní místnosti se servery a datová centra. Proto by aplikace měla být schopná pracovat bez dostupného připojení na aplikační server HeG.

Pro tento scénář je nutné, aby mobilní aplikace podporovala:

- statické vztahy,
- dynamické vztahy,
- větší objemy dat (karty opravovaných či zápůjčních zařízení),
- výchozí hodnoty,
- funkce s parametrickým oknem,
- nahrávání příloh (fotografie),
- workflow,
- funkčnost bez dostupného spojení na server.

4.1.3 Sběr dat v terénu se záznamem pozice pomocí GPS

Pro tuto agendu předpokládejme modelový scénář odečtu stavu elektroměrů. Pracovník společnosti tráví kvůli odečítání stavů elektroměrů podstatnou část své pracovní doby

v terénu. Je tedy žádoucí, aby byly sbírané údaje jednoduchým a rychlým způsobem nahrávány do informačního systému k dalšímu zpracování. Zároveň je vyžadováno, aby byla v momentě odečtu zachycena přibližná poloha GPS pro kontrolní účely. Záznam o odečtu kromě této pozice musí obsahovat kromě informacích o zákazníkovi také čas zápisu, který není třeba ručně zadávat. Zařízení by mělo dále usnadnit našemu pracovníkovi práci například tím, že bude obsahovat kontaktní údaje zákazníka pro případ, kdy je například omezený přístup k měřící jednotce.

Z tohoto scénáře vyplývají následující požadavky na mobilní aplikaci:

- statické vztahy,
- atributy s dynamickou výchozí hodnotou (aktuální čas),
- záznam GPS souřadnic (ruční nebo automatický),
- práce s kontaktními údaji.

4.1.4 Shrnutí

Následuje souhrn požadavků na výslednou aplikaci. Pro lepší přehlednost jsem požadavky rozdělil do tří kategorií, abych oddělil požadavky na základní součásti (3.3) systému HELIOS Green, požadavky na vlastnosti aplikace z hlediska základní funkcionality a na požadavky týkající se lepší přívětivosti aplikace.

- Součásti systému HELIOS Green:
  - subjektové třídy včetně tříd s položkami (dokladové třídy),
  - nonsubjektové třídy,
  - atributy,
  - statické vztahy,
  - dynamické vztahy,
  - funkce s parametrickým oknem,
  - workflow.
- Základní požadavky na mobilní aplikaci:
  - práce v offline režimu,
  - práce s většími objemy dat,
  - respektování aplikační logiky spouštěné na serveru,
  - podpora pro implementaci interaktivní části aplikační logiky (např. dotazníky),
  - záznam GPS souřadnic,
  - nahrávání příloh (fotografie).
- Požadavky na mobilní aplikaci zlepšující použitelnost a přívětivost:
  - statické výchozí hodnoty atributů a statických vztahů,
  - dynamické výchozí hodnoty atributů,
  - speciální zacházení s atributy použitelnými na mobilním zařízení.[1])

4.2 Koncept mobilní aplikace

Na základě požadavků, které jsem identifikoval v předcházejícím oddílu byl vytvořen základní koncept mobilní aplikace. V jeho popisu se zaměřím především na druhou kategorii požadavků – na základní požadavky.

[1]) Například možnost vytočit telefonní číslo z atributu podobným způsobem, jako to umožňuje klient Microsoft Dynamics CRM (2.1.1).

4.2.1 Lokální kopie dat

Jedním ze základních požadavků je možnost pracovat s daty bez dostupného připojení k serveru HELIOS Green (dále jen HeG). Je tedy nutné, aby aplikace měla uloženou lokální kopii dat, se kterými je potřeba pracovat i v případě nedostupného přípojení a která bude synchronizována se serverem. Z důvodu obvykle nižší rychlosti mobilních datových spojení byl jako vhodnější zvolen princip lokální synchronizované kopie i v případě dostupného spojení. Například takové načítání seznamu záznamů při vyplňování vztahů by výraznějším způsobem zpomalovalo odezvu uživatelského rozhraní mobilního klienta. Dalším důvodem je potenciálně složitější údržba a rozvoj aplikace v případě umožnění režimu lokální kopie pro offline režim zároveň s možností provozovat aplikaci v online režimu s přímým spojením na server systému HeG. Kromě stránky vývoje a údržby by podpora dvou režimů znamenala nutnost návrhu systému, pomocí kterého by uživatel explicitně vybíral, která data chce mít v offline režimu k dispozici. Považuji tedy koncept lokální kopie dat synchronizovaných se serverem HeG za obecně vhodnější, protože absenci výhod online přístupu můžeme kompenzovat možností pravidelné synchronizace, či automatické synchronizace po provedení změn v datech na zařízení.

Velikost databáze systému HeG u zákaznických instalací běžně dosahuje několika GB, u středně velkých a větších organizací není ojedinělá ani velikost v řádu desítek GB. Je tedy nemyslitelné, aby se na zařízení vytvářela kompletní kopie těchto dat. Pravidlem také bývá, že zaměstnanec většinou pracuje pouze s malou podmnožinou dat systému, která se týká oblasti jeho působení v organizaci. Toto platí obzvláště s přihlédnutím ke skutečnosti, že se v tomto zadání zaměřujeme na konkrétní agendy, u kterých je možné snadno určit pořadače a data, která jsou pro danou agendu relevantní. Rozhodně ale nebylo cílem vytvořit aplikaci, která konkrétně implementuje zadané agendy. Místo toho má aplikace konfigurovatelnou platformou, na kterou bude možné jednoduše přenést jakoukoliv agendu, jejíž základní požadavky budou podmnožinou požadavků z analýzy v oddílu 4.1.

Kromě samotného výběru jednotlivých pořadačů se zde nabízí ještě možnost filtrovat data přímo s ohledem na konkrétního uživatele. Filtrace je vhodná pro agendu servisního technika (4.1.2), například každý technik může mít ve svém mobilním zařízení k dispozici tikety přiřazené jeho uživateli a dále pouze data vztahující se k organizacím, které má na starosti. U větších organizací přichází v úvahu filtrování dat na základě regionální působnosti uživatele.

Možnost vymezit rozsah dat je nutné zvážit nejen na úrovni pořadačů a množství záznamů, ale také přímo na úrovni struktury jednotlivých záznamů. Již nejednou zmíněná třída pro fakturu vydanou má například ve verzi 43.52.50 na hlavičce 128 atributů a 36 statických vztahů a pro každou položku pak 47 atributů a 16 statických vztahů. Takto velké množství údajů je dáno především obecným návrhem této základní třídy. Záleží vždy na konkrétním zákazníkovi, které údaje potřebuje evidovat a které je možné naopak skrýt a nepoužívat. Dále se používaná pole mohou lišit v závislosti na rolích jednotlivých uživatelů v procesu zpracování záznamu. Pro servisního technika je dostačující, když má u záznamu organizace uvedeny základní údaje, včetně těch kontaktních. Nezajímá ho, že organizace používá pro nákupy kreditní systém, ale tato informace může být naopak důležitá pro obchodníka. Z důvodu snížení velikosti lokální kopie dat by měla aplikace umožnit výběr atributů a vztahů, se kterými uživatel mobilního zařízení potřebuje v rámci dané agendy pracovat a zbylé pole do synchronizace vůbec nezahrnovat.

4.2.2 Aplikační logika

Přenos aplikační logiky do mobilního klienta není tak jednoduše proveditelný, jako je tomu v případě dat záznamů. Vzhledem k široké škále možností, kterými může v rámci instance (3.5) autor třídy chování záznamu ovlivnit, bude nutné přistoupit ke kompromisům. Z mých zkušeností s prací se systémem HeG a s jeho konfigurací vyplývá, že většina pro agendy ze zadání klíčových operací se odehrává při ukládání záznamu, případně spouštěním funkcí nad záznamy, či voláním workflow akce. Časté je i doplnění atributů a vztahů v návaznosti na vyplnění jiného atributu či vztahu.

V oddílu o webové službě ServiceGate (3.6) je jasně uvedeno, že operace prováděné přes službu spouštějí události aplikační logiky. Nemělo by tedy být problémem zařídit, aby mohl kód instance správně zareagovat na uživatelem provedené změny. Kompromisem v tomto případě je fakt, že ke spuštění událostí dochází až v momentě komunikace se serverem HeG, tedy na základě rozhodnutí z předchozích odstavců – při synchronizaci lokální kopie dat. Proto u agend, kde se předpokládá častější práce v offline režimu, je nutné zkontrolovat, zda nevznikají situace, kdy má absence připojení za následek znemožnění práce uživatele.

Specifickým případem z hlediska aplikčaní logiky je třída dotazníku představená v oddíle 4.1.1. Zde se kód instance stará o celou popsanou logiku dotazníku a dynamicky mění vzhled a způsob vyplňování jednotlivých položek. Není tedy možné přesunout odpovědnost na server. Pro tento případ je třeba vytvořit na klientovi koncept instance podle vzoru aplikačního serveru HeG. V rámci této instance bude možné implementovat funkcionalitu, bez které by nebylo možné agendu na mobilní zařízení přenést.

Pro funkce je situace jednodušší, ty se v popsaných agendách vykonávají vždy na straně serveru. Proto stačí, aby mobilní klient po sběru všech potřebných parametrů zavolal příslušnou operaci webové služby ServiceGate.

Drobnou komplikaci představuje typovost ukládaných vztahů na základě metadat (3.4.3) spolu s možností reagovat na třídu vyplňovaného záznamu v rámci aplikační logiky a provádět přepis navazované hodnoty (3.5). Pro mobilního klienta s lokální kopií dat to znamená, že informaci o třídě záznamu navázaného ve vztahu je nutné udržet až do okamžiku předání dat serveru HeG, tedy do provedení synchronizace. I toto tedy bylo nutné zohlednit při návrhu datového modelu mobilního klienta.

4.2.3 Práce s přílohami

Přílohy jsou systémem HeG podporovány již v základní konfiguraci. Pro práci s přílohami je klíčová třída „Externí dokument“ a další související třídy. Záznamy této třídy pak reprezentují jednotlivé soubory příloh, obsahují název a umístění souboru spolu s dalšími informacemi o souboru, jako je například jeho velikost. Pro přiložení ke konkrétním záznamům se používají klasické dynamické vztahy (3.3.7). Fyzické úložiště příloh se konfiguruje prostřednictvím poskytovatelů na úrovni jednotlivých pořadačů externích dokumentů. Kromě souborového systému je možné ukládat soubory přímo do databáze. Aplikační logika této třídy ale odstiňuje uživatele od těchto skutečností a poskytuje standardní rozhraní pro nahrávání a otevírání příloh bez ohledu na použitého poskytovatele. Obdobně je tomu tak mimo uživatelské rozhraní klienta: nad třídou existují dvě funkce, jedna pro nahrávání a druhá pro stahování příloh. Obě funkce je možné volat prostřednictvím webové služby ServiceGate.

Další možností pro práci s přílohami v prostředí systému HeG je modul EDM, který podporuje pokročilou správu dokumentů, jejich verzování, spolupráci při jejich vytváření a v neposlední řadě lepší integraci do workflow (např. schvalování dokumentů). Stejně jako v případě externích dokumentů modul obsahuje funkci pro nahrávání a

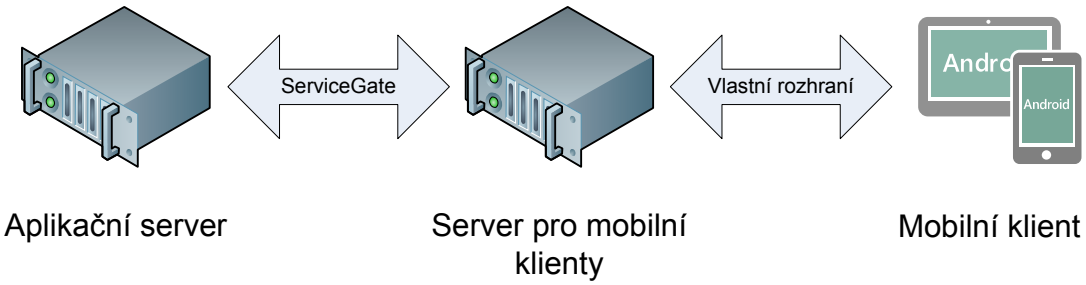
funkci pro stahování dokumentu. Signatura funkcí je shodná s těmi u externích dokumentů, proto nasazení modulu EDM namísto externích dokumentů neznamená zásadní problém z hlediska implementace požadavků agendy servisního technika (4.1.2).

4.2.4 Konfigurace

Na několika místech tohoto oddílu jsem zmínil potřebu mobilní aplikaci konfigurovat. Jednalo se o výběr pořadačů, vymezení záznamů a množiny atributů a vztahů pro synchronizaci. Toto nastavení by mělo být centrální, konfigurace každého klienta zvlášť by znamenala zbytečnou časovou zátěž pro administrátora. Nabízí se tedy umístění konfigurace přímo do systému HeG ve formě vlastní třídy. Kromě možnosti využít pro vybrané pořadače existující filtrační mechanismy (3.3.10) na vymezení rozsahu záznamů pro synchronizaci, bude možné využít správu uživatelských oprávnění systému HeG a také pro administrátory již známé uživatelské rozhraní.

4.3 Vlastní server

Po detailnější analýze rozhraní webové služby ServiceGate (3.6) jsem došel k závěru, že struktura XML zpráv požadavků a odpovědí je s ohledem na data požadovaná mobilním klientem zbytečně komplexní. Například zmiňované vyloučení polí (4.2.1), která jsou pro danou agendu na mobilním zařízení zbytečná, aby došlo k úspoře přenášených dat, vyžaduje obsáhlou specifikaci těchto polí při každém požadavku na data záznamu. Výsledná úspora dat celkové komunikace se tak stává diskutabilní. Rozhodl jsem se proto (ale také z dalších důvodů, viz dále) pro umístění dalšího článku do komunikace mezi systémem HeG a mobilním klientem. Tento vlastní server bude mít za úkol „překládat“ požadavky z mobilního klienta zaslané přes vlastní rozhraní do operací prováděných proti rozhraní webové služby ServiceGate. Vlastní rozhraní serveru je pak možné navrhnout tak, aby co nejvíce odpovídalo požadavkům mobilního klienta a nedocházelo k přenosu zbytečných informací. Schéma popsané komunikace ilustruje obrázek 4.2.



Obrázek 4.2. Schéma komunikace při použití vlastního serveru.

Omezený rozsah vlastního rozhraní je dále možné vnímat jako přínos pro celkovou bezpečnost vytvářeného řešení. Předpokládejme, že by mobilní klient komunikoval prostřednictvím internetu přímo se serverem systému HeG. To by znamenalo, že by bylo nutné veřejně vystavit rozhraní ServiceGate a s ním i stroj, na kterém běží aplikační server systému HeG. Toto často bývá v přímém rozporu s interními pravidly pro informační bezpečnost a není tedy z pohledu organizací žádoucí.[1]) Oproti tomu vlastní server může být umístěný na jiném stroji například v demilitarizované zóně, mimo lokální síť organizace s povoleným přístupem pouze na webovou službu ServiceGate na

[1]) Vycházím z osobních zkušeností a z rozhovorů se zákazníky provedených v rámci vzniku této analýzy.

aplikačním serveru HeG. Vlastní server pak může dále kontrolovat vykonávané operace, zda odpovídají konfiguraci agendy a nedochází k zápisu do nepovolených údajů či pořadačů.

Další výhodou vlastního serveru je možnost řídit přístup k webové službě ServiceGate ve smyslu přihlašování uživatelů. V oddílu 3.3.12 jsem zmínil, že systém HeG z hlediska licence rozlišuje maximální počet uživatelů pro dva režimy přihlašování: interaktivní a neinteraktivní. Přihlašování pomocí webové služby ServiceGate spadá do druhé kategorie, tedy do té, pro kterou se obecně v rámci licence pořizuje menší počet souběžných uživatelů. Systém tento limit hlídá a nedovolí přihlášení většího množství uživatelů. Vlastní server může s tímto limitem pracovat, moderovat na jeho základě přístup klientů ke službě a zabránit tak zbytečným chybovým situacím, které by znemožňovaly rozumné použití aplikace a nutily organizaci k nákupu dodatečných licencí.

4.3.1 Platforma

Pro implementaci vlastního serveru bude vhodné zvolit stejnou platformu, jakou používá aplikační server HeG – Microsoft .NET Framework. Důvodů je více, tím hlavním pak fakt, že společnost, která provozuje systém HeG, má zákonitě k dispozici serverové operační systémy společnosti Microsoft, které jsou nezbytné pro běh serverových aplikací na platformě .NET Framework. Dalším důvodem je existence knihovny ServiceGateAdapter pro .NET Framework. Tuto knihovnu vyvinula společnost Asseco Solutions pro použití aplikacemi třetích stran, které potřebují komunikovat se systémem HeG. Knihovna obsahuje klienta pro komunikaci s rozhraním webové služby ServiceGate včetně všech potřebných tříd pro obsah požadavků a odpovědí, není tedy třeba implementovat nejnižší vrstvu komunikace směrem k aplikačnímu serveru.

4.3.2 Rozhraní

Komunikace mezi mobilním zařízením a vlastním serverem může probíhat plně v režii vlastního protokolu. Jeho základem by měla být webová služba na transportní vrstvě HTTP(S) z důvodu dobré prostupnosti síťovou infrastrukturou a firewally. Pro implementaci webových služeb se nejčastěji jako základ používají dva protokoly: již zmíněný SOAP (3.6.1) a protokoly na základě architektury REST.

Architektura REST je na rozdíl od procedurálně orientovaného protokolu SOAP orientována na zdroje (data). [5] V případě aplikování architektury na protokol HTTP je každý zdroj unikátně identifikován adresou a pomocí klasických HTTP požadavků (GET, PUT, POST, DELETE) je možné s tímto zdrojem pracovat. Například zavolání GET na adresu *http://helios.eu/mobile/folders/50/* by vrátilo seznam všech záznamů v pořadači číslo 50. Operace POST by pak sloužila pro vložení nového záznamu a podobně pro ostatní operace. Pro výměnu dat se nejčastěji využívá XML nebo JSON. Na základní rozdíly těchto dvou formátů se zaměřím v následujícím oddílu.

4.3.3 Srovnání formátů XML a JSON

Oba formáty mají mnoho společného, jedná o formáty čitelné člověkem a v obou případech je podporováno hierarchické strukturování dat. Zaměřím se nyní na jejich rozdíly

Formát XML je obecným značkovacím jazykem, jde o zjednodušenou podobu staršího značkovacího jazyka SGML. Byl vyvinut a standardizován konsorciem W3C. [4] Formát je široce podporovaný a existuje pro něj velké množství knihoven a nástrojů včetně jazyků pro popis a definici struktury XML dat (DTD, XML Schema). Následující ukázka obsahuje zápis serializovaných údajů osoby – její věk a jméno v XML formátu.

```
<osoba>
   <vek>25</vek>
   <jmeno>Pavel</jmeno>
</osoba>
```

Formát JSON (JavaScript Object Notation) [6] byl navržen jako jednoduchý jazyk pro výměnu dat nezávisle na programovacím jazyku, ačkoliv primárně vychází z JavaScriptu. Tímto jazykem je formát velmi dobře podporován a na rozdíl od XML není potřeba pro jeho zpracování dodatečná knihovna. Pokud bychom chtěli data z předchozí ukázka zapsat ve formát JSON, vypadalo by to následujícím způsobem.

```
{
   "vek" : 25
   "jmeno" : "Pavel"
}
```

Již na první pohled je patrné, že JSON ukázka je oproti té ve formátu XML úspornější. Menší velikost by mohla být přínosem pro zrychlení komunikace s mobilním zařízením. Je důležité podotknout, že vhodným zvolení struktury XML dat je možné snížit podíl popisných dat na rozumnou míru a částečně se tak přiblížit úspornosti formátu JSON. Následuje ukázka XML formátu se stejnými daty jako v předchozích případech ale s využitím úspornější struktury.

```
<osoba vek="25" jmeno="Pavel" />
```

V případě zpracování (deserializace) dat velmi závisí na implementaci syntaktického analyzátoru a nelze vytvářet pouze na základě těchto ukázek jednoznačné závěry.[1])

4.3.4 Volba protokolu

Mým favoritem pro volbu protokolu komunikace mezi vlastním serverem a mobilním klientem se na základě zjištěných skutečností z předchozích odstavců stala architektura REST s využitím formátu JSON pro serializaci dat. U REST architektury oceňuji možnost komunikovat jednoduše pomocí HTTP operací, bez nutnosti obalovat volání do specifických XML struktur a zvyšovat tak objem přenášených dat. Pro JSON pak mluví vysoká datová úspornost.

Konečné rozhodnutí o protokolu bylo ovlivněno zadavatelskou společností. Koncept vlastního serveru byl vyhodnocen jako využitelný i pro paralelně vznikající projekt portálového řešení. S ohledem na kompatibilitu byl vznesen požadavek pro implementaci protokolu SOAP a formátu XML[2]).

Z hlediska implementační platformy toto rozhodnutí nepředstavuje žádný problém, .NET Framework nabízí širokou podporu pro oba zmíněné přístupy i oba formáty. V případě Androidu existují knihovny pro oba formáty a implementace klientského kódu je v obou případech podobně obtížná. Problematice volby knihoven se budu více věnovat v následující kapitole. V obou případech je důležité provést implementaci takovým způsobem, aby bylo snadné v budoucnu komunikační vrstvu nahradit a případně porovnat možnosti jednotlivých přístupů.

[1]) Bohužel se mi nepodařilo nalézt žádnou relevantní studii zabývající se měřením výkonu syntaktických analyzátorů pro rozebírané formáty na platformě Android.

[2]) Pomocí protokolu SOAP je samozřejmě možné přenášet data ve formátu JSON, ale nejedná se podle mého názoru o příliš čisté řešení s ohledem na případnou nutnost dvou syntaktických analyzátorů (XML pro hlavičku, JSON pro obsah) na klientovi.

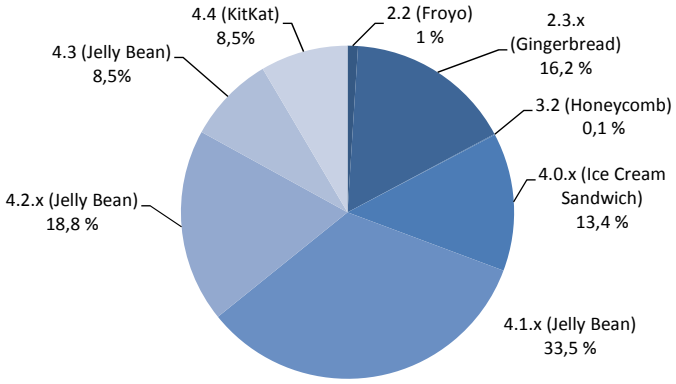
4.4 Android

Závěrem této kapitoly se krátce zaměřím na platformu Android, na které bude mobilní klient implementován. Operační systém Android, založený na Linuxovém jádře, je primárně navržen pro fungování na mobilních zařízení s dotykovou obrazovkou. Poprvé byl projekt oznámen v roce 2007 a první veřejně dostupné zařízení s operačním systémem Android přišlo na trh v říjnu 2008. [7] Projekt operačního systému Android je šířen jako open source pod záštitou uskupení Open Handset Alliance, které se skládá z výrobců mobilních telefonů, telekomunikačních operátorů a jiných firem.

Pro vývoj aplikací jsou poskytovány nástroje umožňující použít syntaxi jazyka Java. Ačkoliv jazyk Java a jeho knihovny jsou volně šiřitelné, virtuální stroj pro Javu nikoliv. Z tohoto důvodu používá android vlastní virtuální stroj Dalvik. Proto musí být aplikace po překladu do byte kódu jazyka Java překonvertována do formátu podporovaného tímto strojem (včetně přelinkování na specifické knihovny). Stejně jako u virtuálního stroje Javy probíhá kompilace android aplikací do instrukcí zařízení během jejich běhu (just–in–time). Tato skutečnost se do budoucna může změnit, pokud se prosadí virtuální stroj ART, který překládá kód aplikace již během instalace (ahead–of–time) a jehož experimentální verze byla jako volitelná součást pro vývojáře představena ve verzi Android 4.4 (KitKat). [7]

4.4.1 Verze systému

Během zhruba šesti let na trhu prošel systém Android značným vývojem a bylo představeno několik klíčových verzí systému. Například verze 3 z poloviny roku 2011 přinesla lepší podporu pro tablety. Tato verze měla existovat souběžně s verzí 2 pro telefony. Avšak již koncem téhož roku byly obě verze sjednoceny pod verzí 4, která přinesla lepší podporu pro implementaci uživatelského rozhraní pro oba typy zařízení. Podporované funkcionality jednotlivých verzí se samozřejmě odrážejí v rozhraní knihoven. Při vývoji aplikací je tedy nutné vymezit minimální podporovanou verzi systému a s ohledem na ní využívat pouze kompatibilní podmnožinu rozhraní knihoven, případně používat knihovny třetích stran, které umožňují nasimulovat funkcionalitu novějších verzí systému na verzích starších.



