VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

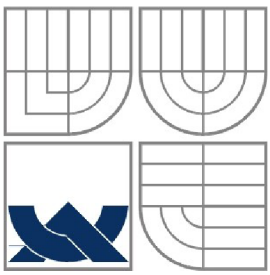
PROSTŘEDÍ FLEX PRO TVORBU INFORMAČNÍCH SYSTÉMŮ

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

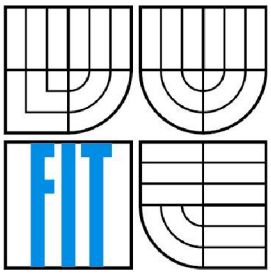
AUTOR PRÁCE
AUTHOR

PAVEL ČECH

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

PROSTŘEDÍ FLEX PRO TVORBU INFORMAČNÍCH SYSTÉMŮ

FRAMEWORK FLEX FOR CREATION INFORMATION SYSTEM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

PAVEL ČECH

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JAROSLAV RÁB

BRNO 2009

Abstrakt

Tato práce si klade za cíl prozkoumat vývojové prostředí Flex vytvořené společností Adobe a jeho využití pro tvorbu informačních systémů. Za tímto účelem byla navržena a vytvořena ukázková aplikace, která využívá základní prvky zkoumaného prostředí a názorně demonstruje jejich použití. V aplikaci je kladen důraz na zpracování dat získaných z databáze MySQL pomocí jazyka PHP, který data převádí do formátu XML.

Abstract

The purpose of this thesis is to analyse the development surrounding Flex, a program produced by the company Adobe and its use in information system creation. With this purpose in mind, a demonstration application which uses basic elements of the examined area, and clearly demonstrates their use, was suggested and produced. In the application, an emphasis is on the processing of data acquired from MySQL database with the help of language PHP, which transforms the data into XML format.

Klíčová slova

Adobe Flex Builder, Adobe AIR Adobe Flash Player, XML, PHP, MySQL, MXML, Action Script

Keywords

Adobe Flex Builder, Adobe AIR, Adobe Flash Player, XML, PHP, MySQL, MXML, Action Script

Citace

Pavel Čech: Prostředí Flex pro tvorbu informačních systémů, bakalářská práce, Brno, FIT VUT v Brně, 2009

Prostředí Flex pro tvorbu informačních systémů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Jaroslava Rába. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Pavel Čech
20. května 2009

Poděkování

Děkuji panu Ing. Jaroslavu Rábovi za rady a konzultace při tvorbě této práce.

Obsah

1 Úvod

Tento dokument si klade za cíl představit architekturu Adobe Flex a posoudit efektivitu tvorby aplikací v tomto prostředí. S tímto vývojovým prostředím jsem se nikdy dříve nesetkal, a proto se považuji za nezaujatého k objektivnímu posouzení efektivity práce v tomto prostředí.

V kapitole 2 budou vysvětleny základní pojmy, se kterými se v dokumentu pracuje. Nalezneme zde popis produktů společnosti Adobe k vývoji i spuštění aplikací. Představen bude i celkový pohled na architekturu Flex a jazyky, které se používají pro vytvoření aplikací.

Třetí část popisuje ukázkovou aplikaci. Nalezneme zde krátké představení cílů aplikace, návrh databáze, která bude použita pro ukládání dat a také jednoduché vysvětlení, jak funguje přenos dat mezi aplikací a databází. Krátce je zde popsána také grafická stránka výsledné aplikace.

Nejrozsáhlejší je čtvrtá část, která popisuje implementaci navržené aplikace. Je zde vidět jistá podobnost s druhou částí, kdy dochází k vytváření dílčích návrhů. Tato část navíc obsahuje zajímavosti zjištěné při vývoji aplikace.

V závěru dokumentu je zmíňka o bezpečnosti, návrh na rozšíření práce a celkové zhodnocení.

2 Popis používaného softwaru

2.1.1 Adobe Flex Builder 3

Je vývojový nástroj postavený na platformě Eclipse, jež si klade za cíl co nejvíce usnadnit práci na vytvářené aplikaci. Nabízí průvodce vytvořením nových projektů, možnost přepínat mezi editací zdrojového kódu a grafickým vzhledem aplikace, zobrazování možností vloženého kódu (tzv. našeptávač). Samozřejmostí je nástroj na kontrolu chyb v kódu (debugger), nástroj pro vytvoření výsledné aplikace (compiler), dále seznam dostupných komponent, Flex Navigator pro jednoduší orientaci ve struktuře zdrojových souborů, zvýraznění chyb v kódu, barevné odlišení a automatické dokončování kódu a další.

Tento nástroj je k dispozici ve free verzi, kterou je možné si na 60 dní vyzkoušet. Dále je možné získat licence pro studijní účely zdarma a to vyplněním formuláře na stránkách společnosti Adobe.[1] Licenci lze pochopitelně také zakoupit, její cena je US$249.[2]

2.1.2 Adobe Flex SDK

Společnost Adobe poskytuje možnost vytvářet aplikace pomocí prostředí Flex také zdarma, bez nutnosti kupovat si licenci k programu Flex Builder. Právě touto možností je Flex SDK. Jedná se o vývojový nástroj bez grafického zpracování. K vytvoření kódu aplikace tak lze použít jakýkoliv textový editor. Výsledný soubor pak přeložíme pomocí dříve nainstaloveného překladače přes příkazovou řádku. Zjednodušeně lze říci, že Flex SDK umí to samé jako Flex Builder. Nicméně pracovní komfort je zde na nízké úrovni, výhodou jsou ušetřené peníze za nákup licence.

2.1.3 Adobe Flash Player

Jedná se o plug-in pro webové prohlížeče, s jehož pomocí je možné zobrazovat aplikace vytvořené v prostředí Flex. Při vytváření aplikací se automaticky generuje html stránka, která zjistí, jestli uživatel má potřebný software. Pokud ano obsah začne načítat. V opačném případě nabídne odkaz na stažení.

2.1.4 Adobe AIR

Je program, který slouží k zobrazování desktopových aplikací vytvořených v prostředí Flex. Předností této technologie je maximální využití výkonu počítače. Aplikace v tomto programu vyžadují instalaci na klientův počítač.

2.2 Architektura Flex

Architektura Flex byla vyvinuta jako alternativa k současným možnostem vývoje webových aplikací, kterými jsou např. XHTML, CSS a AJAX. Cílem této architektury je nabídnout vývojářům jinou cestu k vytváření vzhledově bohatých aplikací.

Je možné vytvářet aplikace, které jsou určené k umístění na internetu nebo aplikace běžící lokálně na počítači. První typ ke svému zobrazení požaduje prohlížeč, který má naistalovaný tzv. zásuvný modul (plug-in) Adobe Flash Player. V druhém případě uživateli zpřístupní aplikaci program Adobe AIR. Oba typy aplikací tudíž nejsou závislé na platformě uživatelova počítače.

Flex používá k vytváření aplikací dva různé jazyky, z nichž první – ActionScript – je převážně používán k vytvoření logiky aplikace, je silně typovaný a objektově orientovaný. Druhý – MXML – slouží k tvorbě grafického uživatelského rozhraní (GUI) aplikace, tento jazyk patří mezi deklarativní jazyky.

