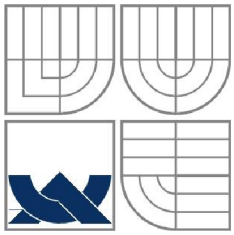
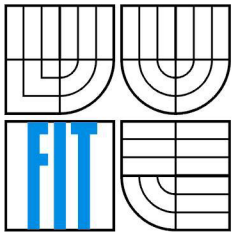


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

MICROSOFT ASP.NET DYNAMIC DATA V PROSTŘEDÍ WINFORMS

MICROSOFT ASP.NET DYNAMIC DATA IN WINFORMS

BAKALÁŘSKÁ PRÁCE
BACHELOR THESIS

AUTOR PRÁCE
AUTHOR

MARCEL ŠERÝ

VEDOUCÍ PRÁCE
SUPERVISOR

ING. MICHAL KADÁK

BRNO 2011

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Michala Kadáka a vedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Brně dne 16. května 2011 .....

Poděkování

Rád bych zde poděkoval Ing. Michalu Kadákovi za fundovaný pohled odborníka a cenné rady při vypracování této práce. Dále bych rád poděkoval Ing. Pokornému za jeho praktickou a konstruktivní kritiku při konzultacích výsledného systému.

Abstract

This bachelor's thesis describes the development of an universal application core allowing developers to build flexibile informational systems in no time. The system follows on existing technology Microsoft ASP.NET Dynamic Data and implements it's ideas on the Windows forms platform. In the opening parts of this thesis are briefly described topics such as the Microsoft .NET plaform, the Dynamic Data, or the questions of a switch-over from the web platform to Windows forms. Following parts of the thesis are dealing with the actual implementation of the example informational system. The conclusion is among other things dedicated to the analysis of an usability of the product in a real world and demands of today's market.

Abstrakt

Tato bakalářská práce se zabývá vývojem univerzálního aplikačního jádra umožňujícího rychlý vývoj informačních systémů, které zároveň poskytuje dostatečnou flexibilitu a možnosti přizpůsobení. Navazuje na již existující technologii ASP.NET Dynamic Data firmy Microsoft a implementuje její myšlenky na platformě desktopových aplikací Windows. V úvodních částech této práce je ve stručnosti popsána technologie .NET framework, Dynamic Data, a nástin problematiky a úskalí přechodu z webové platformy na Windows forms. Další části se věnují samotné implementaci a popisu tvorby ukázkového informačního systému. Závěr je věnován mimo jiné i analýze možností reálného využití a požadavků dnešního trhu.

Keywords

Microsoft .NET framework, ASP.NET, Dynamic Data, Windows forms, Informational system

Klíčová slova

Microsoft .NET framework, ASP.NET, Dynamic Data, Windows forms, Informační systém

Citace

Šerý, M. *Microsoft Dynamic Data v prostředí WinForms.* Bakalářská práce. Brno: VUT v Brně, 2011.

Obsah

1 Úvod

Příchod moderních počítačů zcela redefinoval vnímání pojmu informační systém. Velkou část manuální práce začaly přejímat počítače. Smyslem moderních informačních systémů je zjednodušit a zautomatizovat podnikové procesy s účelem zefektivnění chodu firem a podniků. Moderní technologie poskytují prostředky pro informační systémy, jež umožňují tyto podnikové procesy velmi přesně reflektovat a tím ještě více zjednodušit a zrychlit fungování podniků.

V dnešním dynamickém světě je ovšem stále větším problémem flexibilita. Archaické informační systémy nasazené v mnoha podnicích nestačí reflektovat změny podnikových procesů a tím se stávají překážkou v nasazování nových metod a přístupů. Z tohoto důvodu je dnes kladen důraz na flexibilitu informačních systémů. Ten ovšem staré systémy nejsou schopny reflektovat, protože jsou postaveny na starých základech a myšlenkách, které jsou po mnoha letech vývoje protkané celým systémem. Příkladem jsou ERP systémy jako SAP či KARAT, které jsou plné vazeb, závislostí a podmínění. (Herout, 2003)