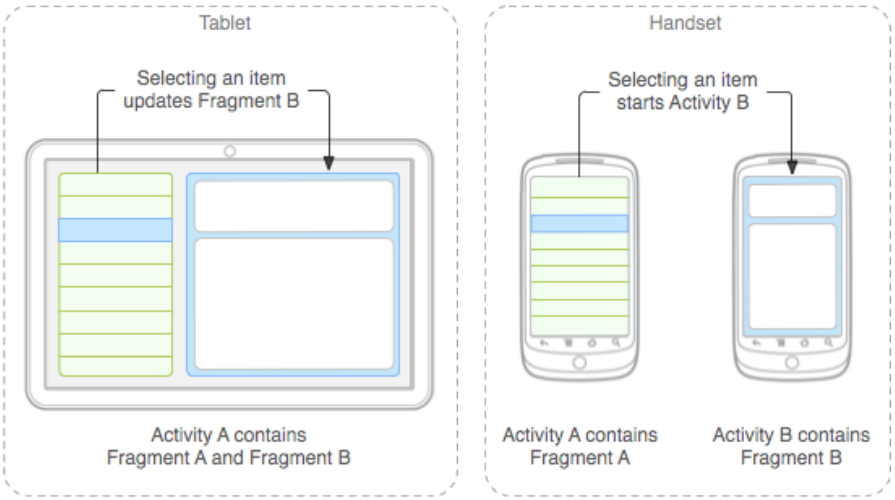
Obrázek 4.3. Podíl verzí systému Android na aktivních zařízeních. (Stav k 1. květnu 2014.)

Obrázek 4.3 ukazuje procentuální zastoupení verzí systému na aktivních zařízení na základě dat z [8]. Verze s podílem menším než 0,1 % nejsou ve statistikách zahrnuty. Na základě těchto dat a s přihlédnutím k požadavku na podporu mobilních zařízení z kategorie telefonů i tabletů jsem se rozhodl, aby mobilní klient podporoval systém Android od verze 3.2. Skupina verzí založených na verzi 2 má podle uvedené statistiky

menší než pětinové zastoupení, proto si myslím, že nemalý čas potřebný k testování a ladění aplikace pro tyto starší verze systému je možné vynaložit v jiné oblasti vývoje mobilního klienta.

4.4.2 Tablet a telefon

Při návrhu uživatelského rozhraní mobilních aplikací, které budou používané na telefonech i na tabletech, je nutné brát ohled na rozdílně fyzické vlastnosti těchto dvou typů zařízení. Vzhledem k výrazně větší ploše obrazovky, kterou tablety nabízí oproti telefonům, je možné také zobrazit více informací. Dále se liší styl ovládání obou zařízení, kdy telefony uživatel ovládá běžně jednou rukou. Je tedy běžnou praxí, že uživatelské rozhraní aplikace a styl ovládání se mírně liší mezi instalacemi na tabletech a instalacemi na telefonech.



Obrázek 4.4. Příklad rozdílného chování aplikace na tabletu a na telefonu. Převzato z [9].

Obrázek 4.4 ukazuje příklad dynamického uživatelského rozhraní. V případě tabletu je na jedné obrazovce [1]) zobrazen zároveň seznam dokumentů v levé části a v pravé části zvolený dokument. U telefonu je chování mírně odlišné, nejprve je zobrazena obrazovka se seznamem dokumentů a následně se po zvolení dokumentu aplikace přepne na druhou obrazovku, na které je zobrazen zvolený dokument, ze kterého lze provést návrat na seznam pomocí systémového tlačítka zpět.

Před příchodem systému Android verze 3 by bylo nutné pro každou obrazovku provést kompletní implementaci. Pro uvedený příklad by to znamenalo vyvinout tři plně funkční obrazovky. Verze 3[2]) přinesla podporu pro takzvané fragmenty [9], ty umožňují implementovat část vzhledu a logiky uživatelského rozhraní a využívat jí ve více obrazovkách. Pro uvedený příklad na obrázku 4.4, který v prvé řadě použití fragmentů ilustruje, by stačilo implementovat dva fragmenty a ty pak vhodně umístit do implementačně nenáročných obrazovek.

Uživatelské rozhraní se nemusí odlišovat jen na základě typu zařízení. Obrazovka zobrazující oba fragmenty najednou by mohla být použita například i pro telefon držený uživatelem v režimu „na šířku“. Proto kromě zmíněné „nové“ podpory pro fragmenty, podporuje systém Android již od svých počátků systém kvalifikátorů pro zdroje aplikace. Mezi obvyklé zdroje aplikace patří různé jazykové verze řetězců (složka *values*),

[1]) Jedna obrazovka se v terminologii systému android nazývá aktivita (activity).
[2]) Spolu s touto verzí vyšla knihovna, která přidává podporu pro fragmenty i na nižších verzích Androidu.

XML definující rozložení aktivit a fragmentů (složka *layout*), obrázky (složka *drawable*) a mnoho dalších. Názvy složek se zdroji lze pak rozšířit o kvalifikátory. Například vyvíjíme aplikaci v angličtině, složka *values* obsahuje soubor s anglickými texty. Pokud budeme chtít aplikaci přeložit do češtiny, stačí vytvořit složku *values-cs* a do ní umístit soubor se stejnou strukturou, ale s českými překlady textů. Aplikace pak automaticky na českých zařízeních hledá textové zdroje nejdříve ve složce *values-cs* a až v případě, že zdroj není nalezen ve složce *values*. Na systémech s jiným nastaveným jazykem pak probíhá hledání přímo ve složce *values*. Analogicky je možné použít kvalifikátory u ostatních zdrojů. Rozdělovat zdroje je dále možné podle orientace displeje (*-land*, *-port*), podle jemnosti rozlišení displeje (*-ldip*, *-mdpi*, *-hdpi*, *-xhdpi* a *-xxhdpi*), verze systému (*-v#*) a mnoha dalších kritérií. Kvalifikátory je možné kombinovat, například obrázkové zdroje pro velmi jemné displeje, české prostředí a zařízení na šířku by se nacházely ve složce *drawable-cs-land-xxhdpi*.

4.4.3 Uživatelské rozhraní

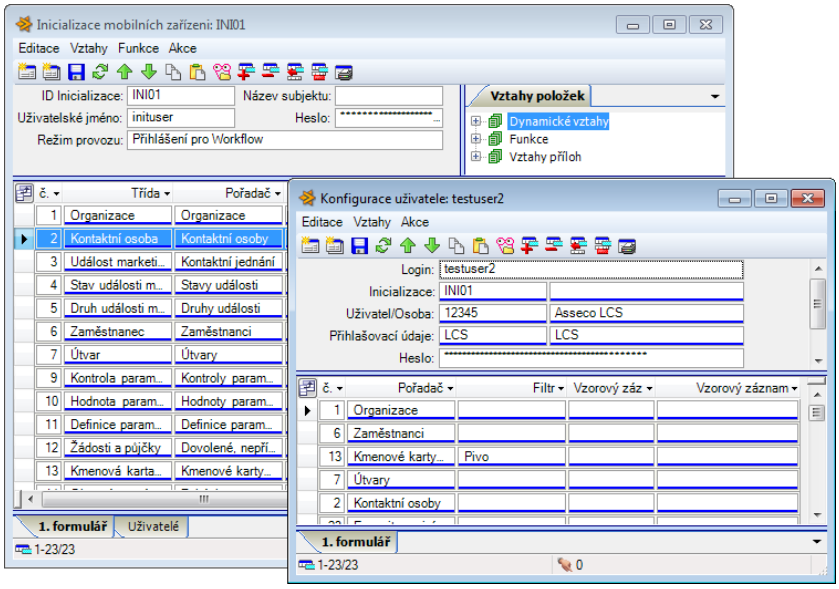
Samotné uživatelské rozhraní vyvíjeného mobilního klienta by mělo vycházet z uživatelského rozhraní Windows klienta HELIOS Green (3.2), na které jsou uživatelé zvyklí. Nabízí se tedy koncept představený v předcházejícím oddíle v rámci příkladu věnujícímu se fragmentům. Hlavním rozdílem je nutnost víceúrovňové navigace, kdy je nejprve nutné zvolit pořadač, následně záznam v něm a až teprve poté se uživatel dostane k „dokumentu“ (záznamu). Dále připadá v úvahu možnost přepínat mezi více záznamy z různých pořadačů a také navigace na záznamy navázané ve vztazích. Samozřejmě musí rozhraní respektovat principy běžně využívané a doporučované pro systém Android. [10]

Kapitola 5
Realizace

Tato kapitola popisuje výsledek implementace řešení na základě analýzy z předcházejících kapitol. Popisuje jednotlivé komponenty návrhu od konfiguračního systému v rámci systému HELIOS Green, přes vlastní komunikační server až po samotného mobilního klienta. Zaměřuje se na implementačně zajímavé části.

5.1 Konfigurace aplikace

V analytické části (4.2.4) této práce jsem zmínil potřebu konfigurovat rozsah dat synchronizovaných do zařízení a další nastavení, týkající se zpřístupnění různých funkcionalit na zařízení. Výsledkem je vytvoření malého konfiguračního modulu pro systém HELIOS Green, který obsahuje několik tříd, jejichž záznamy umožňují konfiguraci agend pro použití na mobilním klientovi na míru každé organizaci. Nyní se zaměřím na dvě nejdůležitější třídy tohoto modulu. Ukázka vyplněných záznamů těchto tříd je na obrázku 5.1.



Obrázek 5.1. Záznamy konfiguračních tříd mobilní aplikace.

5.1.1 Inicializace mobilních zařízení

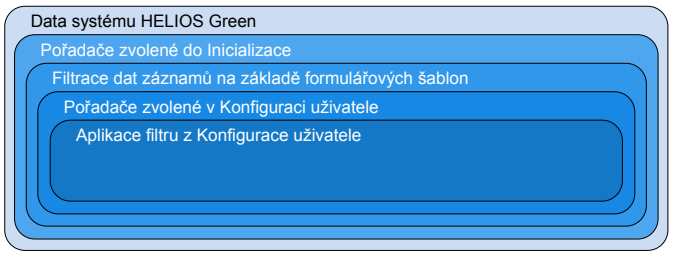
Hlavní třídou modulu je Inicializace mobilních zařízení. Záznam této třídy reprezentuje agendu, která má být z mobilního zařízení vykonávána. Jedná se o dokladovou třídu, hlavička slouží k nastavení identifikace inicializace, přihlašovacích údajů správce inicializace a režimu provozu (5.2.2).

Každá z položek pak reprezentuje jeden pořadač zvolený k synchronizaci a obsahuje atributy a vztahy pro konfiguraci jeho vlastností. Pro každý pořadač je nutné nastavit tzv. systémovou šablonu, která je využívána při synchronizaci (5.4.1), a formulářovou

šablonu (3.3.8) pro hlavičku. V případě pořadačů dokladových tříd je nastavení formulářové šablony položek v konfiguraci volitelné a v případě její absence je třída zobrazována a zpracována bez ohledu na položky. Formulářové šablony neurčují pouze rozložení jednotlivých polí záznamu v uživatelském rozhraní mobilního klienta, ale slouží také jako filtr na atributy a statické vztahy, které chce správce zahrnout do synchronizace. Tento přístup řeší požadavek zmíněný v 4.2.1 intuitivním způsobem bez nutnosti vytvářet speciální konfigurační mechanismus. Ošetřeny jsou i případy, kdy chceme synchronizovat některé z polí skrytě, pouze pro účely lokální aplikační logiky nebo pro zobrazení pouze v přehledové šabloně. V takovém případě stačí nastavit pole ve formulářové šabloně jako skryté. Dynamické vztahy samozřejmě součástí šablony nejsou, jejich výběr pro synchronizaci se řeší pomocí příslušného dynamického vztahu položky inicializace. Stejným způsobem je možné konfigurovat výběr vztahů určených pro práci s přílohami a funkcí spustitelných ze zařízení (včetně těch v rámci workflow). Dále je možné nastavit pořadač pouze pro čtení, zakázat mazání jeho záznamů, skrýt ho ze základní navigace na zařízení[1]) a nebo aktivovat přenos workflow akcí na zařízení.

5.1.2 Konfigurace uživatele

Druhou neméně důležitou třídou je Konfigurace uživatele. Záznamy této třídy se váží na konkrétní záznam Inicializace mobilních zařízení a jasně definují uživatele, kteří mohou na mobilních zařízeních v rámci dané agendy (Inicializace) pracovat. Data hlavičky definují přihlašovací údaje uživatele, kromě možnosti použít klasického přihlašování systému HELIOS Green navázáním příslušných přihlašovacích údajů, je možné zapsat do polí záznamu údaje vlastní. Je tak možné definovat uživatele nezávislého na systému, to může být užitečné, pokud chceme z důvodu bezpečnosti použít pro přihlašování z mobilního zařízení jiné heslo. Vztahy na přihlašovací údaje a konkrétní osobu jsou důležité v případě použití workflow, v takovém případě je při zadání vlastních přihlašovacích údajů nutné použít stejné heslo, v opačném případě by vykonávání workflow akcí končilo chybou.[2])



Obrázek 5.2. Souhrn konfiguračních možností pro vymezení množiny přístupných dat.

Položky záznamů Konfigurace uživatele pak opět tvoří výběr pořadačů pro synchronizaci. Výběr musí být podmnožinou pořadačů definovaných na položkách nadřazeného záznamu Inicializace mobilních zařízení, každý uživatel tedy nemusí mít v rámci jedné agendy k dispozici ty samé pořadače, vše závisí na konfiguračních požadavcích. Kromě samotného vymezení podmnožiny pořadačů je možné pro každý pořadač volitelně nastavit velký flitr (3.3.10) a synchronizovat pro každého uživatele jinou množinu záznamů

[1]) Vhodné pro případy, kdy potřebujeme záznamy pořadače pouze navazovat do vztahů, ale nechceme je běžně prohlížet.

[2]) Tento mechanismus souvisí s režimem provozu (5.2.2) a faktem, že konfigurace ani vlastní server neukládá hesla, ale pouze jejich hashe, může heslo ověřit, ale nemá ho uložené v podobě, ve které by ho mohl použít pro přihlášení.

(například pouze úkoly vlastněné technikem, namísto všech úkolů v pořadači). Obrázek 5.2 shrnuje všechny způsoby jejich vyhodnocování, pomocí kterých je možné data konkrétního uživatele vymezit, včetně jejich posloupnosti. Posledním konfiguračním parametrem na položkách záznamu je takzvaný vzorový záznam. Vzorový záznam reaguje na požadavek (4.1.4) na statické výchozí hodnoty. Pokud správce nastaví vzorový záznam, tento záznam není na zařízení přístupný, ale je použit jako šablona pro každý nově vytvářený záznam v příslušném pořadači. Pro výkaz technika o provedené práci můžeme například na vzorovém záznamu vyplnit všechna pole, která by uživatel vyplňoval vždy stejnými hodnotami a ušetřit mu tak čas strávený vyplňováním.

5.2 Vlastní server

Server, který jsem vytvořil na základě úvah a analýzy v oddílu 4.3 má na starosti předzpracování dat získaných z rozhraní webové služby ServiceGate (3.6). Předzpracování dat respektuje konfiguraci představenou v předcházejícím oddílu. Kromě této konfigurace vyžaduje server pouze drobná nastavení v konfiguračním souboru. Jedná se o přístupové údaje do systému HELIOS Green, kde server vyžaduje uživatele, který má oprávnění spouštět přes rozhraní ServiceGate SQL dotazy. Tyto dotazy server používá pro získání dat z konfiguračních záznamů a také pro získání metadat tříd z dané Inicializace. V některých režimech provozu je tento uživatel používán pro vykonávání téměř všech operací. Dalším parametrem konfigurace serveru je maximální počet souběžných spojení, aby se zabránilo vzniku chyb v důsledku překročení limitů licence organizace. Posledním parametrem konfigurace je umístění složky pro dočasné uložení příloh přenášených na zařízení.

5.2.1 Rozhraní služby

Pro implementaci webové služby využívající protokol SOAP nabízí platforma Microsoft .NET Framework širokou podporu. [11] Pro vytváření webových služeb existují dva základní přístupy. [12] První „top-down“ popisuje postup, kdy nejdříve vzniká definice služby ve formátu WSDL a následně je základní kostra serveru vygenerována z této definice. Tento přístup je obecně považován za složitější, ale výsledkem často bývá čistší návrh rozhraní. Druhým přístupem je „bottom-up“, který funguje opačným směrem. Nejprve vzniká implementace serveru a na základě ní je pak generován soubor definice WSDL. Tento druhý přístup je považován za jednodušší, byť zde existuje možnost zavlečení závislostí na implementační platformě. Platforma .NET Framework využívá přístup „bottom-up“. Pro definici rozhraní stačí upravit konfigurační soubor webové aplikaci a klíčová rozhraní a datové struktury služby doplnit atributy. Definice rozhraní služby doplněná o atributy pak může vypadat například následujícím způsobem:

```
1 [ServiceContract(Namespace = "http://gate.aldor.cz/MobileService")]
2 [XmlSerializerFormat(SupportFaults = true)]
3 public interface IMobileService
4 {
5     [OperationContract]
6     InitSyncConfig GetInitialize(string ID, string userID, string pwd);
7 }
```

Atributy jsou v ukázce umístěny v hranatých závorkách na řádcích 1, 2 a 5. Atribut plní v jazyce C# podobnou funkci jako anotace v jazyce Java, slouží ke značkování tříd, rozhraní, metod a mnoha dalších konstruktů jazyka. S těmito značkami je možné pak

pracovat, ať již jako se vstupem pro generováním na úrovni vývojového prostředí, nebo za běhu při práci s reflexemi. Více je možné se o atributech dočíst v [13].

Operace výsledného rozhraní implementovaného serveru lze rozdělit do dvou pomyslných kategorií. První kategorie je pro operace, do kterých je třeba předávat přihlašovací údaje ze záznamu Inicializace mobilních zařízení (5.1.1), a obsahuje pouze jednu operaci **GetInitialize**. Tato operace slouží ke stažení celé konfigurace agendy do mobilního klienta. V rámci stahování konfigurace server pracuje z daty inicializace, na základě těchto dat (převážně dat formulářových šablon) pak dohledává další podrobné údaje z metadat systému HELIOS Green. K těmto údajům patří definice statických a dynamických vztahů (cílové pořadače), vlastnosti atributů (např. editační styl) a formuláře funkcí. Získané informace neslouží pouze pro účely klienta, ale zároveň jsou využívány serverem při obsluhování požadavků na operace z druhé kategorie.

Druhá kategorie obsahuje operace, které souvisejí se samotným uživatelem (5.1.2), tyto operace slouží primárně pro přenos a práci se záznamy. Pro jejich volání jsou vždy nutné tři parametry: identifikátor inicializace, uživatelské jméno a heslo.

Pro přenos záznamů do zařízení jsou k dispozici následující čtyři operace: **GetData**, **GetDataSince**, **GetDataForFolder** a **GetDataForFolderSince**. Odpověď těchto operací je shodná, vždy obsahuje data záznamů sestavená na základě konfigurace a liší se jen rozsahem odpovídajícím zvolené operaci[1]) a zadaným parametrům. Společným parametrem je přepínač *listOnly*, při jeho nastavení obsahuje odpověď operace pouze výčet čísel záznamů bez jakékoliv hlubší úrovně detailu a pro každý záznam dostupné workflow akce, pokud je workflow v konfiguraci aktivní. Druhým společným parametrem je *partialTag*, ten se používá pro pokračování předešlé dlouho trvající operace, kterou se server rozhodl rozdělit do více částí, v takovém případě je hodnota *partialTag* pro pokračování obsažena v odpovědi. Operace s „Folder“ v názvu požadují v parametrech číslo pořadače a vrací pak záznamy pouze z toho pořadače, na rozdíl od zbývajících operací, které vrací vždy záznamy všech pořadačů z konfigurace uživatele. Klíčové slovo „Since“ v názvu operace pak značí přítomnost parametru pro časové razítko. Časové razítko se používá pro efektivní synchronizaci pouze změněných záznamů a jeho funkce je detailněji popsána v oddílu 5.4.1. V případě parametru operací slouží jako spodní hranice dotazu, v odpovědi jsou tedy pouze záznamy novější (s vyšší časovým razítkem). Server samozřejmě kontroluje hodnoty parametrů proti uživatelské konfiguraci, aby nedocházelo k přístupu k nepovoleným datům.

Opačný směr komunikace – tedy propagaci změn ze zařízení do systému HELIOS Green – řeší dvě operace. První **PutData** slouží pro vytváření a aktualizaci záznamů. Hlavním parametrem je struktura s daty záznamu, která je strukturou shodná se strukturou záznamů v odpovědi operací z předcházejícího odstavce. Požadavek na vytvoření nového záznamu se od aktualizace liší pouze nulovým číslem záznamu. Odpovědí operace je aktuální popř. nově vytvořený záznam. Druhá operace **DeleteData** slouží k odstraňování záznamů. V tomto případě stačí jako parametr číslo záznamu, jeho pořadač a časové razítko pro detekci souběžných změn. Stejně jako v předchozím případě provádí server kontrolu proti konfiguraci, aby nedocházelo k modifikaci pořadačů, které nebyly zvoleny pro synchronizaci, nebo byly nakonfigurovány pouze pro čtení.

Operace pro vykonávání workflow akcí **InvokeWfAction** a operace pro volání funkcí **RunFunction** mají mnoho společného. Hlavním důvodem je fakt, že workflow akce může v rámci vykonání volat funkci. Proto obě operace volitelně přijímají na vstupu parametry volané funkce. Struktura těchto parametrů je shodná s daty (hlavičky) v rámci

[1]) Protokol SOAP (resp. WSDL od verze 1.2) nepodporuje přetěžování operací (oddíl 3.6 v [14]), proto vznikly tyto čtyři varianty jedné operace s odlišným názvem.

struktury záznamu používaných operací. Dalšími parametry jsou opět: číslo záznamu, pořadač a časové razítko. Hlavním a jediným rozdílem v parametrech je, že první operace očekává na vstupu identifikátor workflow akce, zatímco druhá operace pracuje s číslem funkce.

Výčet operací rozhraní serveru uzavírají operace pro práci s přílohami **GetAttachment** a **PutAttachment**. První z operací, jak již název napovídá, slouží pro stažení přílohy. Jejím parametrem je číslo záznamu přílohy (Externí nebo EDM dokument), odpovědí je pouze řetězec, který slouží jako identifikátor pro stažení dat přílohy. Po zavolání této operace server získá zavoláním příslušné funkce data souboru, které uloží do dočasné složky. Z té je možné pak soubor vyzvednout na speciální adrese, skládající se z identifikátoru inicializace a identifikátoru vráceného operací. Pro zajištění bezpečnosti je přístup na adresu podmíněn přihlášením pomocí přihlašovacích údajů uživatele. Tento přístup byl zvolen, aby nebylo nutné směrem na zařízení po převodu do textové reprezentace Base64 kódování, které má svědomí zhruba 30% nárůst velikosti. [15] Druhá z operací slouží analogicky pro nahrání přílohy. Jejími parametry jsou: název souboru, číslo pořadače záznamu, ke kterému bude navázána, číslo dynamického vztahu, do kterého bude navázána a samozřejmě samotný datový obsah souboru. Zde jsem se rozhodl kódování Base64 použít z důvodu snadnějšího začlenění procesu nahrávání do průběhu synchronizace. Podrobněji se práci s přílohami věnuje oddíl 5.7.

5.2.2 Režim provozu

Nastavení režimu provozu z hlavičky záznamu Inicializace mobilních zařízení (5.1.1) udává, které operace vykonává server na rozhraní webové služby ServiceGate jako uživatel z konfigurace serveru a které operace jako příslušný uživatel nastavený v konfiguraci uživatele (5.1.2). Podporovány jsou následující tři režimy provozu:

- **Přihlášení pro Workflow** – veškeré operace probíhají pod účtem uživatele z konfigurace serveru, jedinou výjimkou je operace *InvokeWfAction*. Tento režim je nejefektivnější z hlediska přepínání relací.
- **Přihlášení pro zápis** – operace které nezačínají „Get…“ probíhají pod účtem konkrétního uživatele. Výhodou režimu je skutečnost, že je správně evidován pachatel poslední změny, iniciátor běhu funkce a další obdobné pole v logu systému.
- **Přihlášení pro zápis a čtení** – v tomto režimu se využívá uživatel z konfigurace serveru pouze pro získávání metadat, veškeré operace probíhají pod účtem konkrétního uživatele. Tento režim je pouze pro opodstatněné případy, jako je například nutnost brát v potaz permanentní filtry nastavené pro uživatele.