2.3 Jazyk MXML

Pod zkratkou MXML se skrývá anlický název „Magic eXtensible Markup Language“[3]. Při pokusu o překlad dostaneme něco jako „magický rozšiřitelný značkovací jazyk“. Tento jazyk vychází z jazyka XML, o čemž na první pohled svědčí i podobnost názvů a z nich vyvozených zkratek. V jazyce MXML pro prostředí Flex jsou předdefinovány různé komponenty. Ty se vývojáři zobrazí v prostředí Flex Builder po přepnutí do režimu *Design* v levém dolním rohu (jedná se o výchozí nastavení). Dělí se do pěti základních skupin:

- Controls – obsahuje základní prvky grafického rozhraní, které jsou nejvíce využívané. Do této skupiny patří komponenty pro získávání informací od uživatele, jako je např. tlačítko, vstupní textové pole a zatrhávací tlačítko. Dále pak komponenty sloužící pro zobrazování dat uživateli. Pro výstup textových informací ve formátu XML skvěle slouží komponenta s názvem *DataGrid*, pro zobrazení obrázků pak *Image*.
- Layout – jak již název napovídá, nalezneme v této části prvky, které se věnují rozložení komponent po ploše výsledné aplikaci. Položky, které jsou zde obsaženy, se nazývají kontejnery proto, že se do nich vkládají jiné komponenty, a to převážně ze skupiny *Controls.* Následně můžeme nastavovat umístění všech prvků jednotně, pomocí zastřešujícího kontejneru. Nalezneme zde možnost zarovnat obsah kontejneru horizontálně, vertikálně nebo vložit mezi obsah kontejneru dělící čáru, a to opět horizontální nebo vertikální. Další možností je vytvoření lišty pro menu aplikace nebo tabulky, která bude obsahovat různé komponenty a další.

- Navigators – obsahuje položky, které nabízejí vývojáři usnadnění práce s vytvářením menu pro orietaci v aplikaci. Graficky zajímavá je komponenta *Accordion,* jež vytváří efekt, kdy tlačítka pro jednotlivé obsahy jsou umístněny na sobě a výběr položky znamená přesun všech tlačítek, která jsou nad aktuální položkou nahoru a požadovaný obsah zobrazí do prostoru mezi horními a dolními tlačítky. Tento efekt vypadá velmi pěkně, ale je nutno podotknout, že zhoršuje uživatelovu orientaci v menu, které se po každém kliknutí mění. Jiným řešením tedy může být obyčejné menu z tlačítek nebo záložkové menu. Důležitou komponentou je *ViewStack*. Můžeme do ní totiž vložit několik dalších komponent a jednoduše pak vybírat právě jednu, která se má zobrazit. Odpadá tak nutnost skrývat všechny ostatní části aplikace.
- Charts – zde nalezneme několik možností, jak zobrazit data do grafů. Jsou zde k dispozici koláčové, sloupcové, lineární, bodové a další typy.
- Custom – tato záložka slouží pro vlastní vyvojářem vytvořené komponenty. Využití nalezne především u větších projektů, kdy je vhodné narůstající MXML kód rozdělit do více částí a spravovat tak menší a přehlednější části kódu. Výhodou je možnost použití takovéto části kódu ve více aplikacích.

2.4 Propojení jazyka ActionScript s MXML

Existuje několik možností, jak propojit jazyk ActionScript se zdrojovým kódem Flex aplikace. Nyní si všechny probereme a ukážeme základní příklady.[4]

2.4.1 Použití ActionScriptu pro práci s událostmi v MXML

Nejjednodušším způsobem je vložit reakci na událost do vlastnosti některého MXML objektu. Pro ilustraci si tento případ ukážeme na tlačítku. Tlačítko má vlastnost *click*, která je aktivována poté, co na něj uživatel klikne. Do této vlastnosti můžeme dát kód v jazyce *ActionScript* a to třeba takový, který po kliknutí změní text, který je na tlačítku napsán. Tento kód, *id.label='Použité tlačítko';*, se skládá z jednoznačného identifikátoru tlačíka (*id)*, dále vlastnosti idetifikující popis (*label*) a přiřazení samotného textu.

```
<mx:Button label="Tlačítko" id="id" click="id.label='Použité tlačítko';"/>
```

Příklad 1: Použití jazyka ActionScript pro událost MXML komponenty

2.4.2 Přidání ActionScriptu pomocí tagu <mx:Script>

Druhou možností je použít speciální tag, který oznamuje, že bude následovat script. Použití samotného tagu *Script* ale nestačí, je nutné vložit celý kód v jazyce ActionScript mezi znaky *<![CDATA[a]]>*, jak ukazuje následující příklad.

```
<mx:Script> <![CDATA[ zde je ActionScript ]]></mx:Script>
```

Příklad 2: Vložení jazyka ActionScript do jazyka MXML

Tím se zajistí, že při překladu kódu kompilátor nebude obsah tagu *Script* vyhodnocovat jako XML, ale jako ActionScript. Je tak možné běžně používat ostré závorky (< >) např. při porovnávání, aniž by byli považování za XML tagy.

2.4.3 Vložení externího ActionScript souboru

Tuto možnost lze využít u rozsáhlejších projektů pro zvýšení přehlednosti kódu a jeho lepší udržovatelnost, obdobně jako tomu je u možnosti vytvoření vlastních komponent v jazyce MXML. Prvním krokem je vytvoření samoného souboru, ve kterém bude obsažen kód jakyza ActionScript. Tento soubor se uloží s příponou *.as* a do Flex aplikace se pak vloží pomocí atributu *source* tagu *Script* (příklad 3) nebo pomocí příkazu *include* (příklad 4).

```
<mx:Script source="cesta_k_souboru/nazev.as">
```

Příklad 3: Vložení externího souboru pomocí atribudu *source*

```
<mx:Script> <![CDATA[ include „cesta_k_souboru/nazev.as ]]></mx:Script>
```

Příklad 4: Vložení externího souboru pomocí příkazu *include*

2.4.4 Načtení balíků a tříd ActionScript

Pro načtení balíků, které obsahují třídy, se kterými chceme pracovat, se stejně jako v předchozím případě používá příkaz *include* doplněný příslušným názvem.

Načítání tříd se hojně používá při práci v ActionScriptu, a to pro načtení již definovaných tříd, s jejichž pomocí je práce jednoduší, např. pro práci s kurzorem načteme třídu *CursorManager* a to tímto způsobem:

```
import mx.managers.CursorManager;
```

Příklad 5: Načtení třídy CursorManager z balíku mx.managers

3 Návrh ukázkové aplikace

3.1 Popis aplikace

V prostředí Flex jsem se rozhodl vytvořit jednoduchou aplikaci, která bude sloužit pro uživatele z řad drobných soukromých zemědělců. Zemědělec používající tuto aplikaci si může ukládat základní informace o svých zaměstnancích, strojích, pěstovaných plodinách, používaných hnojivech a postřicích. Důležitou částí aplikace je možnost ukládat podrobnější informace o činnostech prováděných při pěstování jednotlivých plodin a zadávat objednávky od zákazníků.

Vytvořený program ukládá uživatelem zadaná data do databáze. Při komunikaci s databází jsou používány skripty naprogramované v jazyce PHP. Logika a vzhled aplikace je pak vytvořen v prostředí Flex pomocí nástroje Flex Builder.

3.2 Ukládání dat

Pamětí celé aplikace se dá nazvat databáze. Důležitou vlastností vytvořené databáze je nutnost podpory transakcí. V aplikaci se totiž vyskytují případy, kdy uživatel pracuje s daty z více tabulek a jejich editace musí být z pohledu databáze nedělitelná.

Pokud totiž uživatel chce uložit informaci o nové objednávce, je nutné, aby se nejen uložily informace nejen o samotné objednávce, ale také o tom, pro koho objednávka je a co za položky objednávka obsahuje. Tyto informace se ale nacházejí ve více tabulkách, je proto nezbytné provést buď všechny tyto změny, nebo žádnou a právě k tomu slouží transakce.

3.2.1 Návrh databáze

Zemedělec ukládá základní identifikační údaje o zaměstnancích, kteří pro něj pracují. Mezi tyto informace patří jméno a příjmení, město, kde zaměstnanec bydlí, hodinová mzda (povinné údaje), telefon, ulice a číslo ulice (volitelné údaje).

U hnojiv se ukládá název a cena za tunu. V případě postřiků je tomu podobně. Opět se zaznamenává název, stejně tak cena, ale tentokrát je cena za kilogram. Navíc je potřeba uchovat informaci o tzv. ochranné lhůtě. To je doba, která musí uplynout mezi dobou použití postřiku a dobou konzumace ošetřované plodiny.