Tvorba takovýchto informačních systémů je velmi komplexní téma a existuje mnoho cest, jak tuto problematiku řešit. Tato bakalářská práce se pokusí navrhnout řešení prezentační části dynamického informačního systému, konkrétně jádro pro zobrazování a editaci dat schopné obsloužit data obecně jakékoliv struktury. Tato myšlenka je inspirována webovou technologií ASP.NET Dynamic Data uvedenou společností Microsoft v rámci .Net framework 3.5 SP1 roku 2008. (MacDonald, Freeman a Szpuszta, 2010)

2 Technologie

Pro správné pochopení myšlenek tohoto projektu je nezbytné nastínit alespoň základní fakta a souvislosti, a uvést čtenáře do kontextu. Proto budou níže rozebrány základy platformy .NET a související technologie. Tato kapitola zároveň zdůvodňuje výběr cílové platformy a neduhy stávající technologie Dynamic Data, z níž si tato práce bere inspiraci.

Platforma .NET je soubor softwarových technologií tvořících kompletní platformu pro vývoj softwarových aplikací. Sám o sobě je velmi rozsáhlý a poskytuje velmi širokou škálu možností pro budování webových, konzolových, desktopových, mobilních a dnes dokonce i RIA aplikací.

Již od prvního představení v roce 2001 byly do této platformy vkládány velké naděje, jelikož měla řešit problémy a nedostatky tehdejších vývojových platforem. Při návrhu byl kladen velký důraz na podporu lokalizací, zabezpečení, reflexe a verzování knihoven. Další silnou myšlenkou této platformy je běh aplikací v samostatném a nezávislém prostředí podobném mechanismu VirtualMachine, se kterým přišla platforma Java. To umožňuje provozovat .NET aplikace nezávisle na hardwarových či softwarových prostředcích. Tato vlastnost dala za vznik open-source implementacím této plaformy, jako je například projekt Mono. Tento původně komunitní projekt převzala společnost Novell, která z něj vytvořila schopnou platformu pro vývoj pod systémy Unix či MacOS. Dalším zajímavým portem .NET plaformy je například její svobodná implementace DotGNU.

Společnost Microsoft od oficiálního uvedení .NET verze 1.0 v roce 2002 vydala celkem pět dalších verzí frameworku, a to 1.1, 2.0, 3.0, 3.5 a nejnovější verzi 4.0. Nejzásadnější ovšem byla verze 2.0 z roku 2005. Ta přinesla nové jádro pro ASP.NET, které bude popsáno níže, a mnoho změn v oblasti Windows Forms. Pozdější verze frameworku staví na této druhé verzi a jsou ze své podstaty pouze nadstavbami. (MacDonald, Freeman a Szpuszta, 2010; Pokorný, 1992)

Pro tento projekt je velmi důležitou vlastností platformy .NET výše zmíněná reflexe. Ta zjednodušeně umožňuje v aplikaci pracovat s neznámým sestavením, například .NET knihovnou bez toho, abychom ji v době kompilace znali. Pro představu, tento proces umožňuje například procházet veřejnými třídami daného sestavení, číst jejich atributy a vytvářet za běhu jejich instance.

Technologie ASP.NET je nástupcem původního interpretovaného jazyka ASP, ač s ním má společného minimum. ASP.NET je postaveno nad platformou .NET, což zpřístupnilo do té doby nevídané možnosti v oblasti implementace (.NET jazyky Visual Basic.NET a C#), databází (ADO.NET) a vůbec způsobu vývoje webových aplikací. Navíc v pozdějších verzích .NET frameworku přišly rozšiřující technologie jako např. Entity framework, LINQ 2 SQL, Dynamic data nebo podpora pro AJAX. (MacDonald, Freeman a Szpuszta, 2010)

Jak původní ASP, tak ASP.NET technologie jsou silně cílené na web a poskytování webového obsahu. Není tedy možné aplikaci přenést například na klienta, jako je tomu u RIA technologií Microsoft Silverlight nebo Adobe Flex.