Pro organizované volání operací služby ServiceGate a pro dodržování limitu maximálního počtu souběžných spojení vznikla na serveru datová struktura *HegAccessPool*. Struktura existuje na serveru pouze v jediné instanci (návrhový vzor singleton v [16]) a slouží jako zásobník pro aktivní relace ServiceGate, jejichž vytváření může být v některých případech časově náročnější než samotné zpracování požadavku. Relace jsou ve struktuře seřazeny podle času posledního použití a jsou automaticky hlídacím vláknem uklizeny po uplynutí určité doby od posledního použití. Jakmile v kódu serveru vznikne požadavek na zavolání operace, je tento požadavek předán struktuře spolu s uživatelem, pod kterým má být operace provedena. Struktura se pokusí najít, zda již existuje aktivní nevyužívaná relace pro daného uživatele, pokud ano, odebere tuto relaci ze seznamu aktivních spojení, vykoná operací a opět relaci vrátí do seznamu na příslušnou pozici. V případě, že v seznamu aktivních spojení nevyhovuje žádná relace, dojde k odhlášení nejdéle nepoužívané relace a přihlášení nové relace pro požadovaného uživatele. Třetí

možnou situací je, že je seznam aktivních spojení prázdný (neexistuje žádné spojení, nebo jsou všechny právě používány pro volání operace). Pokud nebyl vyčerpán počet maximálních spojení serveru, je vytvořena nová relace, v opačném případě dochází k synchronizovanému čekání a operace je vykonána, jakmile dojde k uvolnění některé z právě používané relace zpět do seznamu aktivních spojení. Chování datové struktury je evidováno v logu serveru, správce tak může zátěž serveru zhodnotit a v případě potřeby upravit konfiguraci maximálního počtu spojení (popř. dokoupit další licence) nebo změnit režim provozu serveru pro některé inicializace.

5.2.3 Chybové zprávy

Protokol SOAP podporuje speciální typ zpráv pro případy, kdy vznikne při vykonávání operace chyba. Struktura této zprávy obsahuje kód chyby, který indikuje druhy chyby a dále důvod, který obsahuje lidský čitelný popis chyby. Server může odpovědět některým z následujících chybových kódů:

- **Config-Fault** – indikuje obecnou chybu konfigurace, může se jednat o konfiguraci serveru, ale i neplatnou kombinaci hodnot v konfiguračních záznamech, které neodchytil konfigurační modul v rozhraní systému.
- **HeG-Fault** – tento kód mají chyby, které vznikly při provádění operací na rozhraní webové služby ServiceGate. Detail chyby obsahuje celé chybové hlášení služby ServiceGate a častým zdrojem chyb bývá porušení některého z integritních omezení definovaných v systému HELIOS Green (například nedodržení povoleného rozsahu atributu).
- **Login-Fault** – je reakcí na neplatné přihlašovací údaje správce inicializace nebo uživatele.
- **Param-Fault** – indikuje předání neplatných parametrů při volání operace. Například při pokusu o zápis do pořadačů či polí, které nebyly nastaveny pro synchronizaci.
- **RecordNotFound-Fault** – je speciálním případem předchozího kódu. K chybě s tímto kódem dochází při volání operací nad záznamem, který v systému neexistuje (například byl od poslední synchronizace ze systému odstraněn).
- **UpdateConflict-Fault** – jedná se o nejkomplexnější chybovou zprávu. Dochází k ní při propagaci změn ze zařízení do systému v případě, že byl záznam souběžně modifikován. Server problém detekuje na základě předaného časového razítka záznamu ze zařízení, součástí chybové zprávy je i struktura s aktuální verzí záznamu, aby bylo možné konflikt změn na zařízení vyřešit.
- **Workflow-Fault** –je speciálním případem *HeG-Fault* a vzniká při volání operace *InvokeWfAction*, například v situaci, kdy se změní seznam dostupných akcí bez změny časového razítka záznamu.

Způsob, jakým s chybami pracuje mobilní klient, se zaměřením na řešení chyby s kódem *UpdateConflict-Fault*, je podrobněji popsán v oddílu 5.6.4.

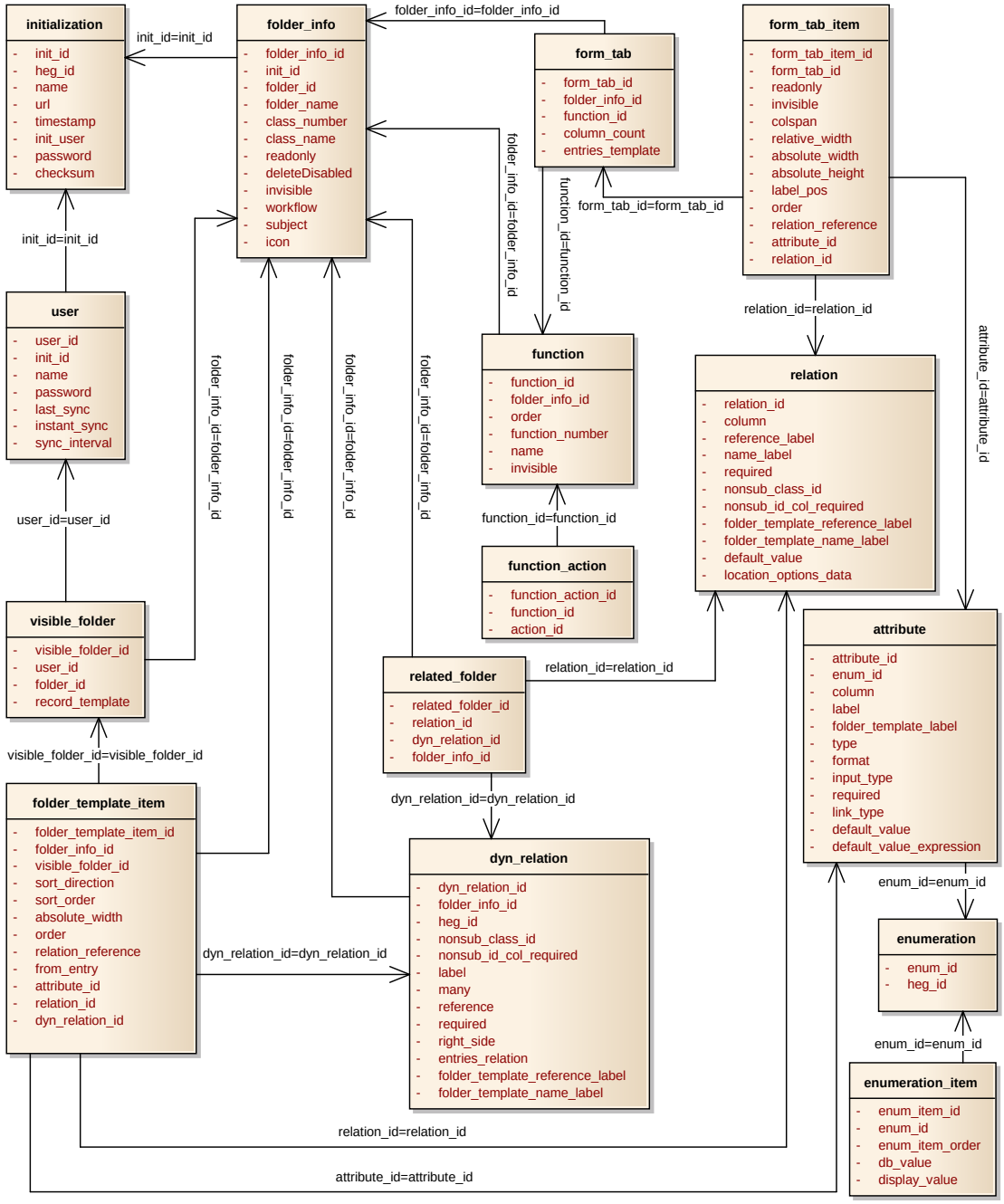
5.3 Datový model

Dříve, než se budu dále věnovat komunikaci mobilního zařízení se serverem, zaměřím se nejprve na způsob, jakým aplikace mobilního klienta ukládá data. Operační systém Android obsahuje již v základu podporu pro relační databázový stroj SQLite. Tento databázový stroj běží v rámci procesu aplikace a každé databázi odpovídá vždy jeden konkrétní soubor dat aplikace. Databáze není tedy uložená na žádném serveru a není nutné provádět její konfiguraci. Implementace databáze splňuje téměř celý standard

SQL-92 [17], výjimkou jsou pouze některé příkazy pro modifikaci struktury tabulek a zápis do databázových pohledů, výčet odchylek od standardu je k dispozici v [18]. Žádná z chybějících částí standardu nepředstavovala omezení při implementaci datového modelu mobilní aplikace.

5.3.1 Data inicializací

Před prvním přihlášením uživatele je nutné mobilního klienta nakonfigurovat na požadovanou inicializaci. Po zadání adresy serveru a přihlašovacích údajů ze záznamu Inicializace mobilních zařízení (5.1.1) klient zavolá operaci *GetInitialize*. Na základě dat odpovědi dojde k naplnění příslušných tabulek datového modelu, jehož diagram je na obrázku 5.3.



Obrázek 5.3. Datový model uložení inicializace na mobilním klientovi.

Hlavní tabulkou je tabulka *initialization*, zde jsou uloženy základní informace přidané inicializace, včetně otisku přihlašovacích údajů, bez jejichž zadání není možné inicializaci z klienta odstranit nebo aktualizovat. Záleží tedy na každém správci, zda dá běžným uživatelům tyto údaje k dispozici, nebo bude inicializaci mobilních zařízení provádět pouze správce. Z návrhu datového modelu logicky vyplývá, že jedna instalace klienta může mít nahrané libovolné množství inicializací.

Každý řádek tabulky *folder_info* pak uchovává informace o pořadačích nastavených pro synchronizaci v rámci příslušné inicializaci. Uložené informace z velké části odpovídají položkám záznamu Inicializace mobilních zařízení. Některé navázané záznamy jsou obsaženy přímo v tabulce (informace o třídě a pořadači), jiné jsou uloženy strukturovaně ve vlastních tabulkách (formulářové šablony a dynamické vztahy).

XML data formulářových šablon (3.3.8) jsou zpracována a následně uložena v tabulkách – *form_tab* a *form_tab_item*, které fungují jako podklad pro vytváření tabulek záznamů a generování formulářů pro práci se záznamy. Každá položka tabu formuláře má pak přiřazený jeden řádek z tabulky *attribute*, pokud se jedná o atribut, nebo z tabulky *relation* pro statické vztahy. U vztahů je dále důležitá tabulka *related_folder*, která definuje množinu pořadačů, ze kterých je možné do vztahu navazovat záznamy. Pro atributy jsou zde navíc dvě tabulky: *enumeration* a *enumeration_item*. Tyto tabulky mohou volitelně obsahovat omezení hodnoty atributu výčtem tak, jak to je popsané v oddílu 3.3.4.

Tabulky formulářů jsou také využívány pro uložení vzhledu parametrických oken funkcí. Základní informace o funkcích se nacházejí v tabulce *funciton*, která zároveň přiřazuje funkci k pořadači. Tabulka *funciton_action* volitelně přiřazuje funkci k workflow akcím, aby mohlo být v případě vykonávání příslušné akce zobrazeno parametrické okno funkce.

Kromě formulářů a funkcí se k pořadači váží dynamické vztahy z tabulky *dyn_relation*. Dynamické vztahy, stejně jako vztahy statické, používají tabulku *related_folder* pro vymezení množiny pořadačů určených k výběru záznamů do vztahu.

Atributy, statické vztahy a dynamické vztahy mohou být kromě polí formuláře (*form_tab_item*) odkazovány ze sloupečku přehledové šablony (3.3.9) v tabulce *folder_template_item*. Ta kromě základních hodnot, jako je popisek a jiné vizuální vlastnosti, definuje výchozí řazení včetně pokročilého (např. řazení kontaktů nejdříve podle příjmení a následně podle jména). Řádky tabulky pak odkazují v případě role výchozí přehledové šablony na pořadač. V případě přehledových šablon, které mohly být upraveny uživatelem (při prvním přihlášení vznikají jako kopie výchozí přehledové šablony), odkazují na uživatele.

Přehled tabulek uzavírá tabulka uživatelů – *user*. Záznam v této tabulce vzniká při prvním přihlášení uživatele. Aby bylo možné provést přihlášení i bez připojení k síti, obsahuje záznam otisk hesla pro kontrolu. Dále tabulka obsahuje základní nastavení spouštění synchronizace, které může uživatel po přihlášení změnit. Tabulka *visible_folder* tvoří výběr podmnožiny pořadačů inicializace dostupných uživateli, jedná se tedy o ekvivalent položek záznamu Konfigurace uživatele (5.1.2), včetně identifikátoru vzorového záznamu.

5.3.2 Objektově relační mapování

Protože datový model inicializace obsahuje několik tabulek s nezanedbatelným počtem sloupečků, hledal jsem tedy způsob jakým s databází efektivně (z hlediska programátorského úsilí) pracovat ve zdrojovém kódu. Z „dospělejších“ platforem a vývojových prostředí jsem byl zvyklý na možnost využití objektově relačního mapování (ORM), které

umožňuje jednoduchým způsobem namapovat pole třídy (tzv. entity) na sloupečky tabulky. Následně se o veškeré načítání, vytváření a ukládání entit stará framework a zvyšuje se tak přehlednost kódu a také snižuje počet příležitostí pro vznik chyb. Prozkoumal jsem tedy situaci na platformě Android a nalezl jsem dvě často používané ORM knihovny: GreenDAO a ORMLite.

GreenDAO funguje na principu generátoru kódu. Nejprve je nutné napsat malou aplikaci, která následující způsobem definuje strukturu entit:

```
Schema schema = new Schema(1, "cz.aldor.heliosmobile.db");

Entity folderEntity = schema.addEntity("folder_info");
folderEntity.addIdProperty();
folderEntity.addIntProperty("folder_id");
folderEntity.addStringProperty("folder_name");
…
new DaoGenerator().generateAll(schema, "../../src-gen");
```

Po spuštění aplikace vygeneruje do zvoleného adresáře potřebné třídy pro entity a přístupové objekty.

ORMLite oproti tomu používá pro mapování polí na proměnné třídy anotace. Je tedy možné využít již existující třídu pouhým doplněním potřebných anotací. Třída se stejnou strukturou jako v předchozí ukázce by vznikla za použití anotací knihovny ORMLite následujícím způsobem:

```
@DatabaseTable(tableName = "folder_info", daoClass = FolderInfoDao.class)
public class FolderInfoEntity {
   @DatabaseField(generatedId = true)
   private int folder_info_id;
   @DatabaseField
   private int folder_id;
   @DatabaseField
   private String folder_name;
   …
```

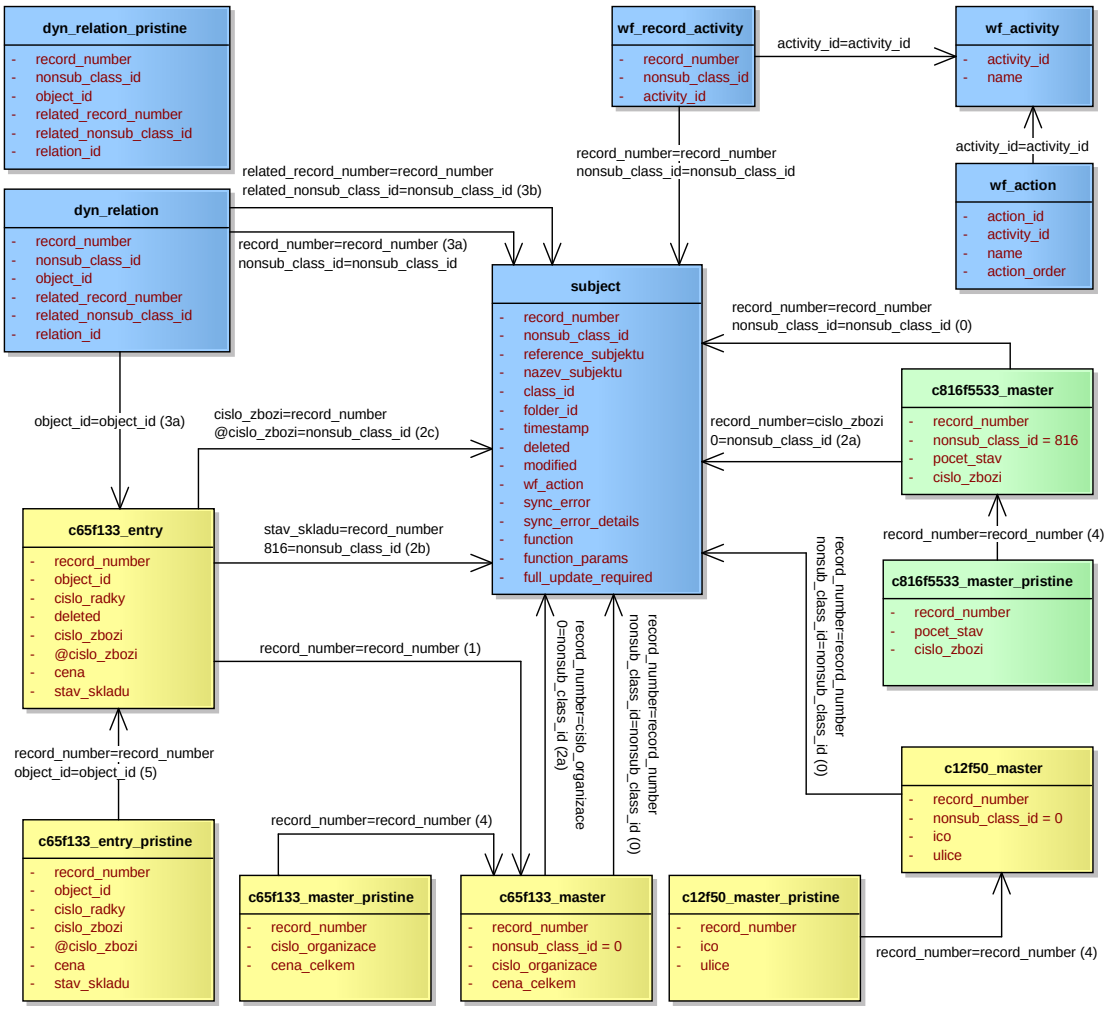
Na základě anotací pak knihovna vygeneruje strukturu tabulek a následně je možné použít dynamicky vytvářené přístupové objekty. Z pohledu podpory cizích klíčů (vztahy mezi tabulkami) a dalších klíčových vlastností pro můj datový model jsou možnosti obou uvedených knihoven srovnatelné.

Konečné rozhodnutí bylo ve prospěch knihovny ORMLite[1]). Hlavním důvodem byla čitelnější definice struktury entity a také možnost provádět ve struktuře rychlé změny bez nutnosti generovat třídu entity znovu. Změny struktury entit byly hlavně z počátku vývoje častým jevem. Za drobný nedostatek, který by se potenciálně mohl projevit dopadem na výkon, jsem považoval samotné použití anotací, které jsou za běhu vyhodnocovány pomocí reflexe. Z hlediska výkonu se použití reflexí obecně považuje za drahé. [19] Knihovna ORMLite ale nabízí pro platformu Android možnost vygenerovat na základě anotací ve třídách entit konfigurační skript. Tento skript je pak při spuštění jednorázově načten a na jeho základě jsou vytvořeny objekty pro přístup, dochází tak k eliminaci použití reflexe. [20] Při experimentálním nasazení tohoto konfiguračního skriptu nedošlo na testovacích zařízení k výraznému zrychlení, proto jsem rozhodl konfigurační skript nepoužívat.

[1]) Webové stránky knihovny na `http://ormlite.com/`.

5.3.3 Data záznamů

Zatím jsem představil pouze způsob, jakým mobilní klient ukládá informace získané z inicializace a jakým způsobem jsou ukládány základní informace o uživatelích. Pro data samotných záznamů se ukázalo jako praktičtější vytvoření soukromé databáze pro každého uživatele zvlášť. Hlavním důvodem je skutečnost, že je podporována souběžná existence více inicializací na jednom zařízení. Z popisu konfigurace (5.1 také víme, že každý uživatel může mít k dispozici jiné záznamy. Dále nepovažuji za žádoucí, aby v případě více uživatelů jedné inicializace docházelo ke sdílení rozpracovaných změn na záznamech před spuštěním synchronizace. Vyčlenění dat uživatele do soukromé databáze je také výhodné z hlediska velikosti otevřených databází, vždy je otevřená pouze databáze s inicializací a databáze se záznamy právě přihlášeného uživatele. Datový model této soukromé databáze je naznačen na obrázku 5.4.



Obrázek 5.4. Datový model uložení záznamů na mobilním klientovi.

Pro lepší názornost jsem pro diagram použil stejné třídy a barevné označení jako v analytické části týkající se datového modelu systému HELIOS Green (3.4). Červené tabulky zde nejsou pochopitelně zastoupeny, protože veškerá metadata jsou uložena v databázi s daty inicializace (5.3.1). Na metadata inicializace je však nepřímo odkazováno přes identifikátory použité v hlavní databázi například sloupce *relation_id* a *folder_id* v tabulkách s modrou barvou.

Protože je konfigurace mobilní aplikace orientovaná na pořadače[1]) jsou i tabulky se specifickými daty záznamu vytvářeny pro každý pořadač zvlášť a pojmenovány unikátním názvem: *c[číslo třídy]f[číslo pořadače]_* v případě dat hlavičky je doplněna koncovka *master*, pro položky pak *entry*. Tabulky začínající *c65f133_* jsou tabulky záznamů faktur vydaných, *c12f50_* tabulky organizací a *c816f5533_* tabulky nonsubjektových rozpadů stavů skladu.

Model principiálně vychází z datového modelu systému, opět je zde společná tabulka *subject*, kde jsou kromě společných vlastností uloženy operace čekající na synchronizaci (např. sloupec *deleted* indikuje požadavek na smazání, sloupec *wf_action* obsahuje zvolenou workflow akci atd.).

K zásadní změně ale došlo v přístupu k subjektovým a nonsubjektovým třídám. Z důvodu snadnější synchronizace a zefektivnění práce editorů vztahů bylo nutné, aby i nonsubjektové třídy měly záznam v tabulce *subject*. V analýze nonsubjektových tříd (3.4.2) jsem uvedl, že pro nonsubjekty stačí unikátnost identifikátoru pouze v rámci jejich tabulky, hrozí tedy kolize identifikátoru s jinými třídami. Důsledkem je zavedení druhé komponenty identifikátoru záznamu - *nonsub_class_id*. První část identifikátoru *record_number* je shodné s identifikátorem v rámci systému HELIOS Green, v případě subjektových tříd se jedná o *cislo_subjektu* pro nonsubjektové třídy je to *cislo_nonsubjektu*. Druhá část *nonsub_class_id* je pro subjektové třídy vždy rovna nule, pro nonsubjektové třídy obsahuje číslo třídy. Toto je na obrázku 5.4 znázorněno výchozí hodnotou sloupce *nonsub_class_id* u tabulek se žlutou barvou (subjektové třídy) a se zelenou (nonsubjektové třídy) barvou. Všechny hlavičkové tabulky odkazují (na obrázku s číslem 0) pomocí cizího klíče na tabulku *subject*. Změna identifikátoru záznamu také ovlivnila práci se vztahy.

Statické vztahy jsou ukládány jako sloupec s identifikátorem navázaného záznamu v tabulkách dat záznamů. Ze změn popsaných v předchozím odstavci logicky vyplývá potřeba dvou sloupců pro statický vztah. V rámci úspory ukládaných dat jsem se rozhodl vytvářet sloupec pro *nonsub_class_id* navázaného záznamu pouze v případě, pokud nelze tuto druhou část identifikátoru pro daný vztah jednoznačně určit na základě jeho metadat. Tedy je do takového vztahu možné navázat pouze subjektové třídy, nebo pouze záznamy jedné nonsubjektové třídy (více viz 3.4.3). Vztahy 2a a 2b jsou příkladem vztahů, kde je z metadat statického vztahu zřejmé *nonsub_class_id*, spojení tabulek se pak pro *nonsub_class_id* provádí s podmínkou na konstantní hodnotu, kde vztahy 2a jsou vymezeny na záznamy subjektových tříd a vztah 2b je vymezen na záznamy nonsubjektové třídy číslo 816 (Rozpad stavu skladu). Vztah 2c naopak může ukazovat na záznam subjektové i nonsubjektové třídy zároveň, proto je nutné použít kromě základního sloupce vztahu *cislo_zbozi* také sloupec *@cislo_zbozi* pro uložení *nonsub_class_id* navázaného záznamu.

U dynamických vztahů je strategie ukládání jednodušší. Oproti systému uložení dynamických vztahů v systému HELIOS Green (poslední odstavec oddílu 3.4.3) jsem se z důvodů zjednodušení práce s modelem rozhodl sloučit tabulky dynamických vztahů do jedné tabulky *dyn_relation*. Tato tabulka strukturou vychází z tabulky *lcs.objektsubjekt* na obrázku 3.8. Sloupec *object_id*, který je součástí levé strany vztahu a identifikuje, které položky záznamu dokladové třídy se vztah týká, umožňuje nastavení nulové (0) hodnoty pro případy, kdy je na levé straně vztahu přímo hlavička záznamu. Hodnota *nonsub_class_id* se ukládá pro levou i pravou stranu vztahu (3a a 3b) a nemá zde tedy smysl pokoušet se o úsporné chování.

[1]) Dva pořadače stejné třídy mohou mít rozdílné formulářové šablony a tedy i jiná pole nastavená pro synchronizaci.