Další informace se týkají strojů a činností. Každý stroj má svůj název, který ho prakticky jednoznačně identifikuje. Informace o spotřebě paliva daného stroje se v databázi neukládájí, protože stanovit průměrnou spotřebu nelze. Dochází zde totiž k velkým rozdílům podle způsobu použití,

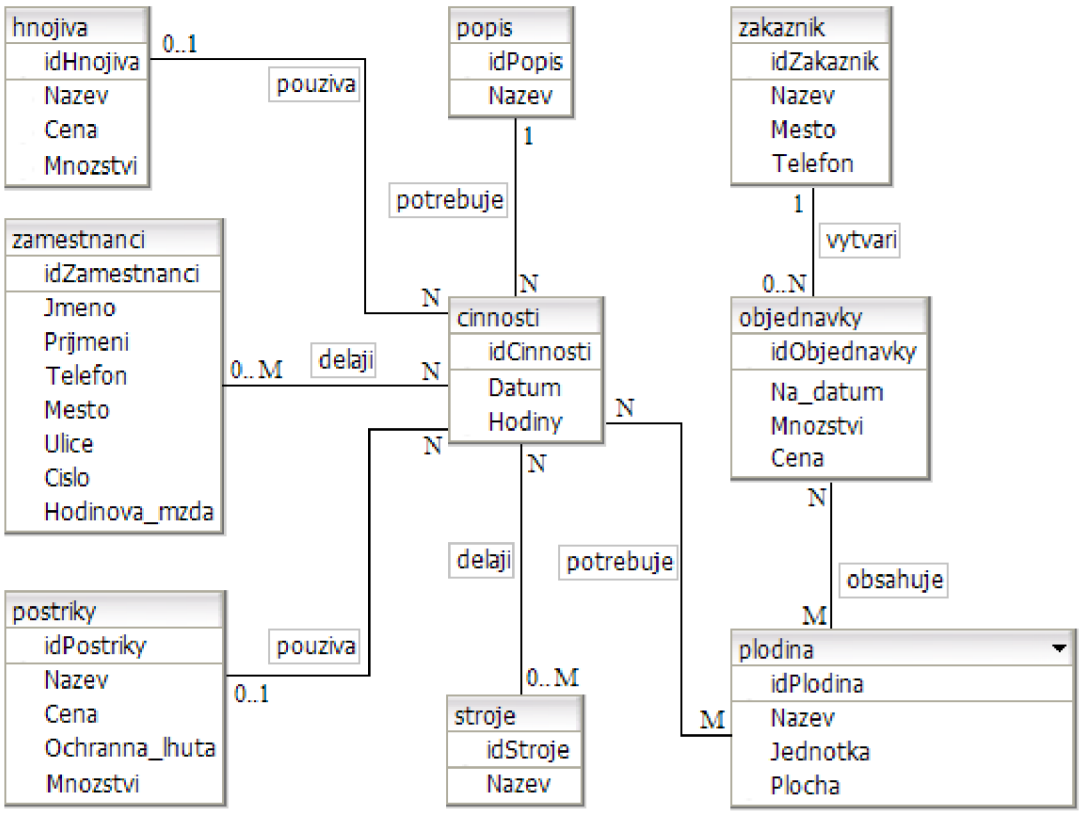
navíc některé stoje ani žádnou spotřebu nemají (vlečka, zavlačovací cívka). U činností je více než název důležité datum vykonání a jak dlouho daná činnost trvala.

K posledním ukládaným datům patří ta, která se týkají jednotlivých plodin. Což je název, plocha v hektarech, na které je plodina pěstována a jednotka prodeje, která může být buď kus, nebo častěji kilogram.

Informace týkající se zákazníků jsou stručné, protože v menší zemědělské prvovýrobě se zemědělec setkává s menšími obchodníky a nemusí tudíž vystavovat složité faktury. Stačí tedy název nebo jméno, město a telefon. Se zákazníky přímo souvisí objednávky. Každý zákazník může mít samozřejmě několik objednávek a každá objednávka obsahuje více plodin, kde u každé plodiny je důležité množství a cena za jednotku.

3.2.2 ER diagram

ER diagram je dalším krokem při návrhu výsledné databáze. Byl vytvořen podle slovního popisu požadavků na výslednou databázi. Označuje vztahy mezi tabulkami, položky tabulek a kardinalitu vztahů.



Obrázek 1: ER diagram navrhované databáze

3.2.3 Model databáze

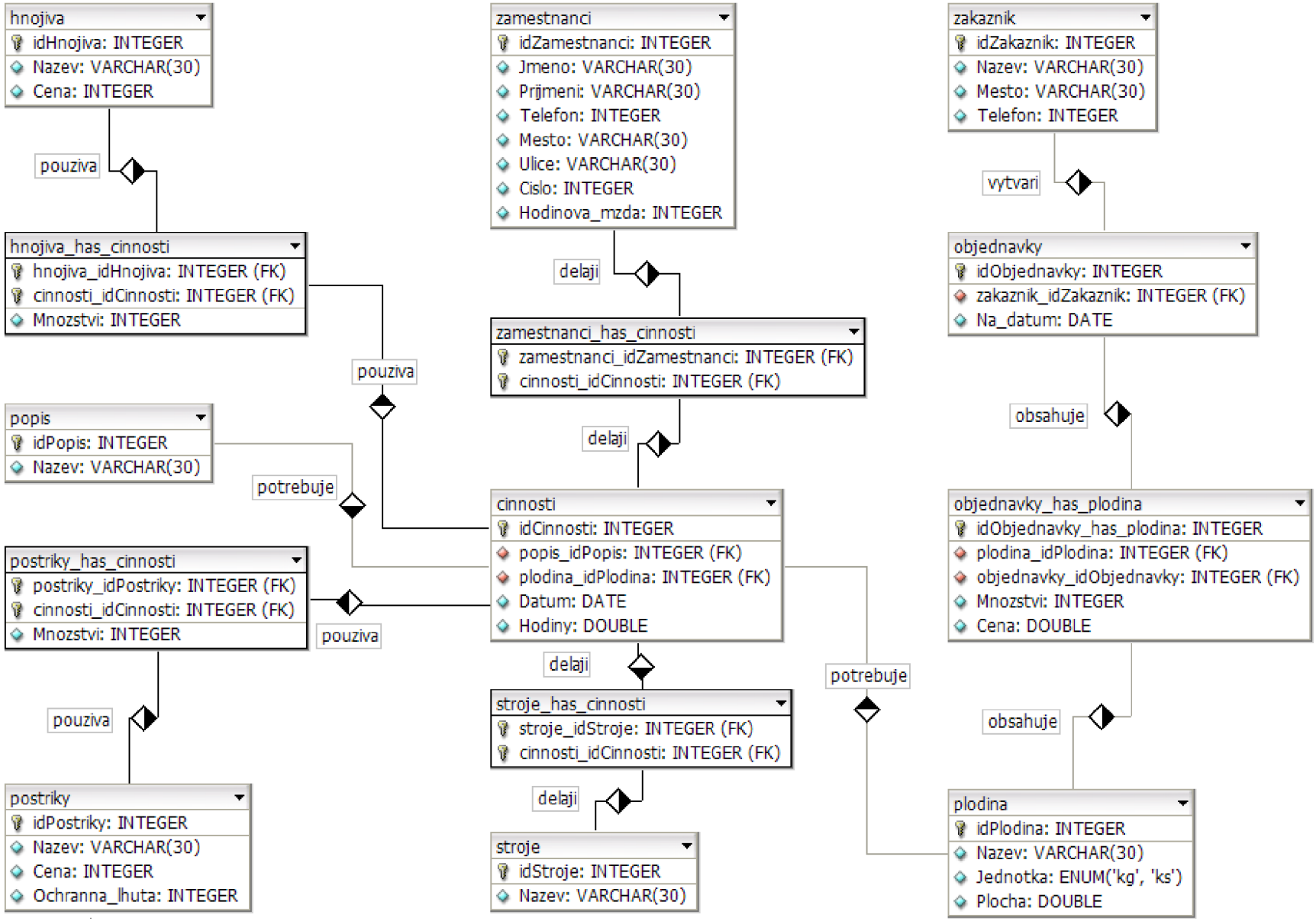
Obrázek 6 zobrazuje výsledný model používané databáze. Tento obrázek byl vytvořen v programu DBDesigner 4, který pro označování vztahů mezi entitami používá zvláštní značení. Vysvětlení jednotlivých pojmů obstarávají následující obrázky 2 – 5.

Obrázek 2: Vztah 1:N jak jej znázorňuje program DBDesigner

Obrázek 3: Znak pro primární klíč

Obrázek 4: Znak pro cizí klíč

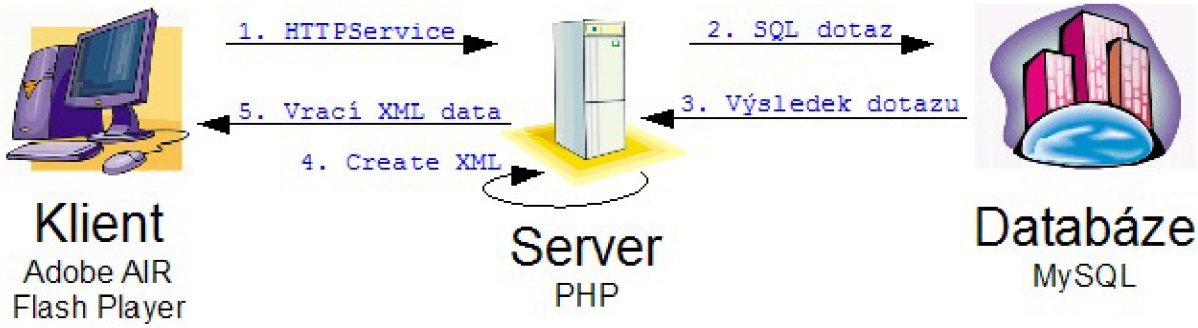
Obrázek 5: Znak pro atribut tabulky



Obrázek 6: Model databáze

3.3 Komunikace

Pro komunikaci mezi výslednou ukázkovou aplikací a daty uloženými v databázi MySQL slouží PHP server. Klient na základě činnosti uživatele vygeneruje požadavek na data, čímž aktivuje odpovídající skript na straně serveru a ten následně provede dotaz nad databází. Databáze skriptu vrací výsledek dotazu, který obsahuje požadovaná data. Skript následně převede získaná data do formátu XML a výsledek odešle zpět klientovi. Proce komunikace graficky znázorňuje následující obrázek.



Obrázek 7: Ukázka komunikace [5] [6] [7]

3.4 Grafické uživatelské rozhraní

Aplikace si klade za cíl vytvořit uživatelsky příjemné a jednoduché prostředí. Tento požadavek vychází z praxe, kde přehnané množství efektů a propracovaná grafická stránka jde proti jednoduchosti ovládání, kterou uživatel v oblastni zemědělství upřednostňuje. Zvolený přístup tedy neukazuje maximální možnost využití efektů zkoumaného prostředí, což také není cílem práce.

4 Implementace aplikace

K vytvoření výsledné aplikace bylo zapotřebí použití celkem čtyř jazyků, a to těch, jež byly již dříve zmíněny. PHP pro obsluhu a práci s daty, SQL pro uložení dat, MXML pro vytvoření grafického uživatelského prostředí a ActionScript pro samotnou logiku aplikace.

4.1 Vytvoření databáze

Tato část byla z celé práce nejjednoduší, protože stačilo převést model databáze (viz. obrázek 6) do SQL kódu a tento kód pak vykonat na databázovém serveru. Pro účel aplikace na webu byla tato databáze vytvořena na databázovém serveru www.webzdarma.cz.

4.2 Obslužné PHP skripty

4.2.1 Základní informace

Jedná se o soubory, které zajišťují propojení aplikace s daty v databázi. Tyto skripty využívají přenášených proměnných z Flex aplikace k tomu, aby zjistily, jaká data uživatel právě požaduje. Všechny proměnné se přenášejí metodou POST a přístup k nim se pak provádí pomocí kódu $_POST["název"]. V proměnné *tabulkaNazev* je uložena informace s jakou tabulkou se má právě pracovat. Další proměnné poskytují skriptům nezbytná data, které se mají uložit, změnit, případně vymazat z databáze. Za základní soubor můžeme považovat soubor s názvem *pripojeniDatabaze.php*, který navazuje spojení s databází. Služeb tohoto skriptu pak využívají všechny ostatní skripty, které lze rozdělit do skupin podle toho, jaké úkony s daty provádějí.

4.2.2 Zobrazení dat

Soubor *vypis.php* zobrazí uživateli požadovaná data. Po provedení dotazu na příslušnou tabulku skript převede získaná data do formátu XML. Tento převod se provádí jednoduše, a to pomocí proměnné, do které se v cyklu ukládají postupně jak formátovací znaky jazyka XML, tak samotné textové informace získaný z databáze. Takto vytvořený obsah zmiňované proměnné je pak jednoduše vytisknut, čímž je odeslán zpět do aplikace Flex. Stejným způsobem pak vytvářejí XML kód ostatní skripty.

4.2.3 Editace dat

O vymazání dat se stará soubor *vymaz.php*. Který řádek tabulky se má vymazat určuje jednoznačný identifikátor zadaný uživatelem. Při mazání objednávky nebo činnosti prováděné při pěstování nastává situace, kdy jsou data uložena ve více tabulkách. Je proto nutné vymazat buď data ze všech tabulek, kterých se tato změna týká, nebo z žádné, a obnovit tak původní stav tabulek. Toto se provádí pomocí transakcí. Výsledek každého příkazu, který se po databázi požaduje, je uložen do samostatné proměnné. Po skončení všech příkazů se zjišťuje, jestli nedošlo u některého k chybě. Pokud ano, změny jsou zrušeny příkazem *ROLLBACK*, v opačném případě se transakce uzavírá příkazem *COMMIT*.[8]

Vkládání nových informací do databáze obstarává soubor *pridej.php*. Skript řeší i situaci, že některé hodnoty tabulky jsou nepovinné, tudíž je musí být schopen vynechat a umožnit tak do databáze uložit prázdnou hodnotu. Obdobně jako při vymazávání, tak i při vkládání dat o objednávkách nebo o činnosti pěstování, skript používá transakci pro zajištění správného provední požadované změny.

Úpravu uložených dat provádí soubor *oprav.php.* Ten nahradí všechny atributy řádku tabulky určeného identifikátorem za nové hodnoty.

4.2.4 Skripty pro práci s více tabulkami

Pro práci s daty o pěstování určité plodiny je používán soubor *pestovani.php*. Vrací seznam všech vykonávaných činností pro konkrétní plodinu. Následně pak může poskytovat i podrobnější informace o vybrané činnosti, jako jsou jména zaměstnanců a strojů, jež vykonávali danou činnost. Dále pak názvy a mnozšství použitých hnojiv a postřiků spojených s touto činností.

Podobně jako při pěstování, i při vytváření objednávek je nutné pracovat s více tabulkami, a proto i objednávky mají svůj vlastní soubor s názvem *objenavky.php,* který vrací všechny položky vybrané objednávky, informaci o jednotce, množství a ceně každé položky.

4.3 Grafické uživatelské rozhraní

Aplikace se skládá ze dvou základních částí. Můžeme je nazvat *menu* a *datová oblast*. Menu bylo vytvořeno jako jednoúrovňové, protože není rozsáhlé, a tudíž použití více úrovní není nezbytně nutné. Navíc by zbytečně komplikovalo orientaci uživatele v aplikaci. Pro každé tlačítko z menu aplikace existuje oblast, která se v reakci na kliknutí zobrazí. V této oblasti se jednak uživateli zobrazují data získané z databáze, ale také se zde nabízí uživateli možnost pomocí vstupních textových polí data do databáze uložit. Položky menu jsou řazeny za sebou a to tak, že prvních sedm položek odkazuje na

práci se základními daty. To jsou informace o zaměstnancích, hnojivech, postřicích, zákaznících, strojích, plodinách a činnostech. Položky ve většině těchto částech musí uživatel vyplnit, aby mělo vůbec smysl pracovat s dalšími dvěma položkami, jimiž jsou *pěstování* a *objednávky*. V této části programu totiž uživatel spojuje dílčí informace, které již uložil, přidává k nim aktuální informace a dohromady vytváří nové položky v informačním systému.