Při prvním pohledu na diagram na obrázku 5.4 je patrné, že veškeré tabulky s konkrétními daty záznamů jsou zdvojené a pro každou *master* či *entry* tabulku existuje tabulka s koncovkou *_pristine*. Účelem těchto tabulek je evidence změn provedených uživatelem na záznamech prostřednictvím uchovávání originálních hodnot. Vždy, když uživatel změní data na hlavičce nebo na položce záznamu, vytvoří se během ukládání změn kopie ukládaného řádku ve zmíněné tabulce. Jakmile dojde k synchronizaci, nebo uživatel uvede data do výchozí situace, je řádek z *..._pristine* tabulky odstraněn. Jinými slovy: řádek v *..._pristine* tabulce existuje pouze v situaci, kdy původní řádek obsahuje nějaké změny. Motivací pro tento přístup je fakt, že na základě porovnání originálního a aktuálního řádku jsou sestavovány seznamy změn odesílané při synchronizaci. Dochází tak k minimalizaci přenášených dat a dále je možné na zařízení v případě chyb odvolat změny a uvést tak záznam do výchozího stavu.

Pro dynamické vztahy existuje také tabulka původních hodnot. Tato tabulka má název *dyn_relation_pristine*. Oproti *..._pristine* tabulkám pro data záznamům se zde ukládá kopie řádků již v momentě synchronizace. Důvodem je, že na rozdíl od řádků v ostatních datových tabulek, není možný řádek tabulky s dynamickými vztahy identifikovat jinak, než prostřednictvím kombinace všech šesti ukládaných hodnot. Proto se při synchronizaci porovnávají data tabulek na úrovni jednotlivých vztahů pomocí množinových operací mezi daty z jedné a druhé tabulky.

Na obrázku 5.4 nejsou zahrnuty pro zlepšení přehlednosti vztahy u *..._pristine* tabulek. Veškeré vztahové spojnice (2a, 2b, 2c, 3a a 3b) by správně měly vést i od příslušných tabulek původních hodnot.

Poslední skupinou tabulek, které zbývá představit jsou tabulky pro uložení workflow akcí. Samotné akce (*wf_action*) jsou uloženy v rámci aktivit (*wf_activity*). Aktivity jsou pak následně pomocí vztahové tabulky *wf_record_activity* mapovány na samotné záznamy z tabulky *subject*.

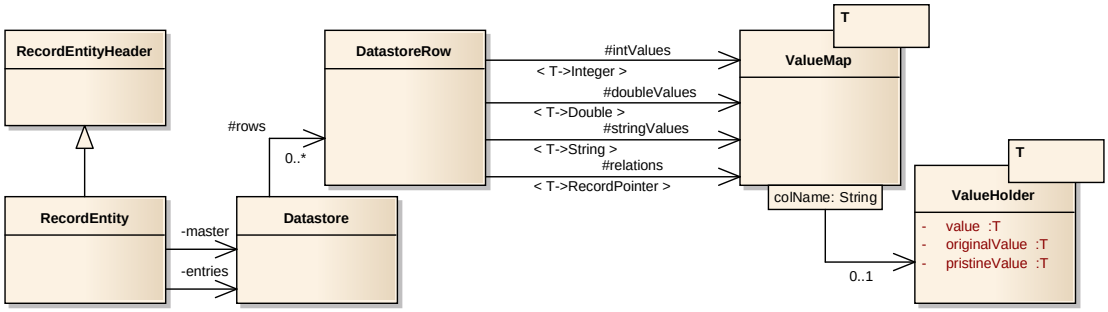
Referenční integrita je z velké části (zvláště pro vztahy) hlídána až na úrovni aplikační logiky klienta. Výjimkou jsou tabulky dat záznamů a workflow akcí, kde jsou nastaveny omezení na cizí klíče po celé cestě od *..._pristine* tabulky až po tabulku *subject* (vazby: 4 a 0), včetně tabulek pro položky (vazby: 5 a 1). Cizí klíče mají nastavené háčky na události mazání (DELETE) a aktualizace (UPDATE). Například při smazání záznamu z tabulky *subject* dojde k propagaci mazání kaskádou do všech tabulek, ve kterých byl záznam uložen, podobným způsobem se propaguje změna identifikátoru záznamu.

5.3.4 Entity

Principy popsané v předcházejícím oddílu nejsou jednoduše aplikovatelné na knihovnu pro objektově relační mapování (5.3.2). Hlavním důvodem je fakt, že struktura tabulek záznamů je dynamicky generována na základě dat inicializace a že jeden záznam je složen z dat několika tabulek najednou (například: *subject*, *c12f50_master* a *c12f50_master_pristine*). Jako reakci na zmíněné požadavky jsem implementoval datový typ *RecodrEntity*, který obstarává celý životní cyklus záznamu v aplikační logice mobilního klienta. Zjednodušený diagram tříd tvořících entitu je na obrázku 5.5.

Základní třídou entity je *RecodrEntityHeader*, ta má na starosti konzistenci údajů v tabulce *subject*, jako je označení pro smazání, zápis požadavků na volání funkce či workflow akce. Tuto třídu pak rozšiřuje už zmíněná třída *RecodrEntity*, prostřednictvím které je již možné plnohodnotně manipulovat se specifickými daty záznamu. Data hlavičky (*master*) a popř. položek *entries* jsou uloženy ve třídě *Datastore*. *Datastore* se pak dělí dále na řádky, ty už korespondují s opravdovými řádky tabulek, včetně

tabulky originálních hodnot (např. *c12f50_master* a *c12f50_master_pristine*). *Datastore* pro hlavičku má samozřejmě pouze jediný řádek.



Obrázek 5.5. Diagram tříd entity záznamu.

Třída reprezentující takový řádek je *DatastoreRow*. Obsahuje čtyři instance mapy hodnot sloupců (*ValueMap*), každou zaměřenou na hodnoty jiného datového typu. Zatímco *DatastoreRow* má na starosti základní databázové operace s řádkem, samotné hodnoty jsou uchovávány v malé datové struktuře *ValueHolder*, ke které se v mapě přistupuje názvem sloupce. Ve struktuře jsou tři verze hodnoty:

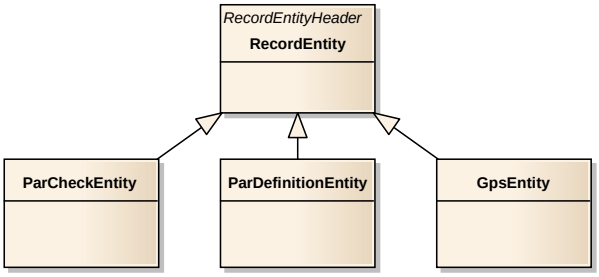
- **value** – aktuální hodnota z uživatelského rozhraní (zatím pouze v paměti zařízení).
- **originalValue** – hodnota v hlavní tabulce *Datastore* (např. v *c12f50_master*).
- **pristineValue** – původní hodnota získaná ze synchronizace, tedy v *…_pristine* tabulce (např. v *c12f50_master_pristine*).

Kromě základní správy těchto tří hodnot poskytuje *ValueHolder* prostředky pro monitorování změn hodnot, na které některé prvky uživatelského rozhraní potřebují reagovat.

Uvedený diagram nezahrnuje práci s dynamickými vztahy, která není implementačně příliš zajímavá. Úložiště dynamických vztahů (*DynRelationsStore*) se nachází v řádku *DatastoreRow* a podobně jako k hodnotám se k dynamickým vztahům přistupuje pomocí abstraktního datového typu mapa za použití čísla vztahu. Následně *DynRelationHolder* obsahuje tři množiny (*HashSet*) navázaných záznamů odpovídající zmíněným třem typům hodnot ve struktuře *ValueHolder*.

V souvislosti s ukládáním struktury *RecodrEntity* se chci ještě zaměřit na sloupec *modified* tabulky *subject*. Tento sloupec obsahuje informaci, zda se záznam liší od záznamu získaného během synchronizace. Využíván je během synchronizace změn do systému HELIOS Green, na základě jeho hodnoty jsou sbírány záznamy, jejichž změny je nutné zapsat na server. Za správnou hodnotu odpovídá právě entita, která má během zápisu do lokální databáze k dispozici veškeré informace o datech záznamu. V případě neexistence tohoto údaje by sběr záznamu musel být prováděn prostřednictvím komplexního SQL dotazu, který by procházel tabulky specifických dat pro jednotlivé pořadače a porovnával je proti *…_pristine* tabulkám. V průběhu synchronizace se také entita podílí na komunikaci se serverem, na základě načtených dat vytváří struktury jako parametry pro rozhraní serveru.

Entita je také jedním z míst implementace specifické aplikační logiky. Vedle samotných dat umožňuje poskytovat informace pro prezentační vrstvu mobilního klienta a ovlivňovat tak chování jednotlivých formulářových prvků (například omezení zápisu nebo výběru hodnot pro vztahy). Pro splnění požadavků ze zadání bylo nutné implementovat pro některé třídy specializované entity, jejich výčet je na obrázku 5.6.



Obrázek 5.6. Entity specializované na konrétní třídy.

Třída *ParCheckEntity* obsahuje implementaci podpory pro dotazníky zmiňované v oddílu 4.1.1, z důvodu čistšího přístupu ke klíčovým atributům vznikla entita i pro třídu definující položku dotazníku – *ParDefinitionEntity*. Pro efektivnější práci se záznamy GPS polohy (více viz 5.8) vznikla třída *GpsEntity*. Vytvoření další specializované entity je snadné, stačí, aby třída dědila z *RecodrEntity*, pak jen zbývá zaregistrovat typ entity pro konkrétní číslo třídy ve správci záznamů (třída *RecordManager*) a aplikace začne entitu používat.

5.3.5 Datové typy

Datové typy využívané aplikací mobilního klienta je možné snadno odvodit na základě obsahu předcházejících odstavců. Použitý databázový stroj SQLite podporuje čtyři datové typy: INTEGER pro celá čísla, REAL pro desetinná čísla, TEXT pro řetězce a BLOB pro surová data. Typový systém je doplněn o prázdnou hodnotu NULL. [21] Pro potřeby mé aplikace jsou dostačující první tři typy, které odpovídají následujícím třídám jazyka Java: Long, Double a String.[1]) Samozřejmě je možné použit datové typy s nižší dostačující přesností, například Integer.

Vyjmenované typy pokrývají většinu typů z analýzy (3.4.4). Výjimkou je datový typ pro datum a čas. Tento datový typ je v databázi uchováván v textové podobě ve formátu „YYYY-MM-DD-hh-mm-ss“, datum 29. dubna 2014 a čas 16:45 je v databázi uloženo jako „2014-04-29-16-45-00“. Pořadí jednotlivých komponent v rámci textové reprezentace zaručuje podporu pro správné chronologické řazení a porovnávání již na úrovni databáze.

Drobný problém může představovat přesnost 8 bytů pro desetinné číslo typu REAL v kontrastu s datovými typy použitými v „dospělé“ databázi na serveru. Tento poznatek ale není v rozporu s žádnou z agend požadovaných v zadání práce. V případě potřeby je v budoucnosti možné zavést přesnější desetinný datový typ využívající pro ukládání do databáze zařízení textovou reprezentaci.

5.3.6 Zástupné záznamy

Navzdory vymezení inicializace pouze na konkrétní pořadače a pole záznamů (5.1) je v některých případech žádoucí synchronizovat vztah na záznamu, aniž by docházelo k synchronizaci celých navázaných záznamů. Typickým příkladem je vztah na zemi na třídě „Organizace“ či „Kontaktní osoba“. Tyto vztahy ukazují na záznamy třídy „Země“, jejich obsahem jsou různé detailní informaci o zemi, jako jsou kódové označení podle norem ISO, telefonní předvolba či příznak, zda se jedná o členskou zemi EU. Pro zmíněné vztahy je ale plně dostačující znát název, případně zkratku názvu země, které tvoří referenci a název záznamů, tedy základní informace zobrazované editory statických a dynamických vztahů.

[1]) Primitivní datové typy (long a double) nebyly použity kvůli chybějící podpoře pro *null* hodnoty.

Pro tyto záznamy „na hranicích“ synchronizované oblasti jsem použil ukládání ve formě tzv. zástupných záznamů. Při přenosu záznamu na zařízení obsahuje datová struktura pro navázané záznamy kromě jejich identifikátoru také název a referenci. Pokud klient zjistí, že odkazovaný záznam na zařízení neexistuje, vytvoří pro něj záznam pouze v tabulce *subject*. Přidaný řádek tabulky obsahuje identifikátor, název a referenci. Pro indikaci, že se jedná o zástupný záznam, je nastaveno číslo pořadače (*folder_id*) na hodnotu -1. Zástupné záznamy nejsou v klientské aplikaci na žádném místě zobrazovány, není je možné navázat do vztahu (chybí informace o pořadači), slouží pouze jako zdroj názvu a reference pro zmíněné editory.

5.4 Synchronizace

Z hlediska práce s daty záznamů jsem představil klíčové součásti na obou stranách. Rozebral jsem rozhraní služby na straně vlastního serveru (5.2.1) a stejně tak způsob, jakým aplikace mobilního klienta ukládá data záznamů (5.3.3). Zbývá se tedy zaměřit na principy komunikace mezi serverem a klientem. Zaměřím se na každý směr komunikace zvlášť. Nejprve ale vysvětlím způsob, jakým dochází k identifikaci změněných záznamů.

5.4.1 Časové razítko

Již v popisu rozhraní služby (5.2.1) jsem naznačil, jakým způsobem dochází k získávání pouze těch záznamů, které byly od poslední synchronizace změněny. Klíčem k zavedení podpory pro tyto operace bylo určení vlastnosti (atributu) záznamu, na základě které by bylo možné snadno a efektivně rozhodnout, zda došlo ke změně záznamu. Prvním logickým kandidátem byl atribut „Datum poslední aktualizace“, který se v systému nachází ve společné tabulce *lcs.subjekty* (viz 3.4) a je přístupný také v uživatelském rozhraní standardního klienta. Tento atribut se ale ukázal jako nevyhovující minimálně ze dvou důvodů. Prvním je malé rozlišení jeho hodnoty, jedná se o klasický datový typ datum a čas s přesností pouze na sekundy, což mělo za následek zahrnutí do odpovědi většího množství záznamů, než bylo nutné. Druhým a také důležitějším problémem je přítomnost tohoto atributu pouze na subjektových třídách. Jelikož je hodnota atributu aktualizována systémem, není možné obecným způsobem přenést chování do nonsubjektových tříd. Jako mnohem vhodnější se ukázal sloupeček *timestamp*, který je také přítomný v tabulce *lcs.subjekty* a je systémem interně využíván při replikaci databáze. Hlavní výhodou tohoto sloupce je jeho datový typ – *ROWVERSION* [22], jedná se o 8 bytů dat, které je možné interpretovat jako nezáporné celé číslo. Pokud je sloupec s tímto typem přítomen v tabulce, databázový stroj při každé modifikaci nebo vytvoření řádky do něj zapíše unikátní (pro celou databázi) hodnotu zvyšujícího se čítače. Aktualizace hodnoty je tedy nezávislá na aplikační logice a sloupec (spolu s atributem) je tak možné přidat i do tabulek nonsubjektových tříd. Prostým porovnáním hodnoty *timestamp* lze zjistit, zda se záznam změnil, či ne.

K časovému razítku se také váže systémová šablona, která byla zmíněna jako povinný položkový vztah záznamu Inicializace mobilních zařízení (5.1.1). Systémová šablona je klíčová pro výkon vlastního serveru. Je tvořena jedním sloupcem, který obsahuje databázový výraz pro získání časového razítka. Šablonu server využívá při všech *Get…* operacích. Důvodem je výrazně rychlejší vyhodnocení ServiceGate operace *BROWSE* oproti operaci *RETRIEVE*, kde v prvním případě nemá slovo aplikační logika, která se v druhém případě používá pro každý záznam. Dalším důvodem je, že pro *Get…*operace s nastaveným *listOnly* parametrem je výsledek *BROWSE* požadavku plně dostačující.

V neposlední řadě lze pro tento požadavek nastavit rozsah odpovědi, lze tak lépe kontrolovat dobu zpracování požadavků a zabránit tak zablokování vlastního serveru čekáním na odpověď služby ServiceGate, která obsahuje velké množství záznamů.

5.4.2 Přenos změn do zařízení

Ačkoliv se synchronizace dat směrem do zařízení provádí až po nahrání změn na server, popíšu nejdříve její principy. Jedním z důvodů je fakt, že na čerstvě inicializovaném mobilním klientovi proběhne tato část jako první. Postup při přenosu nových a změněných záznamů do zařízení je možné shrnout následujících kroků:

1. Aktualizace pro uživatele nastavených pořadačů.
2. Detekce odstraněných záznamů.
3. Aktualizace jednotlivých pořadačů.
4. Aktualizace workflow akcí.
5. Aktualizace uživatelského rozhraní.

První krok zavolá operaci *GetDataSince* s časovým razítkem rovným *null*, výsledkem je odpověď, která je z hlediska dat záznamů prázdná, ale obsahuje základní strukturu s pořadači. Na základě této struktury má aplikace přehled o aktuální nastavení záznamu Konfigurace uživatele (5.1.2) a podle něho upraví strukturu soukromé databáze uživatele na zařízení. Pro nově přidané pořadače vytvoří tabulky specifických dat a pro odstraněné pořadače tyto tabulky odstraní. Samotnému odstranění tabulek předchází převod všech relevantních řádků v tabulce *subject* na zástupné záznamy, aby nedošlo k porušení konzistence již existujících záznamů v jiných pořadačích. U ostatních pořadačů dochází k aktualizaci identifikátoru výchozího záznamu. Veškeré změny záznamu konfigurace uživatele je tedy mobilní klient schopný reflektovat bez nutnosti opětovného nahrání inicializace.

Druhý krok probíhá pouze, pokud se nejedná o první synchronizaci uživatele. Systém HELIOS Green umožňuje základní monitorování mazání záznamů (třída „Smazané záznamy“). Problémem této funkcionality je, že nebývá běžně na zákaznických instalacích pro všechny třídy aktivní, důvodem je skutečnost, že hlavním účelem funkcionality je poskytnutí možnosti obnovit smazaný záznam, při každém mazání záznamu třídy s aktivovanou funkcionalitou dochází k vytváření záznamu třídy „Smazané záznamy“, který obsahuje kompletní data původního záznamu. Dále je struktura těchto záznamů nevhodná pro mé použití, konfigurace probíhá na úrovni tříd, zatímco má aplikace je orientována na pořadače. Informaci o pořadači je velmi těžké pro smazané záznamy získat, nenalezl jsem žádný efektivní způsob, jak od systému získat seznam všech záznamů smazaných v daném pořadači od poslední synchronizace. Rozhodl jsem se zvolit přístup, který kontroluje všechny záznamy na zařízení proti všem záznamům v rámci konfigurace daného uživatele. Tento přístup se může na první pohled zdát jako velmi neefektivní a dlouhotrvající, ale provedená měření (6.3) prokázala, že se jedná o přístup v praxi použitelný.

Nejprve je zavolána operace *GetData* s parametrem *listOnly*, následně proběhne agregace všech záznamů odpovědi (sbírá se identifikátor, pořadač a časové razítko) a výsledek se seřadí podle identifikátoru. Tento seznam záznamů je postupně procházen a porovnáván se souběžně procházeným výsledkem dotazu nad tabulkou *subject*, který je identicky seřazen a jsou vybrány stejné hodnoty. Tímto způsobem lze snadno identifikovat, který záznam byl ze serveru odstraněn a který naopak chybí na zařízení.

Třetí krok probíhá nezávisle pro každý z pořadačů. Pro každý pořadač je možné jednoduchým dotazem zjistit maximální časové razítko. Pomocí operace *GetDataFor-*

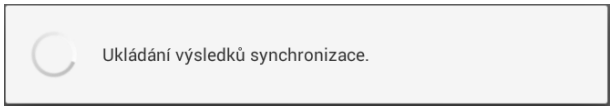
FolderSince pak lze jednoduše získat data všech záznamů, které byly změněny od tohoto časového razítka. Algoritmus zohledňuje i situace, kdy se na základě změny filtru v konfiguraci uživatele v synchronizovaných datech objeví záznam, který se na zařízení nevyskytuje a zároveň má časové razítko s nižší hodnotou, než je maximum. Toto je hlídáno v druhém kroku, kdy na začátku zjištěné maximální razítko může být sníženo na hodnotu razítka záznamu, který chybí na zařízení. Tento krok je také časově nejnáročnější, je zde ale prostor pro případnou optimalizaci synchronizačního algoritmu rozdělením do několika vláken (pro každý pořadač) a využití potenciálu (dnes již v mobilních zařízeních běžných) vícejaderných procesorů.

Čtvrtý krok (stejně jako krok druhý) probíhá pouze, pokud se nejedná o první synchronizaci uživatele, zároveň využívá data získaná v druhém kroku, to je dalším důvodem pro zvolenou implementaci druhého kroku. Během tohoto kroku dochází k aktualizaci aktivit a k zápisu aktuálního mapování těchto aktivit na záznamy.

Pátý krok se na závěr postará, aby se veškeré sesynchronizované změny promítly v uživatelském rozhraní. Každý otevřený záznam či pořadač se v případě, že obsahuje data, která byla změněna, musí načíst znovu. Samozřejmě jsou zohledněny uživatelem právě editované záznamy, u nich k načtení nedochází. K aktualizaci uživatelského rozhraní nemusí docházet pouze na konci synchronizace, ale může k ní dojít v průběhu třetího kroku, pokud je synchronizováno velké množství záznamů.

Při velkém množství záznamů je nutné v průběhu třetího kroku hlídat paměť, kterou aplikace spotřebovává. Velikost haldy přidělené procesu se mění v závislosti na verzi Androidu a v první řadě na základě celkové fyzické velikosti operační paměti zařízení. Dále se mohou záznamy pro různé konfigurace ve velikosti zásadně lišit. Není tedy možné určit pevný počet záznamů, po kterých dojde k přesunu z paměti do databáze. Pro tento účel jsem vytvořil mechanismus, který pomocí informací z třídy *java.lang.Runtime* kontroluje procentuální využití haldy. Po překročení konkrétní hodnoty dojde ke spuštění databázové transakce, která uloží záznamy připravené v paměti.

Nabízí se otázka, proč nejsou záznamy zapisovány do databáze ihned po získání potřebných dat ze serveru. Důvodem je výhodnost využití databázových transakcí. Kromě zřejmé výhody udržení konzistence úložiště v případě neplatných dat či nedokončení transakce z jiného důvodu, je zde ta skutečnost, že veškeré zápisové operace do SQLite databáze na Androidu probíhaly výrazně rychleji, pokud proběhly v rámci transakce. Nevýhodou transakcí je naopak jejich exkluzivní (i pro čtení) zamykání tabulek, do kterých zapisují. Z důvodu udržení plynulosti uživatelského rozhraní, které je velmi závislé na získávání dat z databáze, byl vytvořen popsaný systém předpřipravení záznamů pro zápis a jejich následné hromadné zapsání v rámci jedné transakce při překročení limitu paměti a/nebo po dokončení získávání dat ze serveru. V průběhu zpracování této transakce je uživatelské rozhraní zablokované pomocí dialogu (na obrázku 5.7), aby uživatel neměl pocit, že aplikace přestala odpovídat.



Obrázek 5.7. Dialog blokující uživatelské rozhraní během transakcí.

5.4.3 Zápis změn ze zařízení

Jak již bylo řečeno v předcházejícím oddíle, zápis změn na server probíhá před samotným přenosem změn do zařízení a nebo okamžitě po uložení změn. Pokud je v nastavení aktivní volba „Synchronizovat lokální změny okamžitě“, klient se po uložení záznamu,

po zavolání funkce nebo vykonání workflow akce pokouší spustit částečnou synchronizaci obsahující pouze zápis změn. Pokud není k dispozici spojení na server, nebo nejsou v databázi žádné změny k synchronizaci na server, nedochází k omezení práce s mobilním klientem. Částečná synchronizace je v kontextu daného záznamu dostačující, protože veškeré klíčové operace pro zápis změn vrací jako odpověď aktuální verzi záznamu (viz 5.2.1). Během zápisu změn na server je rozhraní aplikace mobilního klienta blokováno podobným dialogem, jako je na obrázku 5.7. Je tak zabráněno vzniku případných konfliktních editací právě nahrávaných záznamů a případné ztrátě informací. Celý proces zápis změn ze zařízení na server je možné rozdělit do následujících fází:

1. Mazání záznamů.
2. Nahrávání příloh.
3. Vytváření nových záznamů a provádění změn.
4. Volání funkcí.
5. Vykonání workflow akcí.

Mazání záznamů je z hlediska synchronizace jednoduchá záležitost. Algoritmus volá postupně operaci *DeleteData* pro všechny záznamy označené ke smazání (sloupec *deleted* v tabulce *subject*). Po jejím úspěšném vykonání záznam z databáze zařízení odstraní.

Během fáze nahrávání příloh dochází k volání operace *PutAttachment* pro všechny nově pořízené přílohy. Podrobněji se práci s přílohami věnuje oddíl 5.7, v kontextu tohoto oddílu je ale důležité zmínit, že odpovědí operace jsou data nově vzniklého záznamu, který v systému HELIOS Green zastupuje soubor přílohy. Tyto data jsou použita pro nahrazení dočasných informací vzniklých při pořízení přílohy. Rovněž je změněn původní identifikátor na identifikátor přiřazený systémem a aktualizovány vztahy ukazující na přílohu.