4.3.1 Hlavní menu aplikace

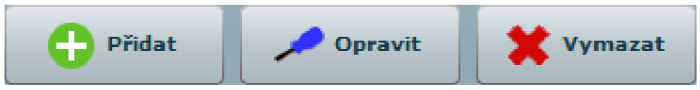
Navigaci v aplikaci zajišťuje sada tlačítek, jež byly vytvořeny pomocí komponenty *<mx:Button>*. Každé tlačítko má svou vlastní ikonu, která se snaží popisovat oblast dat, kterou vybrané tlačítko zpřístupňuje. U vkládaných obrázků nabízí Flex dvě možnosti zobrazování. Pokud je použit příkaz *@Embed* [9], znamená to, že Flex načte obrázek při vytváření výsledné aplikace. V opačném případě dojde k načtení obrázku až při běhu aplikace, což může způsobit, že uživatel musí čekat na zobrazení obrázku. Cesta k obrázku může být jak absolutní, tak relativní. Druhou možnost ukazuje příklad 6.

```
<mx:Button icon="@Embed('obr.png')" label="Tlačítko"/>
```

Příklad 6: Vkládání obrázků do tlačítek

4.3.2 Menu pro editaci údajů

Pro přidávání, opravování a mazání dat slouží vedlejší menu, které obsahuje tři tlačítka. Ty slouží k dílčím úkonům editace a jsou opět doplněny ikonami, které pomahájí uživateli lépe se orientovat, co se s daným tlačítkem provádí. Toto menu je pro všechny části aplikace vždy na stejném místě. [10]



Obrázek 8: Ukázka editačního menu

4.3.3 Oblast pro zobrazení dat

Pro zobrazování dat v aplikaci slouží MXML komponenta *DataGrid* [11]. Ta funguje tak, že přes svojí vlastnost *dataProvider* nastaví zdroj XML dat pro zobrazení. Zdroj dat musí být uveden ve složených závorkách, čímž se odliší od normálního textu. U zdroje musíme také uvést rodičovský uzel, případně uzly, které vedou k požadovaným datům. Pokud očekáváme jednoduchá data, o kterých víme, že jejich struktura nám vyhovuje, stačí použít pouze jeden tag. V tom případě aplikace zobrazí všechna data, a to tak, že podle informace ve vlastnosti *dataProvider* rozdělí zdrojový soubor na jednotlivé potomky, kteří se stanou řádky ve výsledné tabulce. Každý takový potomek má pak své potomky, což jsou sloupce v daném řádku. Pokud neurčíme jinak, stanou se názvy XML tagů těchto

potomků názvy sloupců výsledné tabulky. V příkladu 7 tedy očekáváme data v proměnné s názvem *zdroj*. Rodičovský uzel těchto dat nese jméno *polozky* a jednotlivé položky se pak jmenují *polozka*. Pokud bude položka obsahovat například hodnoty datum v XML tagu *kdy* a adresu v tagu *kde*, bude tabulka obsahovat dva sloupce, jejichž názvy budou „kdy“ a „kde“.

```
<mx:DataGrid id="id" dataProvider="{zdroj.polozky.polozka}" />
```

Příklad 7: Zobrazení XML dat v DataGrig komponentě

Většinou je samozřejmě potřeba zobrazovat složitější data, nastavovat názvy sloupců, případně některé sloupce vynechat. Komponenta *dataGrid* nabízí i všechny tyto možnosti. Pokud chceme přidat sloupec do výsledné tabulky, poslouží nám k tomu komponenta *<mx:DataGridColumn>*. Pomocí vlastnosti *headerText* nastavíme text, který se zobrazí v záhlaví sloupce. Vlastnost *dataField* nám určuje, který XML uzel ze zdrojových dat se bude v tomto sloupci zobrazovat. Nový sloupec, tak jako všechny ostatní sloupce, musí být uvnitř XML tagů *<mx:columns> </mx:columns>*.

Výhodou této komponenty je, že má již v sobě implementovanou podporu řazení obsahu jednotlivých sloupců. Po kliknutí na záhlaví sloupce se tak všechny data seřadí sestupně nebo vzestupně, a to podle obsahu vybraného sloupce. Každý sloupec má nastavenou svoji výchozí hodnotu šířky, ale ta se dá změnit, takže uživatel může myší měnit aktuální šířku každého sloupce, což využije při zobrazování dat, která jsou extrémně dlouhá.

ID	Název	Kč/tuna
1	Ledek	2500
2	NPK	3100
3	Draslík	1200

Obrázek 9: Ukázka komponenty *dataGrid*

4.3.4 Oblast pro editaci dat

Bez této části by aplikace ztrácela svůj význam. K zadávání dat je využita komponenta *textInput*, která již podle názvu naznačuje, že slouží ke vkládání textu. Uživateli se pro vybranou část aplikace zobrazí všechny možné hodnoty, jež může zadat. U každé položky je pak pomocí komponenty *Label* vložen popisek, k čemu je příslušné textové pole určeno.

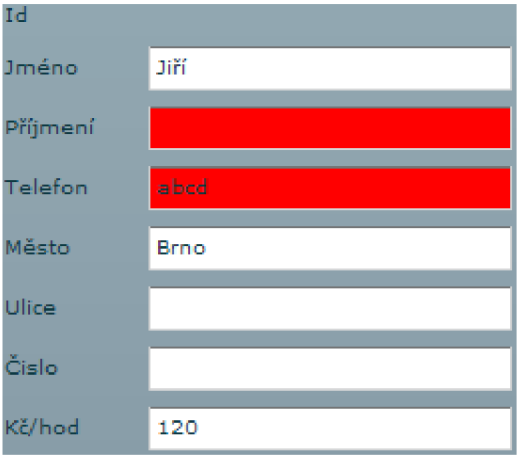
Parametry komponenty *textInput* jsou *id,* díky němuž je možné komponentu jednoznačně identifikovat v rámci celého zdrojového kódu, dále šířka, výška a velikost písma. V ukázkové aplikaci je ale nejdůležitějším atributem *text*. Z nějž se na jedné straně získávají uživatelem zadané hodnoty, ale na straně druhé se pomocí něho také nastavuje uživatelem vybraná hodnota z komponenty *dataGrid*. Tohoto efektu se využívá při upravách již existujících údajů, kdy uživatel

zadal data do systému a následně je chce upravit. V tom případě stačí kliknout na řádek, který chce upravit a jeho obsah se načte do vstupních polí. Toto je umožněno vlastností *selectedItem* komponenty *dataGrid* (příklad 8). Zde pak stačí opravit nalezenou chybu a v editačním menu kliknout na tlačítko *OPRAVIT*.

```
<mx:Text text="{idDataGrid.selectedItem.idSloupce}"/>
```

Příklad 8: Zobrazení obsahu sloupce z vybraného řádku komponenty *dataGrid*

Další často používanou vlastností vstupního textového pole, je jeho barva pozadí. Ta slouží ke zvýraznění chyb, kterých se uživatel při práci s aplikací dopustil. Pokud totiž vynechá povinný údaj nebo místo očekávaného čísla vloží řetězec znaků, upozorní ho na špatný formát dat chybové hlášení. Následně se mu pak barevně odliší chybná místa, jež mu mají usnadnit opravu chybných údajů.



Obrázek 10: Vyznačení chyb ve vstupních polích

4.3.5 Oblast práce s více tabulkami

V těchto částech programu již nemůže uživatel zadávat všechny hodnoty pomocí textových polí, protože se zde pracuje s obsahem několika různých tabulek. Kontrolování, jestli uživatel zadal hodnotu, která v dané tabulce opravdu je, by bylo zbytečně složité. Na řadu tak přichází další z řady komponent, kterou je *ComboBox* [12]. Do této komponenty se načtou všechny možné hodnoty a na uživateli pak je, aby si některou z těchto hodnot vybral. Načítání dat se děje stejně jako u komponenty *dataGrid*, tedy přes vlastnost *dataProvider,* do které se vloží jednotlivé řádky XML dat a pomocí vlastnosti *labelField* je identifikován sloupec, jehož hodnoty se mají zobrazit. Použitím komponenty *ComboBox* se zamezí možnosti vložit hodnotu, která v dané tabulce není. Pokud uživatel nechce uložit žádnou z nabízených hodnot, použije připravené tlačítko na zkrytí dané položky. Toto tlačítko je typu *CheckBox* a pro zkrytí a zobrazení využívá své vlastnosti *change*. Při vyvolání této vlastnosti se změní aktuální vlastnost *visible* zkrývaných komponent.

Poprvé v celé aplikaci je zde také použita komponenta *DateChooser* – kalendář. Vítanou možností je změnit označení dnů a měsíců. Z implicitně anglického kalendáře se tak jednoduše stane kalendář český. Tato změna se provádí přes vlastnost *dayNames* a *monthNames,* do nichž stačí přiřadit pole názvů oddělené čárkami – viz. příklad 9. Zvláštností tohoto kalenáře je, že jako první sloupec není pondělí, ale neděle. Indexování dnů je počítáno od nuly, tudíž při práci s datumem se musí jeden den přičítat. Další užitečnou vlastností je možnost zobrazovat jen některé dny v týdnu pomocí vlastnosti *disableDays* a omezení rozsahu kalendáře. Lze tak např. nastavit, aby se zobrazovali jsen pracovní dny v aktuálním roce.

```
<mx:DateChooser dayNames="[Ne, Po,...]" monthNames="[Leden, Únor,...]"/>
```

Příklad 9: Změn názvů dnů a měsíců v kalendáři

Poslední zajímavostní v této části je postupné zobrazování podrobnějších informací. Podíváme-li se na část *pěstování,* pak na začátku je tabulka, která obsahuje všechny plodiny. Po kliknutí na některou z těchto plodin se zobrazí v další tabulce seznam všech vykonávaných činností na této plodině. Podobným způsobem se pak zobrazují v dalších čtyřech tabulkách podrobnosti o vybranné činnosti.

4.4 Aplikační logika

Mozkem celé ukázkové aplikace je program v jazyce ActionScript. Následující text tak bude pojednávat o zajímavostech spojených s vývojem této aplikace.

4.4.1 Přenos dat pomocí HTTPService

K propojení aplikace s obslužnými php skripty slouží komponenta s názvem *HTTPService* [13]. Ikdyž se jedná o komponentu jazyka MXML, tak převážná část práce s touto komponentou se provádí v jazyce ActionScript, proto je tato zmiňována až zde.

Vlastností *id* nastavíme jednoznačný identifikátor, díky jemuž pak budeme moci přistupovat k datům, které aplikace po vyslání dotazu na php skript získala. Atribut *method* nám podobně jako v jazyce PHP říká, jak se budou přenášet proměnné. Užitečné vlastnosti jsou také *result* a *fault*. V těchto vlastnostech se volají funkce *vporadku* a *chyba*. První se provede, pokud přenos proběhl úspěšně, druhá v opačném případě. Funkci *chyba* je navíc jako parametr předávána událost, která chybu způsobila.

Parametrem nezbytným pro správný přenos je cíl, kam se mají data přenést. Ten se vkládá do vlastnosti *url* a to jako řetězec. K nastavení přenášených dat se v aplikaci používá konstrukce, kdy je k použité HTTPService komponentě připojena proměnná s názvem a hodnotou. Připojení proměnné zajišťuje vlastnost *request.*

```
IdHTTPService.request.nazev_promenne = obsah_promenne;
```

Příklad 10: Nastavení přenášených dat

Samotné odeslání dat se pak provede jednoduchým příkazem *nazev_HTTPService.send()*. Při přenášení dat je také vhodné naznačit uživateli, že se něco děje. Jinak při delším čekání na straně serveru může uživatel nabýt dojmu, že aplikace přestala fungovat. Proto při začátku přenosu nastavíme kurzor myši na hodiny a po skončení přenosu zase zobrazíme původní kurzor, což nám ukazuje následující příklad:

```
CursorManager.setBusyCursor();
CursorManager.removeBusyCursor();
```

Příklad 11: Práce s kurzorem

Výsledek přenosu, který komponenta HTTPService obdrží od PHP scriptu, zobrazí do kompoenty *dataGrid*. Vypsání dat uživateli zařídíme voláním funkce *lastResult* – zobrazíme poslední výsledek komunikace. Tento přístup ve velké části aplikace vyhovuje. V případě, kdy ale chceme zobrazit data z více datových přenosů, musíme použít také odpovídající počet *HTTPService* komponent. K tomuto dochází např. při práci s objednávkami. V jedné tabulce se nám zobrazí informace o zákaznících. Po kliknutí na řádek se zákazníkem se v druhé tabulce objeví výpis všech jeho objednávek. Pokud by byla použita pro tento druhý dotaz stejná *HTTPService* komponenta, pak by se informace o objednávkách vybraného zákazníka zobrazily, ale zmizel by obsah tabulky zákazník. Došlo by tedy k situaci, že by se obě tabulky snažili zobrazit stejná data a jen podle obsahu XML struktury dat by se to jedné nebo druhé střídavě dařilo. Proto je nutné tam, kde zobrazujeme data z více přenosů používat také více *HTTPService* komponent.

4.4.2 Kontrola vstupních dat

Je nutné hlídat všechny možné vstupy aplikace, aby nedocházelo ke zbytečnému odesílání špatných dat na stranu serveru. Např. pokud uživatel nezadá povinnou hodnotu, je zbytečné, aby se prováděl přenos na server, zde se zjistila chyba, poslala se odpověď a následně až pak aplikace ohlásila problém.

Při ověřování dat zadaných uživatelem dochází k využívání toho, že každý prvek z grafického uživatelského rozhraní má svého rodiče (základní vlastnost jazyka XML). Konkrétně všechna vstupní textová pole z obrázku 10 jsou obsažena v jedné komponentě typu *Canvas*. Stačí tedy získat všechny potomky této komponenty a pak zjistit, jestli daný potomek obsahuje právě taková data, jaká se od něj očekávají. Zde ale nastává drobný problém, protože v rodičovské komponentě je mimo vstupních textových polí také několik tlačítek a objektů typu *dataGrid*. Je proto nutné při procházení potomků zjišťovat, jakého typu daný potomek je a ověřovat informace pouze u potomků typu *textInput*. K ověření, jestli komponenta je určitého typu, slouží vyhrazené slovo *is*. Jestli se jedná o typ *textInput* můžeme zjistit takto:

```
if (nazev_komponenty is TextInput)
```

Příklad 12: Ověření typu komponenty

Nyní jsme ve fázi, kdy jsme úspěšně oddělili komponenty typu *textInput* od všech ostatních. Nastává ale další problém, protože některá vstupní pole mohou být prázdná, jiná prázdná být nesmí a další mohou obsahovat pouze číslice. Začněme od konce, tedy od číslic. Každé vstupní textové pole má svůj jednoznačný identifikátor. Pokud v tomto textovém poli očekáváme číslo, v identifikátoru nalezneme speciální řetězec *SUPER*. Všechna textové vstupní pole, ve kterých očekáváme číslo, mají ve svém identifikátoru řetězec *SUPER*. Nyní se tedy můžeme pomocí funkce *indexOf* [14] pokusit vyhledat řetězec *SUPER* v hodnotě identifikátoru (příkaz vrací zápornou hodnotu. V případě nalezení hledané části identifikátoru se tedy má v potomku nacházet číselná hodnota, jinak se jedná o řetězec.

```
if(komponenta.id.indexOf("SUPER") >= 0)
```

Příklad 13: Vyhledání retězece *SUPER* v identifikátoru komponenty

Máme tedy rozdělené komponenty na dvě skupiny, první musí obsahovat čísla, druhá řetězce. Následuje kontrola, jestli může být textové pole prázdné. Zde se postupuje stejně jako při zjišťování, jestli se jedná o číslo. Použijeme tedy opět funkci *indexOf*, tentokrát pro vyhledání řetězce *VYNECH* v idefikátoru komponenty. V případě, že je řetězec nalezen, může být komponenta prázdná, jinak musíme uživatele upozornit, že tento údaj je povinný. Budeme-li chtít použít textové pole, které má obsahovat číslo a jeho vyplnění je nepovinné, pak jeho idefikátor musí obsahovat řetězece *VYNECH* a *SUPER*, tedy např. *zadejCisloVYNECHSUPER*.