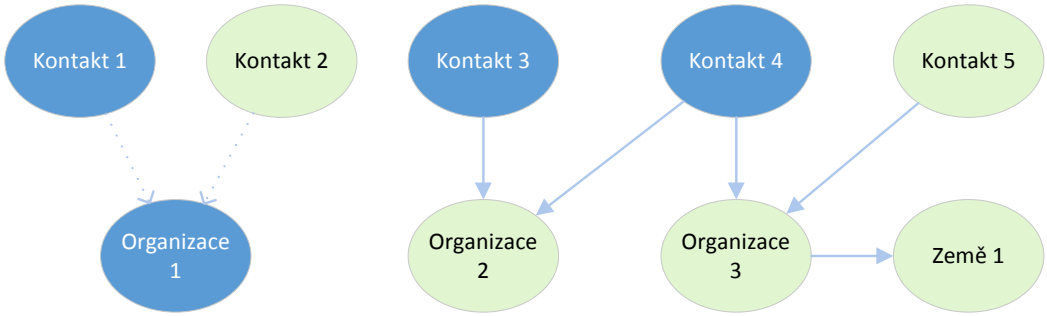
Třetí fáze je ve srovnání s ostatními fázemi nejkomplexnější. Důvodem je způsob, jakým se pracuje s nově vytvářenými záznamy. Každý záznam založený v mobilní aplikaci dostává dočasný[1]) identifikátor, který je záporný a přiřazený na základě globálního čítače. Dočasný identifikátor je dále používán běžným způsobem, používá se při práci se vztahy a ve všech tabulkách týkajících se daného záznamu. Situace se komplikuje v momentě zápisu tohoto záznamu na server, kdy po úspěšném vykonání operace *PutData* klient získá data vytvořeného záznamu a je nutné lokální verzi aktualizovat, včetně změny identifikátoru. Změna identifikátoru musí být, stejně jako v případě příloh, provedena také na všech vztazích, ve kterých je vytvořený záznam navázaný. Důležité je také pořadí nahrávání záznamů. Není možné, aby byl na server odeslán záznam, který některým vztahem odkazuje na nový záznam, o kterém systém ještě „neví“. Z tohoto důvodu jsem vytvořil třídu *DependencyGraph*, která se stará o správné pořadí nahrávání záznamů na server.

Prvním krokem je sběr všech záznamů, které jsou změněny (sloupec *modified*), nebo byly nově vytvořeny (mají záporný identifikátor). Na základě hodnot statických a dynamických vztahů je sestaven graf závislostí. Obrázek 5.8 obsahuje příklad takového grafu.

Každý záznam představuje samostatný uzel grafu, nové záznamy mají zelenou barvu, záznamy změněné mají barvu modrou. Orientovaná hrana mezi uzly značí závislost. Například záznam „Kontakt 3“ závisí na záznamu „Organizace 2“, protože ta je navázána v některém ze vztahů. Tečkovaná spojnice není hranou, značí pouze vztah, který nepředstavuje závislost. Jak příklad ukazuje, graf zpravidla obsahuje více než jednu

[1]) Důvod je zřejmý, není možné si v systému HELIOS Green zarezervovat identifikátor a pracovat s ním.

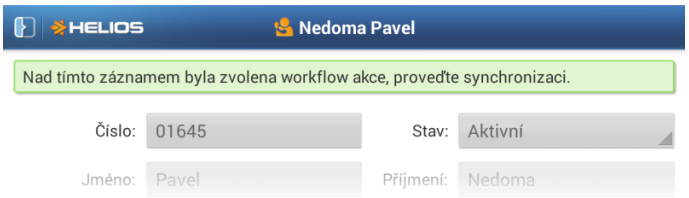
komponentu, výjimkou je aktivní okamžitá synchronizace změn, kdy dochází k pokusu o nahrání změn jediného záznamu ihned po jeho uložení.



Obrázek 5.8. Graf vzájemných závislostí záznamů ukládaných na server.

Algoritmus třídy postupně bere jednotlivé uzly, které ještě nebyly zpracovány. Nad každým uzlem, který nezávisí na žádném dosud nezpracovaném záznamu, dochází k zavolání operace *PutData*, v opačném případě je použito prohledávání do hloubky [1]) pro uložení všech záznamů, na kterých vybraný záznam závisí. V případě volby uzlu „Kontakt 5“, by průchodem do hloubky vzniklo pořadí ukládání: „Země 1“, „Organizace 3“ a nakonec „Kontakt 5“, naopak uzel „Kontakt 2“ by byl zpracován okamžitě. Při každém úspěšném uložení dochází ke zpětné propagaci změny identifikátoru na všechny závisející uzly, po uložení uzlu „Organizace 3“ je tedy nový identifikátor zapsán do uzlů „Kontakt 4“ a „Kontakt 5“. Uzel je považován za zpracovaný, jakmile byla zavolána operace *PutData*, bez ohledu na výsledek. Uzel, jehož ukládání skončilo chybou, blokuje uložení všech závislých záznamů. Cyklické závislosti nejsou povoleny, pokud algoritmus při prohledávání do hloubky navštíví nezpracovaný uzel, ve kterém již byl (algoritmus čísluje navštívené uzly), dojde k zápisu chyby a vynechání uložení všech na cyklu závisejících uzlů.

Poslední dvě fáze – volání funkcí (operace *RunFunction*) a vykonávání workflow akcí (operace *InvokeWfAction*) – jsou z principu totožné. Tyto akce musí být na server přeneseny až po zápisu provedených změn, aby synchronizace odpovídala chování formulářů záznamů. Uživatel může provádět na záznamu změny, ale jakmile uživatel vykoná workflow akci, nebo spustí funkci, záznam je uzamčen pouze pro čtení (viz obrázek 5.9). Prostřednictvím tohoto omezení je dosaženo zachování posloupnosti operací tak, aby odpovídali postupu uživatele. Kdyby záznam uzamčen nebyl, klient by nemohl rozlišit, které změny byly vykonány před, a které po vykonání akce. Zároveň má často volání funkce či vykonání workflow akce za následek modifikaci záznamu, ta by se pak dostala do konfliktu s následnými editacemi.



Obrázek 5.9. Záhlaví formuláře záznamu, na kterém byla již vykonána workflow akce.

Pokud některý se některý ze záznamů účastní více fází (například změna záznamu a vykonání workflow akce), je důležité, aby pro záznam proběhly fáze bez chyb. Pokud

[1]) Více informací o grafech a grafových algoritmech lze nalézt v [23].

některá z fází skončí pro záznam chybou, algoritmus takový záznam do dalších fází nezahrnuje. Více se řešení chybových situací věnuje oddíl 5.6.4.

5.4.4 Zpracování XML

Stejně jako v případě objektově relačního mapování (5.3.2) jsem hledal způsob, jakým zjednodušit práci s XML daty tvořícími strukturu dat používaných při komunikaci s vlastním serverem. Jako nejvhodnější se pro potřeby mobilního klienta ukázala knihovna Simple[1]). Podobně jako ORMLite umožňuje tato knihovna pomocí anotací namapovat pole Java třídy na příslušné XML elementy a atributy. Třída pro data pořadače v odpovědích *GetData…* pak vypadá následujícím způsobem:

```
1  @Root
2  public class FolderData {
3     @Attribute
4     private int ClassNumber;
5     @Attribute
6     private int FolderId;
7     @Attribute(required = false)
8     private long Timestamp;
9     @Attribute(required = false)
10    private Integer RecordTemplate;
11    @ElementList(entry = "Record", empty = false)
12    private ArrayList<RecordData> Records;
13    …
```

Pro zpracování příchozích dat pak stačí pro získání instance třídy rovnou nad proudem dat ze serveru zavolat:

```
1  FolderData fd = new Persister().read(FolderData.class, inputStream);
```

Obdobně jednoduchým způsobem lze převést instanci zpět na XML:

```
1  StringWriter writer = new StringWriter();
2  new Persister().write(fd, writer);
3  String xml = writer.toString();
```

Použití této knihovny je snadné a intuitivní a oproti vlastní implementaci syntaktického analyzátoru XML nabízí značnou úsporu práce.

5.4.5 Propagace změn

Aby nebyl průběh synchronizace ovlivňován událostmi vytváření a obnovování uživatelského rozhraní[2]), běží její kód mimo grafické rozhraní aplikace ve vlastní službě. [24] Do uživatelského rozhraní může zasahovat ale pouze vlákno, které jej vytvořilo, zároveň by služba měla fungovat naprosto nezávisle na uživatelském rozhraní. Proto se veškeré události synchronizace, včetně aktualizace uživatelského rozhraní (pátý krok 5.4.2), rozesílají prostřednictvím broadcastů.

Broadcast je na platformě Android typem hromadné zprávy (Intent), která je doručená všem zaregistrovaným zájemcům posluchačům. Posluchače je možné vymezit jen na úroveň aplikace (můj případ), nebo celého systému. Synchronizační služba rozesílá tyto události:

- změna seznamu dostupných pořadačů,

[1]) Webové stránky knihovny na `http://simple.sourceforge.net/`.
[2]) Například při změně orientace displeje zařízení či při přesunutí aplikace na pozadí.

- změna obsahu konkrétního pořadače,
- změna čísla záznamu (po uložení na server),
- zobrazení a skrytí dialogu blokujícího uživatelské rozhraní (viz obrázek 5.7).

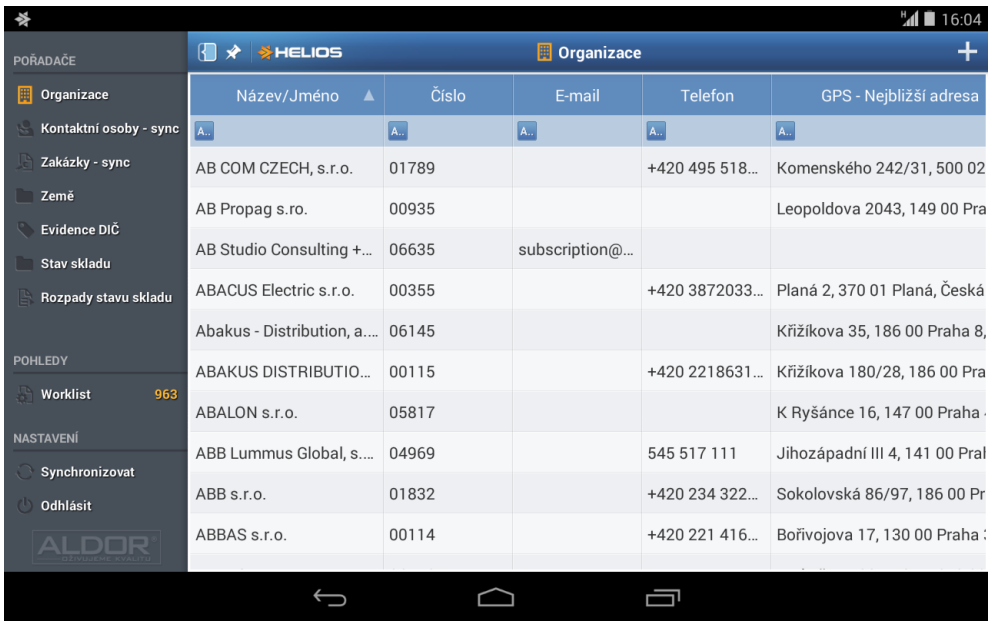
Každá část uživatelského rozhraní se pak přihlásí k odběru pouze pro ní relevantních událostí. Služba také poskytuje rozhraní, které umožňuje zjistit aktuální stav. Toto je důležité pro poslední z uvedených událostí, kdy se součást uživatelského rozhraní zaregistruje k odběru například v momentě, kdy už má být dialog zobrazený.

5.4.6 Pravidelná synchronizace

Pro minimalizaci dopadu offline konceptu mobilního klienta na komfort uživatele byla kromě okamžité synchronizace změn na server (5.4.3) zavedena podpora pro pravidelnou synchronizaci. V nastavení je možné zvolit interval v rozsahu od jedné minuty do šesti hodin, ve kterém se aplikace bude pravidelně pokoušet o synchronizaci dat.

Implementace využívá třídu *android.app.AlarmManager*, která umožňuje načasovat doručení zprávy (intent) v daný časový okamžik. V čase doručení se pak zařízení (v případě že je to nutné) probudí[1]) a podobný způsobem jak v předcházejícím oddíle rozešle zaregistrovanou zprávu. Jelikož je tato zpráva adresována přímo službě synchronizace, systém tuto službu spustí, pokud byla v mezičase uvolněna z paměti. Po dokončení běhu synchronizace je načasována další zpráva pro spuštění následující synchronizace. Plánovaná synchronizace je zrušena pokud se uživatel odhlásí.

Dále je během synchronizace nutné zablokovat přechod zařízení do úsporného režimu. Obzvláště zařízení, které nemají modul pro připojení k mobilní síti přecházejí do úsporného režimu téměř ihned po zhasnutí displeje. Proto je nutné, aby aplikace na začátku synchronizace prostřednictvím třídy *android.os.PowerManager.WakeLock* a její metody *acquire* zamkla systém mimo úsporný režim. Následně je důležité provést odemčení pomocí *release*, aplikace na všech kritických místech provádí uvolnění zámku v blocích *finally*. V případě neodemknutí zámku by mohlo dojít ke zbytečnému vybíjení baterie zařízení.



Obrázek 5.10. Uživatelské rozhraní mobilního klienta na tabletu – zobrazení pořadače.

[1]) Jedná se o probuzení v kontextu úsporného režimu zařízení, nejde tedy o rozsvícení displeje.

5.5 Uživatelské rozhraní

Z hlediska architektury mobilního klienta zbývá prezentační vrstva – tedy uživatelské rozhraní aplikace. Při vytváření uživatelského rozhraní jsem vyšel z rozvahy v oddílu 4.4.2. Uživatelské rozhraní se ve výsledku skládá z několika fragmentů, obrázek 5.10 obsahuje hlavní obrazovku tak, jak je zobrazena na tabletu a na telefonech v režimu na šířku.

Horní část obrazovky tvoří ovládací lišta, která kromě tlačítek akcí, které se mění v závislosti na aktuálně zobrazovaném hlavním panelu, obsahuje název tohoto panelu. Hlavní panel vyplňuje převážnou část obrazovky a na obrázku zobrazuje tabulku s přehledem záznamů zvoleného pořadače. V této hlavní části se následně po výběru záznamu zobrazují formuláře záznamů, kterým se dále věnuje oddíl 5.6. Návrat na přehled je možný pomocí systémového tlačítka zpět. Levá část obrazovky patří bočnímu panelu.

5.5.1 Zobrazení pořadačů

Tabulka přehledu pořadače se snaží možnostmi a chováním o co největší přiblížení přehledovým tabulkám Windows klienta. Sloupce přehledu a výchozí řazení záznamů odpovídá přehledové šabloně nastavené na záznamu Inicializace mobilních zařízení (5.1.1) a to včetně polí z položek pro dokladové třídy a hodnot dynamických vztahů. Stejně jako Windows klient podporuje tabulka rychlé filtrování prostřednictvím editačních polí v záhlaví s podporou pro přepínání druhu rychlého filtru stiskem tlačítka v tomto poli.

Komponentu pro tabulku jsem si z velké části musel navrhnout a implementovat sám, protože systém Android v základu žádnou podobnou komponentu nenabízí. Výsledkem je třída *TableView*, která staví na třídě *HorizontalScrollView* a s pomocí tabulkového rozložení *TableLayout* pro základní strukturu záhlaví, řádku tohoto rozložení *TableRow* pro řádky záhlaví a řádky reprezentující jednotlivé záznamy a komponenty *ListView* pro zobrazení obsahu, utváří široce konfigurovatelnou komponentu pro zobrazování tabulek. Komponenta *ListView* má jednu velmi důležitou vlastnost a tou je *recyklace řádků*. Bez ohledu na celkový počet řádků v tabulce udržuje tato komponenta vizuální reprezentaci pouze pro řádky, které jsou viditelné na obrazovce. Jakmile řádek obrazovku při posunutí opustí, použijí se vizuální komponenty tohoto řádku pro data řádku, který se objevil na druhé straně. Pro definici rozložení a jako poskytovatel dat slouží třída *AbstractTableModel*. Tato třída slouží jako abstraktní předek pro všechny modely tabulek. Implementací abstraktních metod této třídy lze určit počet a šířku sloupců, jejich datové typy a v neposlední řadě zdroj dat. Tabulka se tedy nemusí zabývat původem dat a stejným způsobem zobrazí data z databáze i z nějakého abstraktního datového typu (např. seznamu).

5.5.2 Boční panel

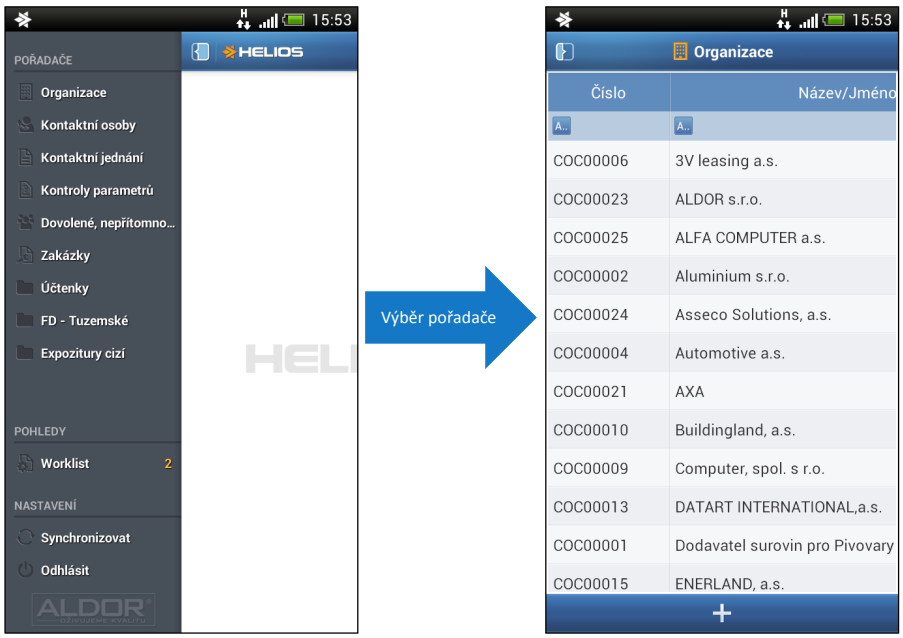
Boční panel slouží pro přepínání pořadačů, pohledů a jiných systémových obrazovek na hlavním panelu. Každý pořadač a pohled si udržuje stav navigace, proto je možné otevřít v každém pořadači záznam a pomocí bočního panelu přepínat mezi otevřenými záznamy. Kromě přepínání hlavního obsahu hlavního panelu slouží boční panel pro odhlášení a ruční spuštění synchronizace, jejíž průběh je indikován animací ikonky tlačítka.

Po vzoru Windows klienta je boční panel skrývatelný a to manuálně pomocí tlačítka hlavní lišty či gesta, nebo automaticky po výběru pořadače. Pro nastavení automatického skrývání slouží tlačítko připínáčku na hlavní liště a ovládá se dlouhým

stiskem. Implementace skrývání panelu využívá mírně upravenou knihovnu SlidingMenu[1]). Knihovna obsahuje komponentu, které se nastaví hlavní fragment a následně jeden nebo dva fragmenty pro boční panely. Dostupnost zdrojového kódu mi umožnila upravit chování knihovny pro své potřeby. Hlavní změnou bylo umožnění interakce uživatele s hlavním panelem při aktivním bočním panelu, výchozí chování knihovny totiž při dotyku v oblasti hlavního panelu automaticky skrývalo boční panel. Druhá neméně důležitá a související změna se týkala změny velikosti hlavního panelu na užší při zobrazení bočního panelu, v základní implementaci docházelo pouze k jeho odsunutí mimo obrazovku bez změny velikosti. Další drobné změny se týkají vymezení oblasti pro gesta pouze na oblast ovládací lišty v případech, kdy je jako hlavní panel rozhraní zobrazena tabulka, která vyžaduje horizontální posun a dále také možnost zapínat a vypínat provedené modifikace prostřednictvím rozhraní komponenty.

5.5.3 Přizpůsobení pro telefony

Jak již bylo řečeno, v režimu „na šířku“ se aplikace na telefonu chová stejným způsobem jako na tabletu. V režimu „na výšku“ ale dochází ke změně chování bočního panelu. Způsob ovládání ilustruje obrázek 5.11.



Obrázek 5.11. Uživatelské rozhraní mobilního klienta na telefonu – zobrazení pořadače.

Boční panel se v tomto režimu vždy po výběru pořadače automaticky skrývá. Změnou je také absence tlačítka připínáčku na hlavní liště, protože toto automatické skrývání nelze v režimu „na výšku“ deaktivovat. Dále nedochází ke změně velikosti hlavního panelu, která by zde neměla z důvodu malé šířky smysl. Knihovna SlidingMenu se tedy chová svým základním způsobem a většina mých úprav je deaktivována. Kromě změněného chování knihovny došlo k přesunutí části hlavní lišty obsahující akční tlačítka k dolnímu okraji obrazovky. Tlačítka jsou tak lépe dostupná při používání telefonu pouze jednou rukou, obzvláště s ohledem na vzrůstající popularitu telefonů s velkými uhlopříčkami displeje.

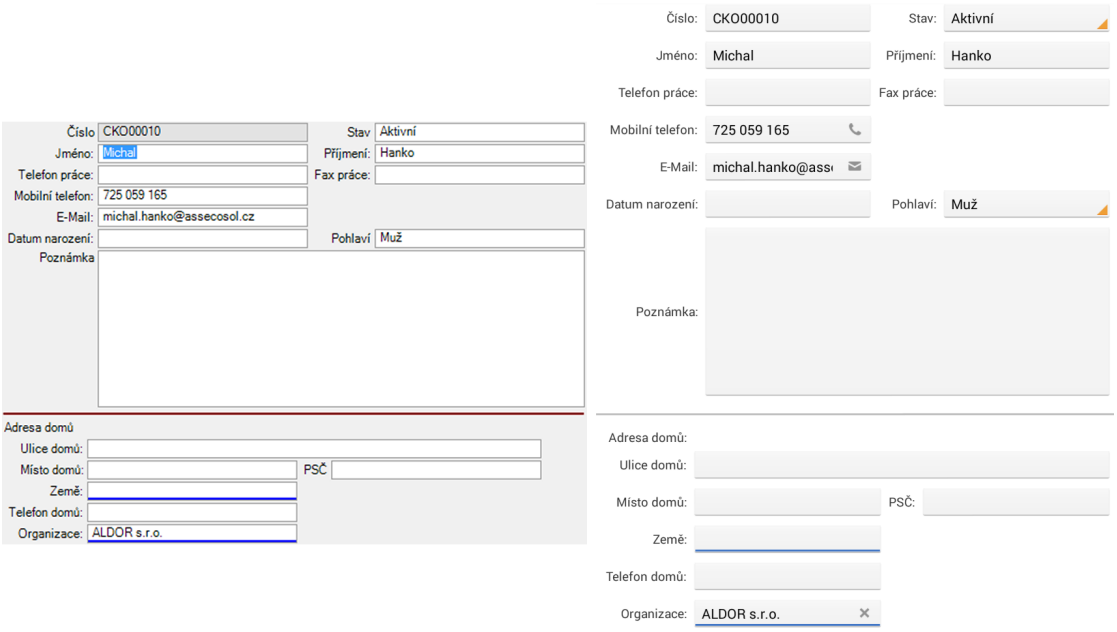
Zvolený přístup s využitím automatického skrývání bočního panelu umožňuje snadnou implementaci bez nutnosti vytvářet specifické obrazovky aplikace pro telefony, vše

[1]) Webové stránky knihovny na `https://github.com/jfeinstein10/SlidingMenu`.

je vyřešeno nastavením parametrů komponenty SlidingMenu při zobrazování hlavní obrazovky. Z hlediska konfigurace aplikace v systému HELIOS Green je nutné brát ohled na nastavené formulářové šablony v inicializaci. Vícesloupcové rozložení formulářů může být při práci s telefonem v režimu „na výšku“ hůře použitelné. Může být žádoucí vytvořit speciální inicializaci pro telefony, která se liší pouze použitými formulářovými šablonami.

5.6 Formuláře

V popisu třídy Inicializace mobilních zařízení (5.1.1) z konfiguračního modulu jsem zmínil, že pro definici vzhledu formulářů na zařízení jsou použity standardní formulářové šablony (3.3.8) systému HELIOS Green. Díky této skutečnosti může administrátor využívat dodávaných nástrojů pro editaci formulářů a také formuláře zkoušet před nahráním do zařízení. Obrázek 5.12 ukazuje srovnání, jakým způsobem je šablona zobrazena Windows klientem a jak je tato šablona interpretována mobilním klientem.



Obrázek 5.12. Srovnání formulářové šablony použité ve Windows klientovi a v mobilním klientovi (napravo).

Podpora formulářových šablon není samozřejmě ve srovnání s Windows klientem kompletní, nicméně je zcela dostatečná pro vytvoření pokročilých formulářových rozložení. Počet sloupců rozložení není omezený, záleží pouze na úsudku autora šablony, počet sloupců se může lišit i mezi jednotlivými částmi. (Na obrázku jsou části odděleny horizontální čárou hnědé resp. šedé barvy.) Mobilní klient dále respektuje atribut *ColSpan*, toto je možné vidět u atributu „Ulice domů“. V souvislosti s tímto atributem došlo k přizpůsobení interpretace atributu pro šířku položky (*Width*). Protože formulář na mobilním zařízení má vždy pevnou šířku a komponenta editoru je vždy zarovnána na celé sloupce, rozhodl jsem se, že šířka bude řídit šířku vizuální komponenty vzhledem k počtu zabíraných sloupců rozložení. Pokud je zadána šířka v procentuálním tvaru, pak se počítá procentuální počet sloupců, pro hodnotu šířky 75% ve čtyř sloupcovém rozložení je komponenta vykreslena do tří sloupců. Pro absolutní hodnoty bylo stanoveno pravidlo, že hodnota pod 200 včetně značí vykreslení pouze do jednoho sloupce a vyšší hodnoty pak plnou šířku *ColSpan*.

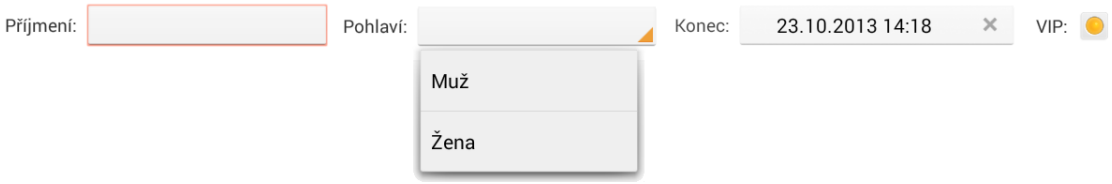
V případě popisků jsou podporovány překladové XML šablony, je tedy v budoucnu možné provozovat aplikaci i v mezinárodních organizacích. [1]) Popisek je podporován pouze na levé straně editoru, případně je možné ho skrýt. Ze základních vlastností (atributů) pole formuláře jsou dále podporovány tyto vlastnosti:

- **Required** – povinnost vyplnění, editor s prázdnou hodnotou je červeně orámován.
- **Protect** a **ReadOnly** – pole pouze pro čtení, pouze hodnoty *true* a *false* (nejsou podporovány dynamické podmínky).
- **Invisible** – skryté pole, není zobrazeno na formuláři, ale je možné ho použít v přehledu a v aplikační logice.
- **InputType** – Druh editoru pro textové hodnoty, podporovány jsou pouze: *text*, *checkbox*, *textarea* a *label*. Ostatní druhy jsou zobrazeny jako *label*.
- **Height** – výška editoru, pouze pro *InputType=textarea*, počet řádek editoru je pak vypočten jako *Height/15*.

Kromě těchto atributů jsem pro některé specifické funkce mobilního klienta zavedl atributy vlastní. Jejich způsob zápisu a účel představím v následujícím oddíle.

5.6.1 Editory

Pro editaci atributů vzniklo několik druhů editačních a zobrazovacích komponent. Ukázka všech podporovaných editorů atributů je na obrázku 5.13.



Obrázek 5.13. Přehled editorů atributů.

Prvním editorem (zleva) na obrázku je klasické textové pole. Červené orámování značí prázdné povinné pole (XML atribut *Required*). Tento editor je využíván pro většinu textových a číselných atributů. Pokud je editor víceřádkový (*InputType=textarea*), jsou aktivovány funkce automatického doplňování pro klávesnici, aby bylo zadávání delších textů komfortnější. Pro číselné atributy je hodnota formátována podle editačního stylu a klávesnice je zobracena v numerickém režimu. Pro textové atributy jsem zavedl speciální atribut *MobileLink*. Tento atribut deklaruje, že hodnota atributu je využitelná v prostředí mobilního zařízení. Při jeho použití dojde k zobrazení tlačítka akce v pravé části editoru (viz obrázek 5.12, atributy „Mobilní telefon“ a „E-Mail“). Podporovány jsou tyto hodnoty atributu *MobileLink*:

- **EMAIL** – atribut obsahuje emailovou adresu, tlačítko otevírá novou zprávu.
- **PHONE** – atribut obsahuje telefonní číslo, tlačítko nabízí číslo pro vytočení hovoru.
- **URL** – atribut obsahuje webovou adresu, tlačítko otevírá prohlížeč.

Implementace této funkcionality vytváří zprávy (intenty) se specifickou akcí a předává je systému, ten pak otevře výchozí aplikaci, nebo dá uživateli na výběr, která aplikace má akci obsloužit, pokud je jich pro zpracování tohoto typu zprávy zaregistrováno více. Tato funkcionalita reaguje na požadavek na práci s kontaktními údaji z 4.1.3.

[1]) Překlad samotné aplikace zatím nebyl vytvořen, ale z oddílu 4.4.2 vyplývá, že překlad aplikace není komplikovaná záležitost.

Druhý editor je použit v případě, že má atribut nastavený výčtový typ, tak jak je to popsáno v oddíle 3.3.4. Po kliknutí na editor je nabídnut výčet vizuálních hodnot a po výběru jedné z nich dochází k zápisu příslušné databázové hodnoty.

Třetí editor je používán pro sloupce s typem *datum a čas*. Podporuje režimy: datum a čas, pouze datum a pouze čas, režim je určen na základě nastaveného editačního stylu. Tlačítko křížku v pravé části editoru slouží pro vymazání hodnoty. Nastavení a editace hodnoty probíhá prostřednictvím dialogu na obrázku 5.14 po kliknutí na editor mimo oblast mazacího tlačítka.

Obrázek 5.14. Editační dialog pro nastavení data a času.

Ve spojitosti s editorem data a času je důležité zmínit pro mobilní zařízení specifický atribut *MobileDefault*, který slouží obecně pro dynamické výchozí hodnoty atributů, ale zatím byl implementován pouze pro datový typ datum a čas. Dynamické výchozí hodnoty jsou vyhodnocovány v momentě vytvoření nového záznamu. Pro výchozí hodnotu data a času s ohledem na aktuální čas jsem zavedl následující syntaxi:

```
MobileDefault="NOW[{zaokrouhlení dolů}]+/-{počet}{časová jednotka}
+/-{počet}{časová jednotka}+/-…"
```

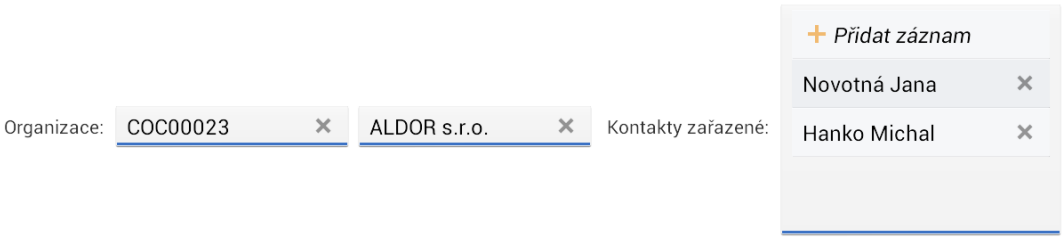
Klíčové slovo *NOW* je povinné, ostatní údaje jsou volitelné. Podporované časové jednotky jsou: m – minuta, h – hodina, d – den, M – měsíc a y – rok. Zápis pro druhý den aktuálního měsíce by pak vypadal takto:

```
MobileDefault="NOW[M]+1d"
```

Čtvrtý a zároveň poslední editor slouží jako klasické zaškrtávací tlačítko logické hodnoty. Na formuláři se vyskytuje pro všechny textové atribut s nastaveným *InputType=checkbox*. Zaškrtnutý stav zapisuje do atributu hodnotu „A“ a odškrtnutý stav „N“ tak, jak je to v systému HELIOS Green pravidlem.

5.6.2 Vztahy

Pro práci se vztahy jsem logicky vytvořil dva typy editorů. Na obrázku 5.15 je ukázka editoru pro statický vztah (nalevo) a editoru pro dynamický vztah (napravo).

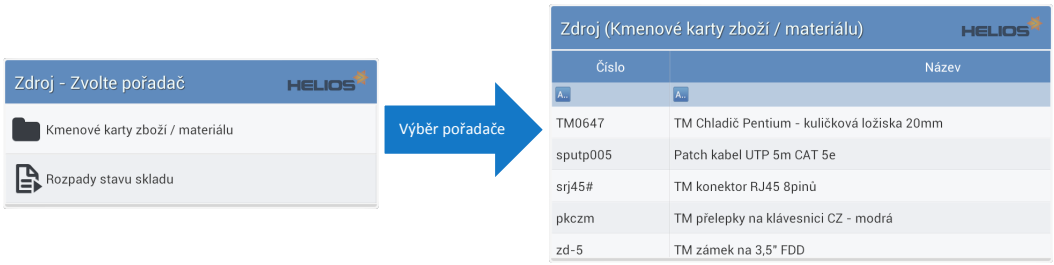


Obrázek 5.15. Přehled editorů vztahů.

Komponenty pro editaci vztahu jsou specifické svým modrým podtržením, to bylo zvoleno s ohledem na co největší podobnost s editory Windows klienta. Editor statického vztahu na obrázku je ve skutečnosti složen ze dvou editorů, první zobrazuje referenci a druhý název. Vnitřně však pracují se stejným vztahem, proto je nutné, aby každý z editorů reflektoval změny provedené tím druhým. Pro zajištění této konzistence využívají systém monitorování změn hodnoty popsaný v oddíle 5.3.4.

Ovládání editoru statického vztahu je velmi podobné editoru atributu pro datový typ *datum a čas*. Pro mazání vztahu slouží opět malé tlačítko křížku v pravé části políčka, chování při stisku se ale mírně liší. Pokud není vztah prázdný, dochází po stisku k otevření náhledu formuláře navázaného záznamu. Stisknutí pro prázdný vztah otevírá sekvenci dialogů pro výběr nové hodnoty, stejnou sekvenci je možné vyvolat dlouhým stiskem editoru s vyplněným vztahem. Možnost náhledu navázaného záznamu je aktivní i v případě, že je editor zobrazený pouze v režimu pro čtení.

Dynamické vztahy se nenacházejí ve formulářové šabloně, ve Windows klientovi jsou zobrazeny v bočním panelu (viz poslední odstavec oddílu 3.3.8). Na mobilním zařízení (obzvláště pak na telefonech) je k dispozici podstatně menší plocha pro zobrazení formuláře, proto jsem se rozhodl umístit tento druh editoru přímo pod obsah formulářové šablony. Editor dynamických vztahů má formát seznamu všech navázaných záznamů. Každý řádek reprezentující záznam má podobné tlačítko pro odvázání ze vztahu, jako je tomu v případě statických vztahů, a stejně tak je možné otevřít náhled navázaného záznamu. Seznamu navázaných záznamů předchází speciální řádek sloužící pro navazování nových záznamů do vztahu, jeho stisknutí otevírá identickou sekvenci dialogů jako editor statických vztahů. V případě dynamických vztahů s omezenou kardinalitou na jeden záznam dochází ke skrytí tohoto řádku, jakmile je již jeden záznam navázán. Obrázek 5.16 obsahuje sekvenci dialogů pro výběr záznamu pro navázání do vztahu.



Obrázek 5.16. Dialogy pro navazování záznamu do vztahu.

První z dialogů slouží pro výběr pořadače. Tento dialog je zobrazen pouze v případě, že má vztah nastavené dva a více cílové pořadače a obsah alespoň dvou je synchronizován do zařízení. V případě jediného pořadače nebo po výběru pořadače v prvním dialogu je zobrazen druhý dialog s obsahem pořadače. Obsah pořadače je zobrazen za použití stejné komponenty tabulky a přehledové šablony jako v případě základního zobrazení pořadače (5.5.1). Je tedy možné použít rychlé filtrování záznamů pro pohodlné dohledání nového záznamu pro navázání. V případě, že není žádný z cílových pořadačů synchronizován do zařízen, dojde při pokusu o navázání nového záznamu ke zobrazení chybové zprávy.

5.6.3 Dokladové třídy

V oddíle 3.3.8 jsem popsal podstatu dokladových tříd. Uvedl jsem, že položky jsou zobrazeny v dolní části okna záznamu a že je možné je editovat přímo v řádcích tabulky. Tento koncept Windows klienta jsem na mobilním zařízení aplikoval pouze částečně,

tabulka položek je zobrazena v dolní třetině formuláře, ale editace přímo v tabulce neprobíhá. Pro práci s atributy a vztahy konkrétní položky je nutné tuto položku označit v tabulce položek. Následně dojde k překreslení zbylých dvou třetin formuláře na formulář dat položky. Tento postup je znázorněn na obrázku 5.17.

Obrázek 5.17. Práce se záznamy dokladové třídy.

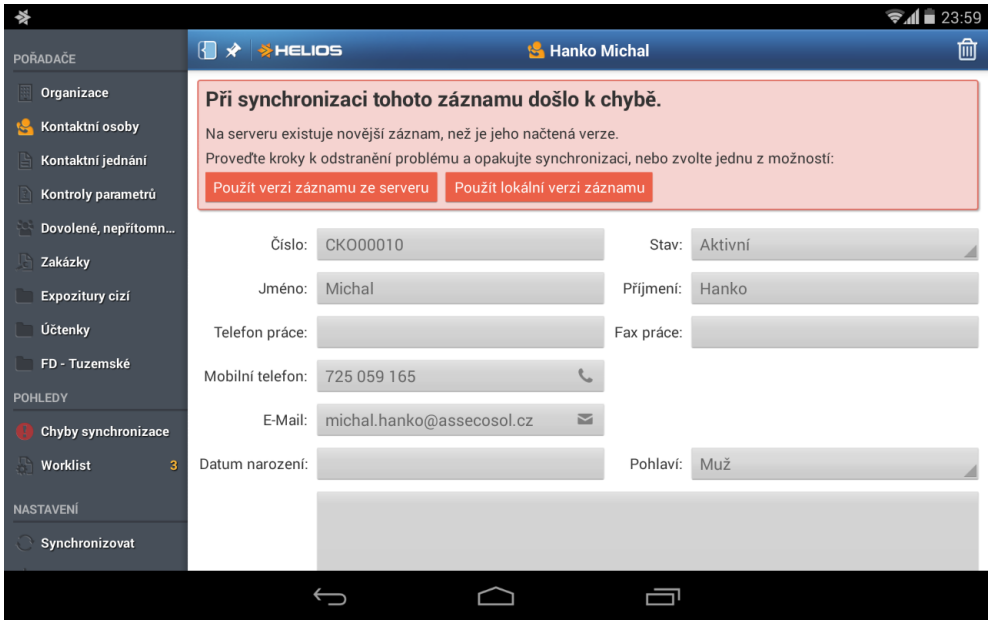
Tabulka využívá stejné komponenty jako doposud zmíněné tabulky se záznamy, liší se pouze zdrojem dat, kde místo výsledku SQL dotazu nad uživatelskými daty (5.3.3) jsou použity řádky z *Datastore* položek na entitě (5.3.4). Kromě zvýraznění řádku aktivní položky je uživatel o vybrané položce informován číslem řádku na konci názvu záznamu na hlavní liště. Samotný formulář položky respektuje formulářovou šablonu položek nastavenou na záznamu Inicializace mobilních zařízení (5.1.1). Pro návrat na formulář hlavičky slouží systémová klávesa „zpět“.

Přidávání nových položek je prováděno akčním tlačítkem na liště. Akční tlačítko pro odstranění položky je zobrazeno, pokud je některá z položek aktivní. Po přidání nové položky je automaticky vyplněn systémový atribut „číslo řádky“, který je povinný na všech položkách dokladových tříd. Předvyplněná hodnota je určena na základě maxima tohoto atributu přes veškeré existující položky záznamu.

5.6.4 Řešení chybových zpráv

Pokud se při synchronizaci změn záznamu na server (5.4.3) vyskytne chyba s kódem odpovídající chybě způsobené obsahem záznamu, tedy nejedná se o chybu komunikace nebo konfigurace serveru. Je tato chyba a její kód zapsána do příslušných polí v tabulce *subject*. Záznamy se zapsanou chybou jsou pak agregovány v pohledu „Chyby synchronizace“ a v běžných přehledech jsou zvýrazněny červenou barvou. Pokud je pak takový záznam otevřen, je příslušná chybová zpráva zobrazená v záhlaví formuláře. Na základě kódu chyby a aktuálního stavu[1]) záznamu jsou případně zobrazeny akce umožňující nápravu chyby. Obrázek 5.18 obsahuje ukázku formuláře záznamu, při jehož synchronizaci došlo k chybě.

[1]) Stavem se myslí plánované operace nad záznamem, které se nepodařilo synchronizovat – mazání, ukládání změn, volání funkce atd.



Obrázek 5.18. Zobrazení chyby synchronizace – souběžná editace.

Chyby vzniklé synchronizací změn se nejčastěji týkají porušení integritních omezení. Proto, kromě akce na zrušení provedených změn, bývá nejčastějším řešením úprava dat tak, aby odpovídala požadavkům. Pro ostatní operace a jejich kombinace jsou pak ve většině případů k dispozici akce pro kompletní nebo částečné zrušení operace. Rušení operací je možné díky systému ukládání původní kopie dat (5.3.3).

Výjimkou je konflikt verzí záznamu zmiňovaný již v popisu chybových zpráv rozhraní vlastního serveru (5.2.3). Chyba tohoto typu je zobrazená na obrázku 5.17. V tomto případě došlo v čase mezi stažením záznamu a synchronizaci změn zpět na server k souběžné editaci záznamu z jiného místa (např. jiným uživatelem prostřednictvím Windows klienta). Nahrání změn ze zařízení by mělo za následek narušení konzistence dat, neboť by tyto změny byly aplikovány na záznam s potenciálně odlišným obsahem. Pokud je detekována tato chyba, dojde ke stažení aktuální verze záznamu ze serveru a k uložení její skryté kopie v databázi zařízení. Rozhraní formuláře pak ukazuje lokální verzi záznamu v režimu pouze pro čtení a nabízí akci pro zachování této verze, nebo naopak použití aktuální verze získané ze serveru. Uživatel si může být jistý, jakou verzi chce použít a zvolit příslušnou akci přímo.[1]) Druhou variantou je otevření aktuální verze prostřednictvím jiného klienta a vizuální kontrola rozdílů oproti lokální kopii.

Dosavadní nasazení mobilního klienta ukázalo minimální výskyt tohoto typu chyby a s ohledem na toto pozorování považuji aktuální implementaci za dostačující. Nicméně do budoucna je vhodné zvážit rozšíření uživatelského rozhraní o prostředí, které by umožňovalo porovnání dvou verzí záznamu přímo na zařízení spolu s nástrojem pro jednoduché manuální sloučení změn z těchto verzí.

5.7 Práce s přílohami

Způsob, jakým mobilní klient s přílohami pracuje, jsem v textu nastínil již na několika místech. Cílem tohoto oddílu je shrnutí všech skutečností a doplnění informací z oblasti konfigurace příloh a práce s nimi v uživatelském rozhraní mobilní aplikace.

[1]) Například provedl pouze jednoduché změny, které si pamatuje, a není tedy náročné je na aktuální serverové verzi vykonat znovu.

5.7.1 Stahování příloh

Podobně, jako je tomu u synchronizace, pro stahování příloh do zařízení existuje samostatná služba. Důvodem je skutečnost, že stahování příloh probíhá na pozadí a na rozdíl od jejich nahrávání nezávisle na procesu synchronizace. Služba pracuje s frontou požadavků na stažení příloh. Pro každý požadavek dochází k volání operace *GetAttachment* a na základě identifikátoru z odpovědi dochází ke stažení dat přílohy (podrobněji proces popisuje oddíl 5.2.1). Jsou rozlišovány požadavky dvou typů: požadavek na předběžné stažení a požadavek z uživatelského rozhraní na stažení za účelem otevření přílohy. Druhý zmíněný typ má vyšší prioritu.

Vztahy příloh pro synchronizaci jsou konfigurovány na záznamu Inicializace mobilních zařízení (5.1.1) pomocí dynamického vztahu položek „Vztahy příloh“. Do tohoto vztahu jsou metadata dynamických vztahů vázány prostřednictvím záznamu třídy „Konfigurace vztahů příloh“. Záznamy této třídy specifikují, jaký druh úložiště mobilního zařízení je pro ukládání příloh využíván. Interní úložiště je vždy v hlavní paměti zařízení a na neupravených[1]) zařízeních může do tohoto úložiště přistupovat pouze aplikace klienta a operační systém. Externí úložiště je pak veřejně přístupné, ale zároveň může být umístěno na paměťové kartě, která často poskytuje vyšší kapacitu, než interní úložiště.

Celý mobilní klient vyvinutý v rámci této práce je koncipován tak, aby mohl pracovat bez dostupného připojení na server. Protože přílohy ve srovnání se samotnými záznamy představují mnohem větší datovou zátěž jak pro komunikační kanály, tak i pro samotné úložiště mobilního zařízení, implementoval jsem pro přílohy specifický konfigurační mechanismus pro vymezení příloh, které se mají předběžně stáhnout do zařízení a které mají být naopak dostupné pouze na vyžádání.

Mechanismus se zakládá na skutečnosti, že pro získávání příloh jsou dostatečné zástupné záznamy, tak jak je popisuje oddíl 5.3.6. Jakmile je do zařízení synchronizován celý záznam přílohy (cílový pořadač vztahu příloh je zahrnut v konfiguraci synchronizace), je tento záznam chápán jako záznam označený pro předběžné stažení. Po každé synchronizaci aplikace zkontroluje, zda se v průniku množiny synchronizovaných záznamů příloh a množiny zástupných záznamů u vztahů příloh neobjevil nový záznam a ten je pak případně zařazen službě do fronty požadavků. Tímto způsobem lze zařídit, že na zařízení budou k dispozici například veškeré přílohy záznamů s velikostí pod 100 kB. Při otevírání přílohy, která byla již předběžné stažená, dochází při dostupném přípojení na server k ověření kontrolního součtu soubor, aby byla zaručena aktuálnost a neporušenost dat. Služba stahování příloh samozřejmě kontroluje volné místo na zařízení, aby nedocházelo k chybovým situacím.

5.7.2 Nahrávání příloh

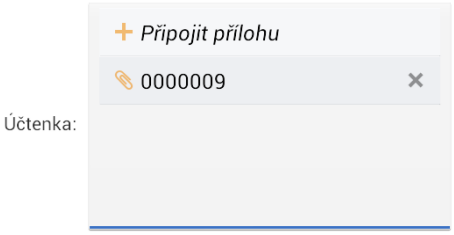
Nahrávání příloh probíhá, jak již bylo řečeno, během synchronizace změn ze zařízení na server (5.4.3). Operace *PutAttachment* pracuje na rozdíl od operace pro opačný směr s daty v textovém kódování Base64 [15]. Protože operační paměť přidělená aplikaci je velmi limitovaná, nepřipadá v úvahu, aby byl celý soubor převedený na řetězec nahraný v paměti. Proto klient webové služby využívá při odesílání požadavků třídu *android.util.Base64InputStream*, která pracuje přímo s proudem dat využívá tak minimální prostor operační paměti. Po úspěšném nahrání souboru je nutné řešit podobnou situaci jako při nahrávání nových záznamů – tedy změnu čísla záznamu reprezentující přílohu a propagaci změny na všechny odkazující vztahy. V případě příloh je také

[1]) Někteří uživatelé si na mobilních zařízeních aktivují režim „root“ uživatele i pro nesystémové aplikace, ty pak mají neomezená práva na celé zařízení a v některých případech představují hrozbu pro bezpečnost systému.

nutné přejmenovat soubor přílohy, aby nedocházelo při požadavcích na otevření přílohy k opakovanému stahování.

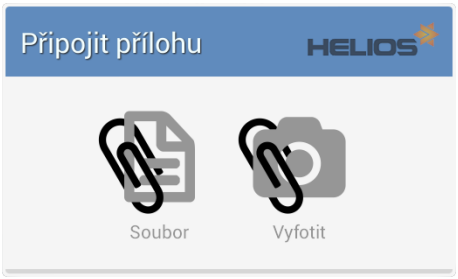
5.7.3 Uživatelské rozhraní

Vztahy příloh jsou speciálním případem dynamických vztahů. Je tedy logické, že i editor pro vztahy příloh bude vyházet z již představeného editoru dynamických vztahů. Příklad editoru s jednou navázanou přílohou ukazuje obrázek 5.19.



Obrázek 5.19. Editor vztahu příloh.

Editory příloh jsou umisťovány také pod obsah formulářové šablony a jsou řazeny před dynamické vztahy. Pro jednodušší orientaci uživatele je každý řádek opatřen ikonou sponky. Mazání navázaných záznamů se provádí také křížkem, ale odstranění záznamu musí uživatel potvrdit dialogem. Samotné fyzické mazání „osiřelých“ souborů příloh je v kompetenci služby stahující přílohy. Pro přidání nové přílohy slouží opět první řádek seznamu. Jeho stisknutí vyvolá dialog z obrázku 5.20.