Jiným možným přístupem k rozlišení, co které vstupní pole má obsahovat, může být vložení těchto polí do rodičovských komponent podle druhu obsahu. Všechna textová pole, která očekávají řetězec, by tak byla ve vlastní komponentě, ta jenž očekávají na svém vstupu číslo, by byla v další

komponentě. Jiná komponenta by určovala vstupní pole s textem, které může být prázdné a další pole očekávající číslo, které může být také prázdné. Výhodou je, že již není nutné zjišťovat u každého potomka, co má obsahovat.

Další z řady kontrol je věnována délce uživatelem zadaného vstupu. K tomu je použita funkce *length*. U řetězce je nastaveno omezení délky na maximálně 30 znaků, u čísla pak na 9 znaků. Číslo je zatím pouze řetězec číslic. Až pokud je i jeho délka v pořádku, přistoupíme k převedení řetězce na číslo pomocí funkce *Number()*. Ještě vysvětlení, proč je číslo omezeno délkou 9 znaků. Pro uložení čísel v databázi je použit datový typ *unsigned integer*, jeho rozsah je 0-4 294 967 295 [4], což je 10 číslic, které ale mají omezení právě maximální hodnotou. Pokud tedy použijeme jen devět číslic, můžeme uložit sekveci devíti libovolných číslic. Telefonní číslo, pro které se také tento typ používá, má právě 9 číslic, u ostatních položek, jako je množství, cena nebo hodinová mzda, je tento rozsah dostačující.

Poslední věcí, kterou aplikace musí hlídat, je ukládání informací o pěstování. Zde již uživatel z větší části pracuje s komponentami typu *ComboBox*, u kterých není potřeba kontrolovat, jestli dodržel formát vstupních dat. Ale je nutné hlídat, aby uživatel nezadal např. dva stejné zaměstnance na jednu činnost. Opět se při kontrole využívá toho, že *ComboBox* je potomkem komponenty *Canvas*. Ze všech potomků vybereme prvního a jeho obsah porovnáme s obsahem ostatních potomků. Postup se opakuje až dokud neporovnáme všechny potomky navzájem.

4.4.3 Práce s chybami

Pokud při některé kontrole aplikace narazí na chybu, uživateli to oznámí chybové hlášení (viz. příklad 12) a následné barevné zvýraznění místa, kde byla chyba nalezena. Aplikace funguje tak, že ověří všechny problémové místa a zjistí tak všechny chyby, kterých se uživatel dopustil. Až po této důkladné kontrole vypíše všechny chyby najednou.

```
Alert.show(nazev, "Popisek chybového okna");
```

Příklad 14: Vypsání chyb, které jsou obsaženy v proměnné *nazev*

Důležité je při nalezení chyby zachovat obsah všech vstupních polí. Nenutíme tím uživatele k opětovnému vkládání správných údajů a zároveň mu ukazujeme jeho konkrétní chyby. Pokud chyba není nalezena, pak je obsah textových polí vymazán a tím uživateli naznačujeme, že aplikace něco vykonala (např. uložení). Zároveň je nutné hlídat stav, kdy uživatel nejprve zadá špatné informace, ty mu aplikace vyznačí, on je opraví a provede např. uložení. Nyní je nutné nejenom vymazat obsah textových polí, ale také jejich barvu. K obojímu se využije toho, že všechna textové pole jsou v aplikaci něčí potomci, a tak stačí tyto potomky pouze projít a nastavit jim výchozí hodnoty.

4.4.4 Navigace v aplikaci

Správné fungování aplikace zajišťuje několik funkcí, které jsou volány na základně uživatelem stisknutých tlačítek. Pokud uživatel vybere položku z hlavního menu, zavolá tím funkci *nacti()*, která zobrazí příslušnou část komponenty *ViewStack* [15]. Zobrazování pomocí této komponenty je velmi jednoduché. Pokud by jsme se o to samé pokusili pomocí komponent *Canvas*, bylo by nutné pro všechny komponenty nastavit atribut *visible* na hodnotu *false* a pouze u žádané komponenty zadat hodnotu *true*. Toto za nás udělá *ViewStack* sám. Každému jeho potomku nastavíme identifikátor a pokud chceme zobrazit některého z potomků, stačí jej jednoduše vybrat (příklad 15). V ukázkové aplikaci komponenta *ViewStack* obsahuje pro každé tlačítko z menu jednu komponentu *Canvas*, která se zobrazí právě při stisknutí tohoto tlačítka.

```
id_ViewStack.selectedChild = id_Canvas;
```

Příklad 15: Výběr potomka, který má být zobrazen

4.4.5 Směrování dotazů na PHP skripty

Jedná se o část, která zabírá nejrozsáhlejší část zrojové kódu v jazyce ActionScript. Aplikace musí rozeznat s jakou tabulkou uživatel právě pracuje a co za úpravy s jejím obsahem chce provádět. Všechna tlačítka z editačního menu (viz. obrázek 8) po svém stisknutí volají funci *edituj()*. Tato funkce má dva parametry, první je název souboru, který obsahuje volaný PHP skript. Druhým parametrem je název tabulky, se kterou se bude pracovat. První parametr se použije pro vytvoření cílové adresy. Druhý pak slouží k rozlišení, jaké proměnné se mají nastavit. Právě kvůli rozdílným proměnným, které si práce s každou tabulkou vyžaduje, velmi narůstá kód v této části programu. Je totiž nutné nejdříve identifikovat vybranou tabulku, a pak nastavit právě ty proměnné, které očekává.

Možné zjednodušení by se dalo udělat tím, že nebudeme přenášet názvy a hodnoty jednotlivých proměnných, ale hodnoty uložíme do pole a to jako jedinou proměnnou přenese na stranu PHP skriptu. Toto řešení lze ale použít pouze pokud máme pevně dáno, jaká data a v jakém pořadí v poli jsou. Bohužel tuto podmínku nelze zajistit např. při práci v sekci *pěstování*. Zde totiž uživatel má možnost některé položky vložit vícekrát (zaměstnanci nebo stroje), ale zároveň také některé položky úplně vynechat (hnojiva, postřiky, zaměstnanci a stroje).

Další nevýhoda, kterou jsem objevil, je ukládání historie přenášených dat. Pokud uživatel uloží např. informace, že nějakou činnost dělal jeden zaměstnanec a jeden stroj, pak proběhne napoprvé vše v pořádku. Pokud ale uživatel bude chtít uložit stejné informace, ale již si přeje vynechat stroj, pak aplikace uložila místo vynechaného stroje stejné informace jako při předchozím uložení. Bylo tedy nutné zjistit, jestli uživatel chce přenášet informaci a pokud ne, tak ji vynulovat. Přenesením této prázdné proměnné pak bylo PHP skriptu řečeno, že tuto položku může vynechat.

4.5 Dynamická práce s komponentami

Při práci s aplikací se některé vlastnosti používaných komponent mění proto, aby uživatele upozornili na jeho chybu nebo když si to uživatel vyžádá – např. přidání nové komponenty [16].

4.5.1 Přidání komponenty

Při této činnosti opět využíváme toho, že každá komponenta může být něčím potomkem. V aplikaci se dynamické přidávání komponenty využívá např. u objednávek. Uživatel má možnost ke každé objednávce připojit několik plodin. Pokud tedy klikne na tlačítko pro přidání plodiny, zobrazí se nový *ComboBox* a dva *textInput.* Jeden pro množství a druhý pro cenu vybrané plodiny. *ComboBox* slouží pro výběr plodiny. Pro přidávání těchto komponent jsou připraveny tři rodičovské komponenty typu *Canvas*, jež jsou na začátku prázdné. Důvodem, proč jsou komponenty tři a ne jen jedna, je následná snažší práce při ověřování správnosti vložených dat. Nemusíme totiž vůbec testovat vybrané položky v komponentách *ComboBox*, čímž ušetříme třetinu práce. Kdyby totiž všechny položky měly jednoho rodiče, musely by jsme procházet všechny potomky, tedy i ty, které takto můžeme vynechat.