Obrázek 5.20. Dialog pro připojení přílohy.

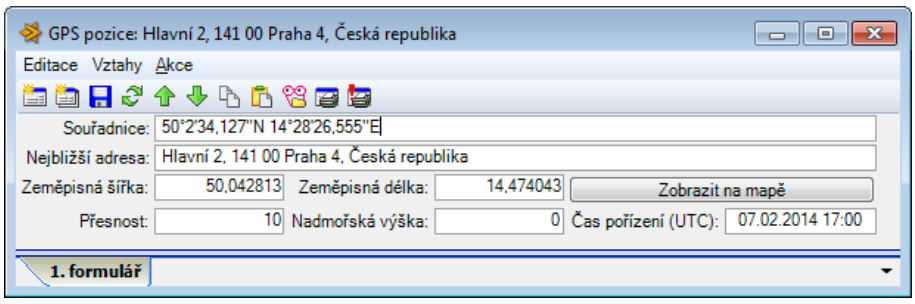
Dialog nabízí dva zdroje příloh. První volba „Soubor“ předá požadavek prostřednictvím akce (intent) cizí aplikaci, která se u systému registrovala jako aplikace, která poskytuje soubory. Mezi takové aplikace obvykle patří správci souborů, tato volba tedy představuje způsob, jakým lze připojit přílohu téměř jakéhokoliv typu. Druhá volba „Vyfotit“, jak již název napovídá, spouští pomocí akce (intent) aplikaci fotoaparátu, ve které uživatel pořídí fotografii. Jakmile je s výsledkem spokojen a potvrdí jej, je fotografie předána zpět mé aplikaci, která fotografii uloží jako novou přílohu.

5.8 GPS

Možnost zaznamenat polohu GPS je nejen součástí jedné z agend zadání (4.1.3), ale v prvé řadě je jednou z oblastí, ve které mobilní klient rozšiřuje základní možnosti systému HELIOS Green. Tento oddíl rozebírá práci s GPS pozicemi od jejich reprezentace v systému až po práci s nimi v uživatelském rozhraní mobilního klienta a zároveň uzavírá tuto kapitolu.

5.8.1 Třída GPS pozice

Na rozdíl od příloh není pro ukládání GPS pozice v systému HELIOS Green implementována žádná podpora. Proto jsem v rámci konfiguračního modulu vytvořil nonsubjektovou třídu „GPS pozice“. Jako alternativa připadala v úvahu textová reprezentace v rámci atributů (všechna data pozice v jednom řetězci), řešení využívající záznamy jsem ale zhodnotil z hlediska systému HELIOS Green jako čistší a do budoucna lépe rozšiřitelné. Záznam výsledné třídy je na obrázku 5.21.



Obrázek 5.21. Záznam třídy GPS pozice.

Kromě základních údajů o GPS pozici, jako je: zeměpisná šířka, zeměpisná délka, přesnost, nadmořská výška a čas pořízení, obsahuje záznam také dvě pole vyplňovaná aplikační logikou. Pole s názvem „Souřadnice“ slouží jako reference záznamu a obsahuje souřadnice přepočtené z formátu desetinných stupňů na kombinaci stupňů, minut a vteřin. Pro uživatele jsou tak k dispozici vždy dva formáty zápisu, dostačující pro zadávání do většiny aplikací třetích stran bez nutnosti dodatečného převodu. Název záznamu zastupuje atribut s názvem „Nejbližší adresa“. Jak již název napovídá, tento atribut obsahuje adresu nejblíže zeměpisným souřadnicím a slouží pro lepší orientaci uživatele a pro kontrolu dat, protože samotné souřadnice jsou pro člověka těžko interpretovatelné. O vyplnění nejbližší adresy se stará aplikační logika záznamu při jeho ukládání. Dohledání adresy probíhá prostřednictvím Google Geocoding API, které poskytuje webovou službu umožňující dohledání nejbližší adresy na základě souřadnic a naopak. [25]

Pomocí tohoto záznamu je možné evidovat pozici u jakékoliv existující i nově vytvářené třídy. Stačí na této třídě založit UDA statický vztah (3.7) odkazující na záznamy třídy „GPS pozice“. Pokud se rozhodneme evidovat GPS pozici například organizací, může být vhodné doplnit souřadnice pro již existující záznamy organizací. K tomuto účelu jsem vytvořil funkci „Dohledat GPS pozice podle adresy“. Funkce je spouštěna nad pořadačem třídy „GPS pozice“ a jako parametry vyžaduje vztah, do kterého má záznamy navazovat, a filtr vymezující záznamy, pro které má hledání probíhat. Dalším parametrem je formátovací řetězec, který definuje, jakým způsobem atributy a statické vztahy cílové třídy tvoří adresu, na základě které mají být souřadnice nalezeny. Funkce pak v implementaci využívá již zmíněný opačný směr rozhraní Google Geocoding a na základě výsledků, které vrátí pro naformátované adresy, funkce vytvoří záznamy třídy „GPS pozice“ a naváže je k příslušným záznamům z cílového pořadače.

5.8.2 Konfigurace

Pro synchronizaci a pořizování záznamů GPS pozice na zařízení stačí, aby byl příslušný statický vztah přítomný ve formulářové šabloně na záznamu Inicializace mobilních zařízení (5.1.1). A aby byl cílový pořadač vztahu pro konkrétního uživatele nastaven k synchronizaci. Specifickou vlastností konfigurace těchto cílových pořadačů je absence

formulářové šablony. Záznamy třídy „GPS pozice“ mají na zařízení specifický režim zobrazování a z toho vyplývající rozsah atributů, které je nutné přenést na zařízení. Tato základní konfigurace je pro plnohodnotnou práci s GPS pozicí na zařízení dostačující, zavedl jsem ale další konfigurační možnosti pro upřesnění chování zařízení.

Podobně jako v případě dynamické výchozí hodnoty atributů (5.6.1) vznikl nový atribut pro pole vztahu na úrovni formulářové šablony. XML atribut s názvem *MobileLocation* má následující formát:

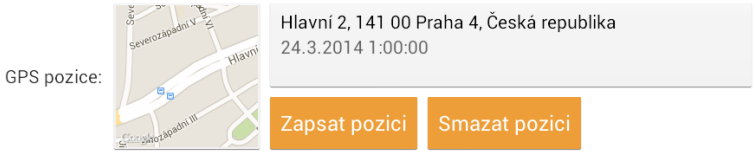
```
MobileLocation="gpsOnly;accuracyLimit=15;timeout=10;writeOnCreate;
writeOnUpdate"
```

Příklad obsahuje veškeré podporované parametry, v praxi může atribut obsahovat pouze část parametrů, nebo úplně v šabloně chybět. Následuje popis funkcí jednotlivých parametrů:

- **AccuracyLimit** – tento parametr deklaruje nejhorší povolenou přesnost (přesnost pozice musí být menší nebo rovna této hodnotě), kterou je možné na zařízení zapsat.
- **GpsOnly** – pokud je tento parametr v atributu přítomný, zařízení nedovolí uložit pozici z jiného než GPS zdroje.
- **Timeout** – časový limit, uvedený v sekundách, po který bude vyžadováno splnění dvou výše uvedených parametrů. Absence tohoto parametru znamená neomezenou platnost omezení.
- **WriteOnCreate** – tento parametr aktivuje automatický zápis pozice při prvním uložení nového záznamu.
- **WriteOnUpdate** – podobně jako předcházející parametr – tento aktivuje automatický zápis pro každé uložení záznamu kromě prvního uložení.

5.8.3 Mobilní zařízení

Pokud jsou splněny konfigurační předpoklady uvedené v předcházejícím oddíle, je na formuláři záznamu na mobilním zařízení místo klasického editoru statického vztahu zobrazen speciální editor pro GPS pozici. Ukázka editoru je na obrázku 5.22.



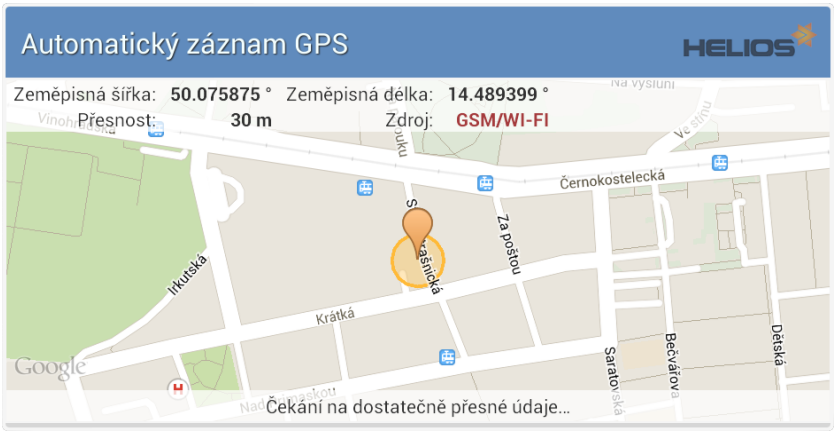
Obrázek 5.22. Editor GPS pozice.

Data v editoru z ukázky odpovídají záznamu z obrázku 5.21. Kromě nejbližší adresy doplněné serverem (nebo souřadnic, pokud adresa chybí) je zobrazen čas pořízení. V pravé části editoru je náhled mapy stažený na základě souřadnic v navázaném záznamu. Pro získání mapy používá mobilní aplikace Google Static Maps API. Tato služba umí poskytnout část mapy okolo zadaných souřadnic s požadovaným rozlišením a úrovní přiblížení. Licence služby bohužel neumožňuje dlouhodobé ukládání získaných dat [26], proto je obrázek s náhledem mapy stahován vždy při otevření záznamu a v případě nedostupného datového spojení je zobrazena na jeho místě zástupná ikona. Tento náhled mapy resp. zástupná ikona slouží pro otevření detailního zobrazení pozice prostřednictvím dialogu na obrázku 5.23.

Obrázek 5.23. Záznam třídy GPS pozice v mobilním klientovi.

Základem dialogu je mapa oblasti zobrazená pomocí knihovny Google Maps Android API v2[1]), konkrétně pomocí komponenty *MapView*. Tato knihovna nabízí široké možnosti integrace rozhraní známého z aplikace Mapy od Google do prostředí vlastní aplikace. Při otevření dialogu je pohled zacílen na pozici záznamu s vyznačenou oblastí přesnosti, mapu je ale možné libovolně posouvat, přibližovat či otáčet. Nad komponentu mapy jsem doplnil panel s atributy záznamu a tlačítkem pro přesun mapy zpět na pozici záznamu. Ovládací rozhraní dialogu uzavírá tlačítko „Navigovat“, které při stisku odešle akci (intent) se souřadnicemi, adresovanou všem aplikacím, které se zaregistrovaly pro obsluhu cílů navigace. Uživatel tak může snadno pozici předat navigační aplikaci a nechat se na ní dovést.

Pokud chce uživatel navázanou pozici odstranit, obsahuje pro tento účel tlačítko „Smazat pozici“. Tlačítko „Zapsat pozici“ pak slouží pro ruční záznam pozice. V průběhu záznamu je zobrazen dialog, jehož ukázka je na obrázku 5.24.



Obrázek 5.24. Dialog automatického záznamu GPS pozice.

Dialog na obrázku je zobrazen v režimu automatického zápisu, tedy jako reakce na konfigurační parametr *WriteOnCreate* nebo *WriteOnUpdate*. Zápis reagující na tyto parametry je spouštěn bez ohledu na viditelnost či editovatelnost příslušného editoru. Automatický záznam je úspěšně dokončen v okamžiku splnění minimálních požadavků z konfigurace a po uplynutí alespoň pěti sekund. Dialog pro ruční záznam má navíc ve své dolní části tlačítko pro potvrzení zápisu, toto tlačítko je rovněž možné stisknout až v momentě splnění minimálních požadavků na kvalitu pozice. Na obrázku 5.23 je možné

[1]) Webové stránky knihovny na `https://developers.google.com/maps/documentation/android/`.

vidět indikaci nedostatečného zdroje pozice z důvodu přítomnosti parametru *GpsOnly* v konfiguraci vztahu.

Pro získávání pozice využívá implementace třídu *LocationManager*. Pomocí této třídy se u poskytovatelů zaregistruje k odběru aktualizací polohy zařízení, od kterých pak s určitou frekvencí (v závislosti na poskytovateli) aplikace získává informace o aktuální poloze. Systém Android nabízí dva poskytovatele polohy. Poskytovatel polohy *NETWORK_PROVIDER* určuje pozici na základě vysílače mobilní sítě, ke kterému je zařízení připojeno, a dále na základě bezdrátových WiFi sítí v okolí. Tyto údaje systém zpracovává pomocí serverů společnosti Google, proto tento poskytovatel potřebuje ke své funkčnosti kromě aktivního WiFi adaptéru a/nebo modulu pro mobilní sítě také přístup k internetu. Poskytovatel polohy *GPS_PROVIDER* pak pracuje s polohou získané ze sítě satelitů systému GPS, pokud je zařízení vybaveno příslušným modulem. Údaj o přesnosti polohy ukládaný do záznamu je získaný od poskytovatelů polohy. Hodnota přesnosti vyjadřuje poloměr kruhu, ve kterém se zařízení s 68% pravděpodobností nachází. [27]

Kapitola 6
Testování

Testování je prověřování funkčnosti produktu. Provádí se za účelem nalezení a následné eliminace chyb. Na základě výsledků testování lze získat představu o stavu aplikace, například jak je aktuální verze vhodná pro zahájení distribuce mezi koncové uživatele. Proces testování je významnou součástí každého softwarového projektu, například podle pravidla 40–20–40 (návrh/analýza–kódování–testování) by mělo být testování věnováno 40 % časových prostředků projektu. [28]

Tato kapitola popisuje, jakým způsobem bylo prováděno testování v průběhu realizace diplomové práce. Dále se zaměřuje na popis výsledků testů kompatibility a zhodnocení dopadů vnějších vlivů na výkonnost klienta.

6.1 Testování během vývoje

Již od prvních funkčních prototypů byla implementace testována. Hlavním cílem bylo vytvoření regresních testů, pomocí kterých bylo možné snadno ověřit, zda následná implementace nových funkcionalit neovlivnila dosavadní implementaci.

6.1.1 Vlastní server

Testování serveru je vzhledem k jeho povaze nenáročné na volbu nástrojů. Volání služby probíhá jednoduchým HTTP požadavkem a tělo zprávy tvoří formát XML, pro který je dostupné velké množství editorů. Jako velmi vyhovující se ukázala aplikace SoapUI [1]), která je přímo navržena pro testování rozhraní webových služeb.

Protože jakákoliv změna v logice serveru pro transformaci dat systému HELIOS Green může mít za následek změny v obsahu odpovědí služby, vytvořil jsem pro klíčové operace rozhraní (5.2.1) sadu testovacích požadavků prostřednictvím aplikace SoapUI. Pro všechny tyto požadavky jsem uložil kopie XML odpovědí. Při každé větší změně implementace serveru (obzvláště před nasazením nové verze na testovací servery) jsem tyto operace spouštěl a porovnával odpovědi s již uloženými kopiemi. Odpovědi většiny operací samozřejmě závisí na obsahu databáze systému HELIOS Green, proto se veškeré zmíněné operace prováděly na mé vlastní izolované testovací kopii databáze, která vznikla pouze za účelem tohoto testování.

Samotné porovnávání XML dat odpovědí by bylo příliš náročné provádět pouhým jejich pročítáním, protože objem odpovědí u některých operací může dosahovat stovek kB. Dále některé z prvků XML nemají pevně dané pořadí v dokumentu, dvě odpovědi tedy mohou být datově identické, ačkoliv při textovém porovnání se mohou lišit. Z tohoto důvodu jsem prozkoumal několik pokročilejších nástrojů pro porovnávání XML dokumentů. Nejlépe se pro mé požadavky osvědčila aplikace Altova DiffDog [2]).

[1]) Webové stránky aplikace na `http://www.soapui.org/`.
[2]) Webové stránky aplikace na `http://www.altova.com/diffdog.html`.

6.1.2 Mobilní klient

V případě mobilní aplikace není možné aplikovat předpřipravené testy tak jednoduchým způsobem jako v případě vlastního serveru. Po prozkoumání dostupných možností pro automatizaci testování uživatelského rozhraní na platformě Android [29], jsem došel k závěru, že implementace testů tak, aby byly pro projekt přínosem, by byla nesmírně časově náročná. Proto jsem se rozhodl mobilní aplikaci a její uživatelské rozhraní testovat manuálně.

Pro manuální testy mobilního klienta vznikla dokumentace ve formě testovacích scénářů. Hlavní výhodou tohoto přístupu je možnost jejich aplikace při regresním testování v průběhu dosavadního a i budoucího vývoje. Tyto testy vznikaly vždy jako odpověď na implementaci nové funkcionality a zároveň sloužily také jako dokumentace správného chování funkcionality. Každý testovací scénář se skládá z popisu výchozí situace (konfigurace, stav aplikace), druhou část tvoří sekvence kroků, které je potřeba vykonat, tyto kroky jsou proloženy kontrolními body, které popisují očekávaný stav a chování aplikace. Scénář je pak uzavřen postupem pro návrat do výchozího stavu tak, aby bylo možné test provést znovu. Sekvence prováděných a očíslovaných kroků umožňuje jednoduchou identifikaci momentu, ve kterém se projevuje chyba a je tak možné snadno ověřit, zda došlo k její opravě. Obecný zápis kroků pak umožňuje, aby byly scénáře v případě potřeby vykonávány i jinou osobou. Příklad jednoho z používaných testovacích scénářů obsahuje příloha B.

6.2 Kompatibilita mobilního klienta

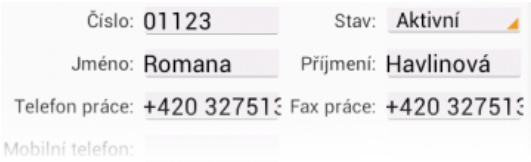
Ačkoliv bylo v oddíle 4.4.1 rozhodnuto o podpoře systému Android od verze 3.2, z obrázku 4.3 se statistikou ve stejném oddíle lze jasně vyčíst, že zastoupení jednotlivých verzí z rodiny verze 4 je velmi vyrovnané. Kromě rozdílné verze systému se běžně používaná zařízení mohou výrazně lišit hardwarovými parametry. S přihlédnutím k těmto skutečnostem a také k požadavku v zadání práce na alespoň dva různé telefony a dva různé tablety jsem zavedl skupinu testovacích zařízení. Přehled základních informací o testovacích zařízení obsahuje tabulka 6.1.

Kód	Výrobce	Model	Systém	Uhlopříčka	Rozlišení
P1	HTC	One S	4.1.2	4,3“	540 × 960
T1	Asus	Nexus 7	4.4	7“	1280 × 800
T2	Modecom	FreeTAB 8014	4.2	8“	1024 × 768
P2	HTC	One (M7)	4.4	4,7“	1080 × 1920
P3	Samsung	Galaxy S4 mini	4.2	4,3“	540 × 960

Tabulka 6.1. Mobilní zařízení využívaná pro testování kompatibility.

Zvolená zařízení pokrývají většinu nejrozšířenějších verzí systému Android a zároveň se dostatečně liší z hlediska hardwarových parametrů. Telefon *P1* a obzvláště tablet *T1* byly soustavně využívány během vývoje aplikace a také na nich nejčastěji probíhalo testování podle scénářů z předcházejícího oddílu. Testování na ostatních zařízení probíhalo rovněž na základě testovacích scénářů, ale v delších časových intervalech.

Během tohoto testování kompatibility bylo objeveno několik chyb závislých na verzi systému. Jedna z chyb se například projevovala pouze na systému Android verze 4.1.2, kdy docházelo ke špatnému nastavení okrajů komponent formuláře. Ukázka této chyby je na obrázku 6.1.



Obrázek 6.1. Problém s grafickým rozložením na Androidu 4.1.2.

6.3 Měření výkonu

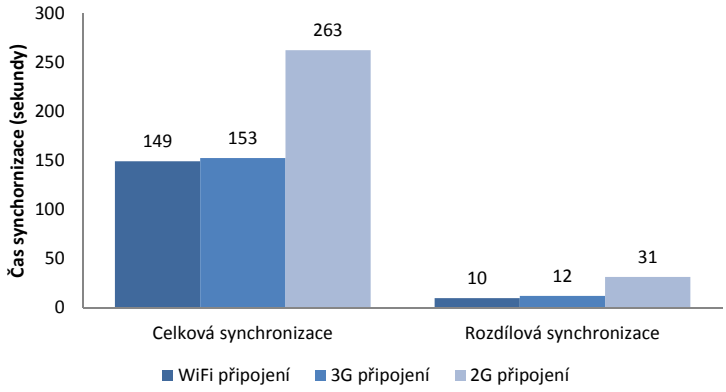
Tento oddíl se zaměřuje na zhodnocení výkonu implementovaného řešení. Veškeré provedené testy probíhaly na zařízení *T1* z tabulky 6.1.

6.3.1 Vliv kvality mobilního datového spojení

Mobilní internet je dnes k dispozici téměř po celé republice. Nejvyšší rychlosti internetu v sítích 3. a 4. generace jsou ale dostupné pouze ve větších městech. Na ostatních místech se musí uživatel často spokojit s relativně nízkou rychlostí mobilního připojení 2. generace. Teto test se proto zaměřuje na zhodnocení dopadu rychlosti sítě na délku trvání synchronizace. Testovány byly následující tři situace:

- Připojení k internetu prostřednictvím Wifi. (802.11n a připojení 120/20 Mbps)
- Připojení k internetu prostřednictvím mobilní sítě 3. generace (UMTS/HSPA).
- Připojení k internetu prostřednictvím mobilní sítě 2. generace (GPRS/EDGE).

Připojení k internetu prostřednictvím mobilní sítě 4. generace jsem se rozhodl netestovat z důvodu nedostatečného pokrytí signálem v oblasti provádění testů. Měření pro každou situaci probíhala desetkrát a výsledný čas je průměrem všech opakování. Testovaná inicializace obsahovala šest pořadačů a celkem 9 590 záznamů, z nichž většina byla rovnoměrně rozložena ve čtyřech pořadačích a 500 záznamů bylo součástí aktivního procesu workflow. V druhé testovací situaci byla po celou dobu testu prováděna vizuální kontrola indikátoru připojení[1]). Graf na obrázku 6.2 shrnuje výsledky měření.



Obrázek 6.2. Vliv kvality připojení na celkový čas synchronizace.

Výsledky testů ukázaly, že zatímco v mobilních sítích 3. generace je mobilní klient schopný poskytnout téměř shodný výkon jako při připojení prostřednictvím WiFi, v sítích 2. generace je synchronizace výrazně pomalejší. Obzvláště rozdílová synchronizace, která vykonává mnoho objemově malých požadavků trvá až třikrát déle.

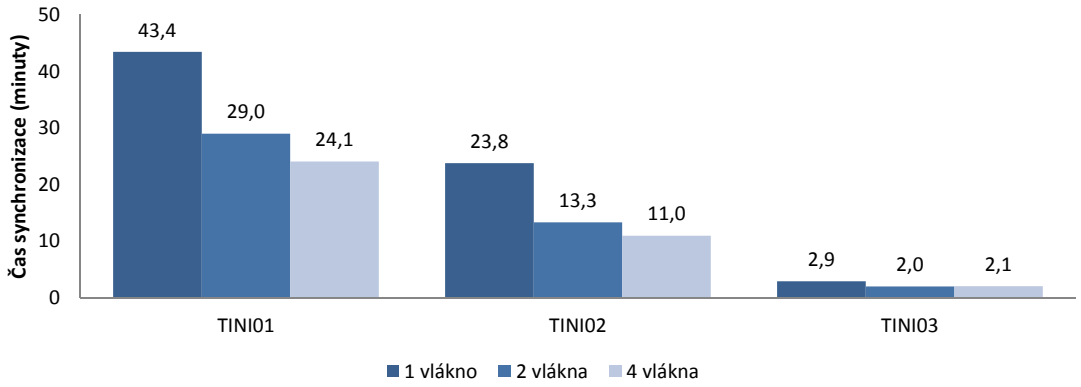
[1]) Pokud zařízení vyhodnotí signál pro připojení 3. generace jako nedostatečný, může se přepnout na síť 2. generace.

6.3.2 Vícevláknová synchronizace

Možnost vícevláknové synchronizace je poprvé zmíněna v oddíle 5.4.2 o synchronizaci směrem do zařízení. Implementace této části synchronizace byla navržena s ohledem na jednoduché rozdělení synchronizace jednotlivých pořadačů a vykonávání této části pomocí individuálních vláken. Otázkou ale zůstávalo, kolik vláken je vhodné povolit jako maximum. Cílem tohoto testu je porovnat dopad počtu použitých vláken na celkovou dobu trvání synchronizace. Test byl prováděn pro tyto tři inicializace:

- **TINI01** – 9 pořadačů, 69 500 záznamů.
- **TINI02** – 9 pořadačů, 44 700 záznamů.
- **TINI03** – 6 pořadačů, 7 500 záznamů.

Test probíhal s připojením prostřednictvím WiFi. Stejně jako v předchozím testu bylo každé měření desetkrát zopakováno a výsledky zprůměrovány. Počet vláken pro jednotlivé testy byl zvolen s ohledem na počet jader (4) procesoru zařízení *T1*. Pro každou verzi testu byla aplikace zkompilována s příslušným nastaveným počtem vláken. Výsledné průměry celkových časů shrnuje graf na obrázku 6.3.



Obrázek 6.3. Vliv počtu vláken na celkový čas synchronizace.

Z grafu je na první pohled zřejmé, že použití více než jednoho vlákna zkracuje celkovou dobu synchronizace. Rozdíl mezi jedním a dvěma vlákny je velmi výrazný, jedná se téměř o poloviční úsporu času. Úspora času při použití čtyř vláken již není tak výrazná, u nejmenší inicializace dokonce došlo k mírnému zpomalení. Z principu synchronizace vyplývá, že míra zrychlení je závislá na rozložení záznamů v pořadačích, pokud například jeden z pořadačů počtem záznamů výrazně převyšuje ostatní pořadače, bude celková doba synchronizace velmi závislá na stažení všech dat pro tento pořadač.

Na základě výsledků tohoto testu jsem se rozhodl nastavit počet vláken využívaných během synchronizace záznamů do zařízení na maximální počet dvou vláken. Tento test také ověřil schopnosti aplikace pracovat s větším množstvím záznamů, kdy navzdory dlouhému času první celkové synchronizace dosahoval čas rozdílové synchronizace pro inicializaci *TINI01* rozumných 45 sekund.

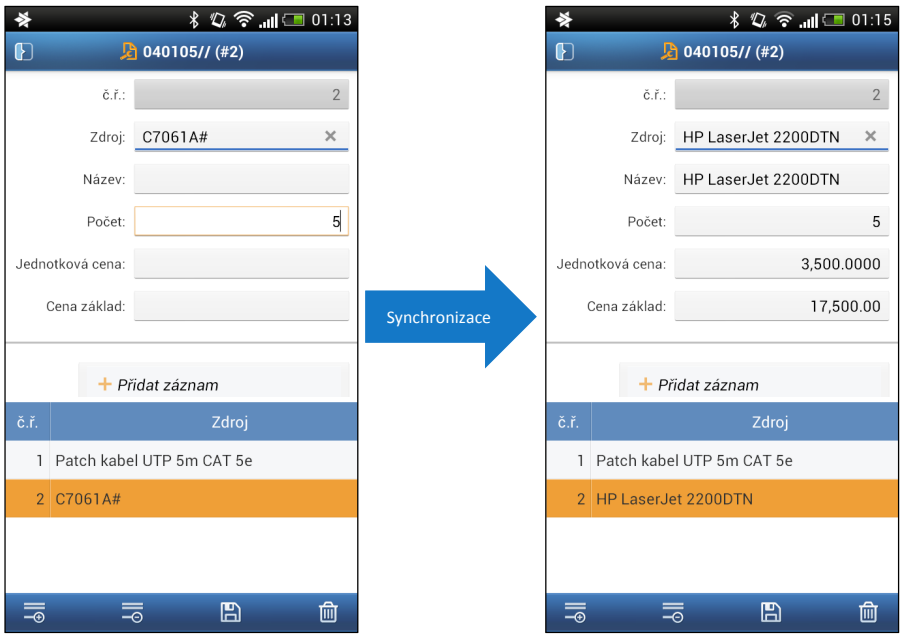
6.4 Podpora pro agendy ze zadání

Závěrem této kapitoly zhodnotím splnění požadavků jednotlivých agend zadání, které jsem popsal v oddíle 4.1. Zaměřím se pouze na specifické požadavky agend, které doposud nebyly v rámci kapitoly 5 představeny. Na základě této kapitoly lze považovat za splněné tyto požadavky:

- subjektové třídy včetně tříd s položkami (dokladové třídy),
- nonsubjektové třídy,
- atributy,
- statické vztahy,
- dynamické vztahy,
- workflow,
- práce v offline režimu,
- práce s většími objemy dat (viz předcházející oddíl),
- dynamické výchozí hodnoty atributů a speciální zacházení s atributy použitelnými na mobilním zařízení. (viz 5.6.1)

6.4.1 Vytváření objednávek (cenotvorba)

Pro agendu obchodníka (4.1.1) je klíčové respektování aplikační logiky při vytváření objednávek a nabídek pro zákazníka. Obrázek 6.4 ilustruje postup vytváření zakázky prostřednictvím mobilního klienta na telefonu.

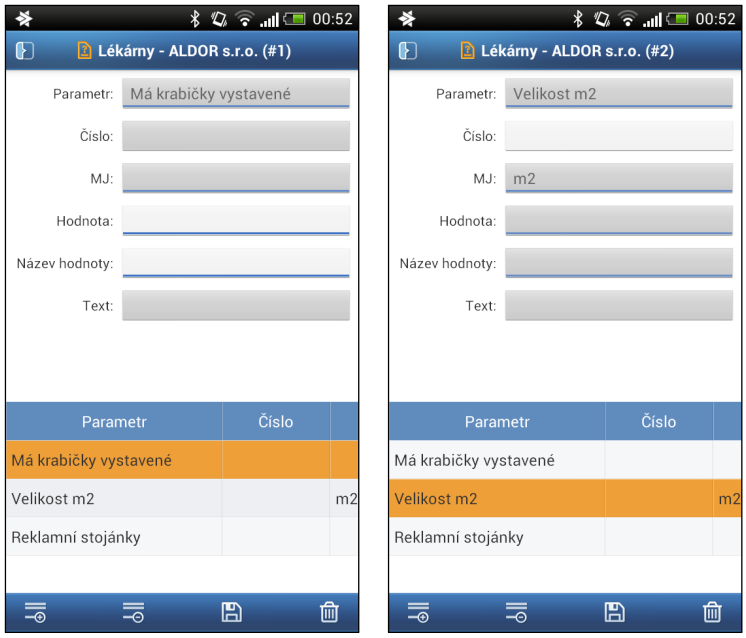


Obrázek 6.4. Práce se záznamy dokladové třídy.

Prvním krokem je výběr záznamu nonsubjektové třídy „Rozpad stavu skladu“ a jeho navázání do položkového vztahu zdroj. Dále je potřeba zadat počet kusů výrobku. Následná synchronizace vytvoří odpovídající položku na záznamu v systému HELIOS Green a mobilní klient získá jako odpověď aktuální verzi záznamu.

Položka na aktuální verzi záznamu má správně nahrazený zdroj odpovídajícím záznamem třídy „Kmenové karty zboží a materiálu“. Na základě ceníku byl také správně doplněn atribut „Jednotková cena“, na základě které byl správně vypočítán atribut „Cena základ“ s ohledem na objednávané množství. K uplatnění základní aplikační logiky na straně serveru tedy dochází správně.

Druhou podstatnou funkcionalitou pro agendu obchodníka byla podpora pro třídu „Parametry provozovny“, která reprezentuje dotazník. Chování záznamů této třídy v mobilním klientovi ilustruje obrázek 6.5.

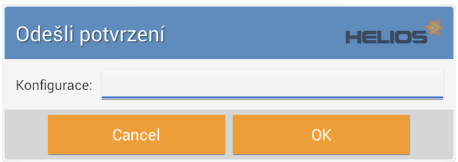


Obrázek 6.5. Práce se záznamem dotazníku.

Rozdíl mezi formulářem první položky záznamu a formulářem druhé položky je patrný na první pohled. Zatímco parametr „Má krabičky vystavené“ dává ve vztahu hodnota na výběr záznamy hodnot „Ano“ a „Ne“, parametr „Velikost m2“ jak již název napovídá, očekává číslo pro předvyplněnou měrnou jednotku. Podporu pro dotazníky lze tedy také považovat za dostatečnou.

6.4.2 Vykazování práce techniků v rámci servisních zásahů

Agenda servisního technika (4.1.2) vyžadovala kromě možnosti pořizovat z mobilního zařízení přílohy, kterou detailně popisuje oddíl 5.7, také podporu pro volání funkcí s parametrickým oknem, kterou tato práce pokryla spíše okrajově. Obrázek 6.6 obsahuje ukázku parametrického okna funkce.



Obrázek 6.6. Parametrické okno funkce na mobilním klientovi.

Parametrické funkce používají pro vytvoření formuláře stejnou logiku, jaká je použitá pro běžné formulářové šablony (5.6).

Z hlediska testování je tato agenda velice důležitá, protože během vytváření této práce došlo k odkladu jejího odevzdání a aplikace během prvního čtvrtletí roku 2014 dospěla do verze, která byla použitelná v běžném provozu. V tomto momentě společnost ALDOR spustila pilotní provoz mobilního klienta, kterého se účastnilo několik techniků. Tato agenda byla tedy prověřena v praxi. Zároveň mělo testování mezi reálnými uživateli za následek několik věcných připomínek k používání aplikace, které jsem mohl implementovat. Jedním takový požadavkem byla úprava práce s položkami (5.6.3), kde se dříve pro návrat na formulář hlavičky používalo akční tlačítko na hlavní liště aplikace, namísto aktuálně používaného systémového tlačítka „zpět“.

6.4.3 Sběr dat v terénu se záznamem pozice pomocí GPS

Agenda pro sběr informací v terénu se záznamem GPS pozice (4.1.3) je ve svém popsaném principu nenáročná. Největší komplikace spočívala v implementaci záznamu pozice GPS (do detailu popisuje oddíl oddíl 5.8) a výsledek umožňuje široké možnosti konfigurace záznamu pozice od vymezení požadované přesnosti až po automatický a povinný záznam. Požadavek na předvyplnění aktuálního času řeší přímo třída „GPS pozice“, která registruje čas pořízení pozice, nebo je možné pro tento účel použít dynamickou výchozí hodnotu atributu (5.6.1).

Kapitola 7
Závěr

Cílem této diplomové práce byl návrh a implementace mobilního klienta pro ERP systém HELIOS Green na platformě Android.

Po úvodním seznámení se systémem HELIOS Green a zavedení terminologie na začátku kapitoly 3, jsem se zaměřil na způsob, jakým systém ukládá data (3.4) a prozkoumal jsem možnosti propojování tohoto systému s jinými systémy (3.6). Na základě získaných informací jsem pokračoval analýzou a návrhem principů fungování vyvíjeného mobilního klienta v kapitole 4. Výchozím bodem pro tuto analýzu byl rozbor jednotlivých agend ze zadání (4.1), aby bylo možné určit rozsah funkcionalit, které musí mobilní klient podporovat. S ohledem na informace získané analýzou rozhraní webové služby systému HELIOS Green, vznikl koncept vlastního serveru (4.3) s cílem minimalizace objemů datových přenosů a zlepšení možností konfigurace celého řešení.

Samotné implementaci aplikace se věnuje kapitola 5, která se zaměřuje především na implementačně zajímavé části. Popisuje výsledek od struktury konfiguračního modulu v systému HELIOS Green (5.1) a přes rozbor principů použitích při implementaci vlastního serveru (4.3) se dostává k samotné mobilní aplikaci. U té se zaměřuje na popis specifických změn, které byly provedeny v návrhu datového modelu (5.3) oproti systému HELIOS Green, aby byla v co největší míře zachována existující aplikační logika. Dále je podstatná část kapitoly věnována použitým algoritmům u obou směrů synchronizace (5.4) a také uživatelskému rozhraní aplikace (5.5). U toho je důraz kladen na způsob, jakým se aplikace přizpůsobuje rozdílným typům zařízení, tedy jak se liší styl ovládání na telefonech a na tabletech. (5.5.3) Kapitolu uzavírá popis práce s přílohami (5.7) a záznamu GPS pozice (5.8), tedy dvou specifických oblastí, které rozšiřují portfolio systému HELIOS Green o prvky ze světa mobilních zařízení. Jakým způsobem byla aplikace v průběhu vývoje testována popisuje kapitola 6. Kromě samotného testování se zaměřuje na měření výkonu aplikace při práci s reálnými daty (6.3) a celou práci uzavírá zhodnocením úrovně splnění požadavků jednotlivých agend (6.4).

Výsledkem této diplomové práce je mobilní aplikace pro platformu Android, která umožňuje na základě jednoduché konfigurace přenést na mobilní zařízení většinu agend podporovaných systémem HELIOS Green díky obecné implementaci podpory pro jednotlivé stavební prvky používané systémem pro vytváření těchto agend. Předností aplikace je i její otevřenost pro další vývoj a rozšíření. V průběhu prvního čtvrtletí roku 2014 začala tuto aplikaci společnost ALDOR nabízet pod názvem „HELIOS Mobile“. Od svého uvedení byla tato aplikace nasazena do testovacího provozu v několika organizacích, které si produkt objednaly. Aplikace se také stala finalistou prvního kola soutěže „IT produkt roku“ [30], pořádané nakladatelstvím IDG IT a dále se stala vítězem ankety „Produkt roku 2013“ [31] v kategorii „Systémy pro počítačové řízení údržby“ časopisu „Řízení a údržba průmyslového podniku“. S ohledem na všechny tyto skutečnosti považuji výslednou aplikaci za schopnou naleznout uplatnění v reálném nasazení.

Literatura

[1] E. J. Umble, R. R. Haft a M. M. Umble. Enterprise resource planning: Implementation procedures and critical success factors. *European journal of operational research*, 146(2):241–257, 2003.

[2] Asseco Solutions, a.s. Dokumentační portál produktu HELIOS Green, stav z 2.4.2014.
`https://forum.helios.eu/green/doc/cs/index.php?title=Hlavn%C3%AD_strana`.

[3] I. Halaška a M. Valenta. Podklady pro přednášky předmětu 36DBS, 2007.

[4] World Wide Web Consortium (W3C). SOAP Version 1.2 Part0: Primer, stav z 10.4.2014.
`http://www.w3.org/TR/2007/REC-soap12-part0-20070427/`.

[5] Alex Rodriguez. RESTful Web services: The basics, stav z 10.4.2014.
`https://www.ibm.com/developerworks/webservices/library/ws-restful/`.

[6] Ecma International. The JSON Data Interchange Format, stav z 11.4.2014.
`http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf`.

[7] Wikipedia. Android (operating system), stav z 20.4.2014.
`http://en.wikipedia.org/wiki/Android_(operating_system)`.

[8] Android Open Source Project. Dashboards, stav z 20.4.2014.
`http://developer.android.com/about/dashboards/index.html`.

[9] Android Open Source Project. Supporting Tablets and Handsets, stav z 11.4.2014.
`http://developer.android.com/guide/practices/tablets-and-handsets.html`.

[10] Android Open Source Project. Design Principles, stav z 20.4.2014.
`https://developer.android.com/design/get-started/principles.html`.

[11] Microsoft Corporation. .NET Framework Support for SOAP Formats, stav z 13.4.2014.
`http://msdn.microsoft.com/en-us/library/vstudio/4cxy91t2(v=vs.100).aspx`.

[12] J. Vokřínek. Podklady pro přednášky předmětu A4M36AOS – Architektury orientované na služby, 2012.

[13] Microsoft Corporation. C# Attributes, stav z 13.4.2014.
`http://msdn.microsoft.com/en-us/library/aa287992(v=vs.71).aspx`.

[14] World Wide Web Consortium (W3C). Web Services Description Language (WSDL) Version 1.2, stav z 13.4.2014.
`http://www.w3.org/TR/2002/WD-wsdl12-20020709/`.

[15] The Internet Society. The Base16, Base32, and Base64 Data Encodings (RFC 4648), stav z 14.4.2014.
`https://tools.ietf.org/html/rfc4648`.

[16] E. Gamma, R. Johnson, R. Helm a J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[17] Digital Equipment Corporation. The SQL-92 standard, stav z 14.4.2014.
http://www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt.

[18] SQLite. SQL Features That SQLite Does Not Implement, stav z 14.4.2014.
http://www.sqlite.org/omitted.html.

[19] Oracle. Trail: The Reflection API, stav z 14.4.2014.
http://docs.oracle.com/javase/tutorial/reflect/index.html.

[20] Gray Watson. Using With Android, stav z 14.4.2014.
http://ormlite.com/javadoc/ormlite-core/doc-files/ormlite_4.html#Use-With-Android.

[21] SQLite. Datatypes In SQLite Version 3, stav z 14.4.2014.
http://www.sqlite.org/datatype3.html.

[22] Microsoft Corporation. Rowversion (Transact-SQL), stav z 20.4.2014.
http://technet.microsoft.com/en-us/library/ms182776.aspx.

[23] J. Kolář. *Teoretická informatika*. Česká informatická společnost, 2004.

[24] Android Open Source Project. Services, stav z 20.4.2014.
http://developer.android.com/guide/components/services.html.

[25] Google. The Google Geocoding API, stav z 1.5.2014.
https://developers.google.com/maps/documentation/geocoding/.

[26] Google. Static Maps API V2 Developer Guide, stav z 1.5.2014.
https://developers.google.com/maps/documentation/staticmaps/.

[27] Android Open Source Project. Location, stav z 1.5.2014.
http://developer.android.com/reference/android/location/Location.html.

[28] B. Mannová. Podklady pro přednášky předmětu 36SI2, 2008.

[29] Android Open Source Project. UI Testing, stav z 1.5.2014.
http://developer.android.com/tools/testing/testing_ui.html.

[30] Computerworld.cz. Finalisté prvního kola soutěže IT produkt roku 2014, stav z 1.5.2014.
http://computerworld.cz/it-produkt/finaliste-prvniho-kola-souteze-it-produkt-roku-2014-50757.

[31] Řízení a údržba průmyslového podniku. Ocenění čtenářské ankety Produkt roku za loňskou sezónu je za námi, stav z 1.5.2014.
http://udrzbapodniku.cz/hlavni-menu/artykuly/artykul/article/oceneni-ctenarske-ankety-produkt-roku-za-lonskou-sezonu-je-za-nami/.

Příloha A
Zkratky

CRM	Customer relationship management, systém pro podporu komunikace se zákazníkem.
ERP	Enterprise Resource Planning, komplexní informační systém pro podpoření agend společnost.
GPS	Global Positioning System, vojenský globální družicový polohový systém.
HeG	ERP systém HELIOS Green společnosti Asseco Solutions, a.s.
HTML	HyperText Markup Language, značkovací jazyk pro hypertext.
HTTP	HyperText Transfer Protocol, internetový protokol určený pro výměnu hypertextových dokumentů ve formátu HTML.
JSON	JavaScript Object Notation, formát pro přenos serializovaných dat.
MDI	Multiple document interface, uživatelské rozhraní pro práci s více dokumenty najednou.
REST	Representational state transfer, architektonický styl.
SGML	Standard Generalized Markup Language, obecný značkovací jazyk.
SOAP	Simple Object Access Protocol, protokol pro výměnu zpráv založených na XML přes síť, hlavně pomocí HTTP.
SQL	Structured Query Language, standardizovaný dotazovací jazyk používaný pro práci s daty v relačních databázích.
WSDL	Web Service Definition Language, XML formát pro popis rozhraní webové služby.
XML	Extensible Markup Language, obecný značkovací jazyk.

Příloha B
Ukázka testovacího scénáře – Aktualizace seznamu pořadačů

Všechny testy jsou prováděny z tabletu, proti serveru `http://open.helios.eu/MobileService` a inicializaci `INI01` a pod uživatelem `testuser2`, pokud není uvedeno jinak.

B.1 Postup

1. Inicializace mobilního klienta a první přihlášení.
   - Vše proběhne bez chyb, v aplikaci jsou vidět prázdné pořadače odpovídající konfiguraci uživatele a atributu „Skrytý“ na položkách záznamu inicializace. Pořadí pořadačů zohledňuje číslo řádky položek záznamu inicializace.
2. Postupné (opakované) přepínání pořadačů (včetně pohledů – Worklist).
   - Aplikace nehlásí žádné chyby.
3. Spuštění synchronizace.
   - Synchronizace proběhne v pořádku. Aktuálně otevřený pořadač na klientovi se znovu načte a zobrazuje záznamy. Proběhne aktualizace počtu záznamů Worklistu.
4. Postupné (opakované) přepínání pořadačů (včetně pohledů – Worklist).
   - Aplikace nehlásí žádné chyby.
5. Odhlášení. (vyzkoušet i variantu s ukončením aplikace)
   - Aplikace nehlásí žádné chyby.
6. Odstranění položky pořadače „Kontaktní osoby“ z konfigurace uživatele a uložení změn.
7. Přihlášení do mobilního klienta.
   - Z panelu pořadačů zmizel pořadač „Kontaktní osoby“ a není zobrazen Worklist. Pořadí pořadačů zohledňuje číslo řádky položek záznamu inicializace.
8. Spuštění synchronizace.
   - Synchronizace proběhne v pořádku.
9. Přidání položky pořadače „Kontaktní osoby“ do konfigurace uživatele a uložení změn.
10. Spuštění synchronizace.
    - Již krátce po spuštění synchronizace se objeví v nabídce pořadač „Kontaktní osoby“ a v nabídce pohledů Worklist. Pořadí pořadačů zohledňuje číslo řádky položek záznamu inicializace. Oba jsou zatím prázdné, je možné do nich ale přepnout,

aplikace nesmí v tomto momentě vyhodit chybu. Po skončení synchronizace musí dojít ke znovunačtení přehledu, pokud je aktivní jeden z dvou výše uvedených.

11. Odstranění položky pořadače „Kontaktní osoby“ z konfigurace uživatele a uložení změn.
12. Pokud není vybrán pořadač „Kontaktní osoby“, přepneme na něj.
13. Spuštění synchronizace.
    - Již krátce po spuštění synchronizace zmizí z panelu pořadačů pořadač „Kontaktní osoby“ a klient přepne zobrazení na některý jiný pořadač. (Zkontrolujeme, zda zobrazený přehled odpovídá zvýrazněnému pořadači na panelu.) Synchronizace skončí bez problémů.
14. Postupné (opakované) přepínání pořadačů.
    - Aplikace nehlásí žádné chyby.
15. Přidání položky pořadače „Kontaktní osoby“ do konfigurace uživatele a uložení změn.
16. Spuštění synchronizace.
    - Již krátce po spuštění synchronizace se objeví v nabídce pořadač „Kontaktní osoby“ a v nabídce pohledů Worklist. Oba jsou zatím prázdné, je možné do nich ale přepnout, aplikace nesmí v tomto momentě vyhodit chybu. Po skončení synchronizace musí dojít ke znovu načtení přehledu, pokud je aktivní jeden z dvou výše uvedených.
17. Postupné (opakované) přepínání pořadačů (včetně pohledů – Worklist).
    - Aplikace nehlásí žádné chyby.
18. Opakování kroků 11. – 17. Bez odhlášení z aplikace (v kroku 12 lze například přepnout na Worklist)

B.2 Úklid

- Po skončení tohoto testu musí být v konfiguraci uživatele položka pořadače „Kontaktní osoby“.

Příloha C
Instalační příručka

Instalaci mobilního klienta je možné provést pomocí APK souboru z přiloženého DVD do mobilního zařízení a následným spuštěním tohoto souboru pomocí správce souborů na zařízení. Druhou a vhodnější variantou je instalace aplikace „HELIOS Mobile“ přímo z obchodu Play.[1])

C.1 První spuštění a konfigurace

Při prvním spuštění aplikace sama nabídne stažení demo inicializace. Po jejím stažení je možné se přihlásit na demo uživatelský účet, jehož přihlašovací údaje aplikace předvyplní. Tuto situaci ukazuje obrázek C.1.

Obrázek C.1. Přihlašovací obrazovka mobilního klienta.

Pro stažení dalších inicializací je možné z přihlašovací obrazovky podržením třetího políčka přejít do správy inicializací. Ve správě inicializací pak stačí vybrat volbu „Přidat inicializaci…“ a vyplnit přihlašovací údaje inicializace. Následující oddíl obsahuje několik volně přístupných inicializací.

C.2 Ukázkové inicializace

Všechny uvedené inicializace mají vytvořeného uživatele `testuser2` s heslem `654321`.

- **Demo inicializace (Tablety)**
  - **Adresa serveru:** `https://open.helios.eu/MobileService`
  - **Inicializace:** `INI01`
  - **Uživatelské jméno:** `inituser`
  - **Heslo:** `123456`

[1]) `https://play.google.com/store/apps/details?id=cz.aldor.heliosmobile`

- **Demo inicializace (Telefony)**
  - **Adresa serveru:** `https://open.helios.eu/MobileService`
  - **Inicializace:** `INI02`
  - **Uživatelské jméno:** `inituser`
  - **Heslo:** `123456`
- **Inicializace servisního technika**
  - **Adresa serveru:** `http://gate.aldor.cz/MobileService`
  - **Inicializace:** `TE003`
  - **Uživatelské jméno:** `inituser`
  - **Heslo:** `123456`

Příloha D
Obsah přiloženého DVD

- **Distribuce**
  - **Android** – Instalační APK balíček mobilního klienta.
  - **Server** – Přeložený projekt serveru.
- **Projekty**
  - **Android** – Zdrojové kódy klienta ve formě projektu Eclipse ADT.
  - **Server** – Zdrojové kódy serveru ve formě projektu Microsoft Visual Studio 2012.
- **Text**
  - **PDF** – Text této práce ve formátu PDF.
  - **TeX** – Text této práce ve zdrojovém formátu TEX.