Nejvíce lze přidat sedum nových položek pro plodiny. Toto omezení je kontrolováno tak, že se použitím funkcí *getChildren* a *length* zjistí aktuální počet potomků komponenty typu *Canvas*.

```
var pocet:int = nazev_rodice.getChildren().length;
```

Příklad 16: Zjištění počtu potomků

Pokud je vše v pořádku, můžeme přidat nového potomka, kterému nastavíme základní vlastnosti, jako je velikost, identifikátor a umístění na ploše. Při počítání souřadnic, kam nového potomka zobrazit, přichází na řadu počet všech potomků. Právě tolikrát posuneme nového potomka v ose y. V případě, že přidáváme *ComboBox,* musíme mu také nastavit zdroj dat, která bude zobrazovat. K tomu opět použijeme vlastnost *dataProvider*, čímž vybereme řádky ze zdrojového XML souboru a vlastností *labelField,* pak řekneme, který sloupec se má zobrazit. Nyní máme nového potomka připraveného na vložení do scény a to se provede pomocí funkce *addChild.*

```
nazev_rodice.addChild(nazev_pomotka);
```

Příklad 17: Vložení potomka do aplikace

4.5.2 Vymazání komponenty

Poté, co jsme do aplikace za běhu přidali komponentu, musíme ji umět také odstranit. Podobně jako při přidání je opět důležitý aktuální počet potomků. Pokud totiž není větší jak nula, znamená to, že nemáme co odebrat. V opačném případě pak projdeme všechny potomky a posledního z aplikace odstraníme. Aplikace také používá odstranění všech potomků, a to v případě, že uživatel přejde do jiné části a poté se vrátí. Vždy se mu tak naskytne stejné výchozí nastavení.

5 Bezpečnostní opatření

Při vytváření aplikací v prostředí Flex musíme mít napaměti bezpečnostní omezení. Tyto aplikace totiž mohou přistupovat k údajům kdekoliv na internetu (v případě AIR aplikací i v počítači) a to není žádoucí. Proto má každá aplikace implicitně omezenu možnost získávat pro sebe data z jiných domén. Platí pravidlo, že aplikaci jsou přístupná data pouze ze stejné domény, ve které je sama aplikace umístěna. To je jistě výhodné a uživatele to chrání.

V praxi je ale nutné umožnit aplikaci získat data i z jiných domén. V ukázkové aplikaci bylo nutné povolit aplikaci přistupovat k datům v databázi, která je na jiném serveru. Umožnit aplikaci získat data z jiného serveru můžeme pomocí souboru s názvem *crossdomain.xml*. Ten musí být ve stejném adresáři jako aplikace. Do tohoto soubor pomocí tagu *allow-access-from* a jeho atributu *domain* říkáme, ke kterým serverům aplikaci umožníme přístup.

```
<?xml version="1.0"?>
<cross-domain-policy>
 <allow-access-from domain="mysql.webzdarma.cz" />
 <allow-access-from domain="cechpavel.wz.cz" />
</cross-domain-policy>
```

Příklad 18: Ukázka obsahu souboru *crossdomain.xml* pro ukázkovou aplikaci

6 Závěr

Vytvořená aplikace splnila základní požadavky na funkčnost. Její nasazení v ostrém provozu je tedy možné. Aby ovšem plnila skutečný efekt informačního systému je potřeba dodělat výstupní části. Např. celkový přehled rentability jednotlivých plodin. Zajímavým rozšířením může být ukládání informací o počasí, protože počasí v zemědělství hraje důležitou roli. Takovéto rozšíření by pak poskytlo uživateli kompletní práci s daty. Tedy nejen jejich ukládání a schromažďování, ale také zajímavé grafické výstupy a statistiky.

Vytváření aplikací v prostředí Flex se dá považovat za velmi příjemné. K tomu velmi přispívá jednoduchost jazyka MXML a efektivní využívání jazyka ActionScript. Pokud porovnáme vývoj podobné aplikace pomocí prostředků moderních webových technologií (XHTML, AJAX), oceníme především možnost přímé tvorby vzhledu aplikace (*Design menu*). Výhodou je také platformová nezávislost, odpadá tedy zkoušení výsledného vzhledu pro různé prohlížeče. Důležitá je také cena, která je v tomto případě spíše nižší a je tak dalším důvodem proč se rozhodnotu právě pro tuto cestu vývoje své nové aplikace.

Literatura

[1]*Adobe® Flex™ Builder 3 Pro for Education* [online]. c2009, [cit. 2009-05-17]. <https://freeriatools.adobe.com/flex/student.php>.

[2] *open source framework | Adobe Flex* [online]. c2009, [cit. 2009-05-17]. <http://www.adobe.com/products/flex/>.

[3] *MXML – Wikipedia, the free encyclopedia* [online] poslední revize 18.5.2009, [cit. 2009-05-18]. <http://en.wikipedia.org/wiki/MXML>.

[4] *Flex 3 – Adobe Flex 3 Help* [online]. , [cit. 2009-05-18]. <http://livedocs.adobe.com/flex/3/html/help.html?content=usingas_2.html>.

[5] *workstation2151 Clipart | Freeze.com* [online]. c2001-2007, [cit. 2009-05-18]. <http://www.freeze.com/Content/Item.aspx?pid=18895>.

[6] server2 *Clipart | Freeze.com* [online]. c2001-2007, [cit. 2009-05-18]. <http://www.freeze.com/Content/Item.aspx?pid=16426>.

[7] *database Clipart | Freeze.com* [online]. c2001-2007, [cit. 2009-05-18]. <http://www.freeze.com/Content/Item.aspx?pid=16957>.

[8] *Integritní omezení a transakce v MySQL* [online]. c2005-2006, [cit. 2009-05-18]. <http://www.manualy.net/article.php?articleID=17>.

[9] *Flex 3 – Adobe Flex 3 Help* [online]. , [cit. 2009-05-18]. <http://livedocs.adobe.com/flex/3/html/help.html?content=embed_4.html>.

[10] *Adobe – Flex Quick Start Basics: Skinning your components* [online]. c2009, [cit. 2009-05-18]. <http://www.adobe.com/devnet/flex/quickstart/skinning_components>.

[11] *Flex 3 – Adobe Flex 3 Help* [online]. , [cit. 2009-05-18]. <http://livedocs.adobe.com/flex/3/html/help.html?content=dpcontrols_6.html>.

[12] *Flex 3 – Adobe Flex 3 Help* [online]. , [cit. 2009-05-18]. <http://livedocs.adobe.com/flex/3/html/help.html?content=dpcontrols_5.html>.

[13] *Flex 3 – Adobe Flex 3 Help* [online]. , [cit. 2009-05-18]. <http://livedocs.adobe.com/flex/3/html/help.html?content=data_access_2.html>.

[14] *Flex 3 – Adobe Flex 3 Help* [online]. , [cit. 2009-05-18]. <http://livedocs.adobe.com/flex/3/html/help.html?content=09_Working_with_Strings_09.html>.

[15] *Flex 3 – Adobe Flex 3 Help* [online]. , [cit. 2009-05-19]. <http://livedocs.adobe.com/flex/3/html/help.html?content=navigators_3.html>.

[16] *Flex – komponenty pomocí ActionScriptu | Flash.cz* [online]. 2007-05-07, [cit. 2009-05-19]. <http://www.flash.cz/portal/clanek.aspx?id=764>.

[17] *Flash tip trik – externí data a zabezpečení | Flash.cz* [online]. 2007-01-03, [cit. 2009-05-19]. <http://flash.cz/portal/clanek.aspx?id=512>.

[18] BOLDIŠ, Petr. *citace2.pdf* [online]. c 1999-2004, poslední aktualizace 11.11.2004. <http://www.boldis.cz/citace/citace2.pdf>.

Seznam příloh

Příloha 1. CD se zdrojovým kódem ukázkové aplikace