ASP.NET Dynamic Data jsou nadstavbou nad platformou ASP.NET a stala se inspirací pro tuto bakalářskou práci. Umožňují vytvořit daty řízenou webovou

aplikaci během velmi krátké doby a s minimem úsilí. Funguje tak, že na zaregistrovaný databázový model (Entity Framework či LINQ 2 SQL) je schopen vygenerovat obecné rozhraní pro prohlížení a editaci dat všech tabulek v modelu, které je schopen vývojář velmi jednoduše přizpůsobit potřebám systému. Takže vývojář přistupuje k obecnému systému, který je schopen samostatně fungovat a pracovat se všemi tabulkami modelu i bez jediného zásahu do zdrojových kódů. Problém této technologie je však její webový základ, který je velmi svazující, ať již po stránce uživatelského rozhraní, integrace s jinými aplikacemi, nebo komunikace s periferiemi. Velký nedostatek je spatřován právě v uživatelském rozhraní. Rychlost odezvy uživatelského rozhraní je závislá na aktuální konektivitě klienta, což negativně ovlivňuje uživatelský komfort. Právě proto je cílem této práce implementovat tuto myšlenku na platformě Windows Forms, která těmito neduhy netrpí.

Windows forms (dále jen WinForms) jsou rozsáhlou sadou .NET knihoven poskytujících abstrakci nad standardními funkcemi Win32 API například pro práci s okny či ovládacími prvky. Její integrace do vývojového prostředí MS Visual Studio byla již od první verze .NET na velmi dobré úrovni. Také je u této platformy výhodou, že kompletní uživatelské rozhraní je postaveno na klasických .NET třídách (komponentách) a ne například na jazyku XAML jako v případě nových .NET technologií WPF či Silverlight. Tento fakt se stal jedním z rozhodujících argumentů pro zvolení WinForms jako cílové platformy. Dalším významným momentem byla již výše zmíněná rychlost odezvy uživatelského prostředí, a tím i vysoký komfort uživatelů výsledných systémů.

3 Popis systému

Systém DynamicForms slouží jako jeden z pilířů při budování informačních systémů. Zajišťuje tvorbu uživatelského rozhraní, a to jen na základě předaných dat a případných poskytnutých rozšíření v podobě aplikačních knihoven. Tato rozšíření programuje osoba zvaná implementátor. Systém DynamicForms si klade za cíl poskytnout implementátorům komplexní a dostatečně obecný nástroj. Nástroj, který šetří práci nejen při samotné implementaci, ale hlavně při budoucím rozšiřování systému. Další klíčovou vlastností systému je to, že poskytuje implementátorům obrovskou škálovatelnost stupně implementace. Mohou definovat uživatelské rozhraní od té nejobecnější úrovně společné pro všechny objekty v systému, až po implementaci pro konkrétní datový typ, v konkrétním případu užití. To dává implementátorům k dispozici širokou škálu možností, jak přesně dle potřeb rozšiřovat systém tak, aby zároveň šetřili čas a nemuseli definovat každý formulář a ovládací prvek pro všechny datové typy v systému.

Rozšiřování systému zvenčí probíhá téměř bezobslužně, pouhým dodáním vlastních knihoven do náležitého adresáře. Tyto knihovny není potřeba na žádném místě či souboru registrovat, či s nimi jakkoliv manipulovat. Aplikace si je vždy při vytvoření jádra DynamicForms automaticky načte a vyhledá v nich použitelné komponenty a jejich mapování na data. Názornou ukázku implementace vlastních formulářů a ovládacích prvků lze nalézt v kapitole 5.4.

Editace a procházení dat je nejzákladnější požadavek na systém tohoto typu. Ovšem podpora dědičnosti a případů užití posunují systém DynamicForms daleko nad systém Dynamic Data či jiné alternativy. Právě využití dědičnosti umožňuje škálovat zacílení vlastních implementací tak, aby se zabránilo redundantním pracím.

Pro představu je uveden krátký příklad: Nechť je definována třída Osoba a k ní vlastní formulář se specifickým rozložením ovládacích prvků. V případě, že je požadavek tuto implementaci systému rozšířit o novou třídu Zaměstnanec, která dědí od třídy Osoba, není potřeba jediného zásahu do stávající implementace rozšíření. DynamicForms při předání datového objektu této nové třídy pro ni sám najde nejpříhodnější formulář. Tím je v tomto případě předem definovaný formulář pro třídu Osoba, jelikož je v dědičné linii nejblíže třídě Zaměstnanec. Systém je tak obecný, že implementátor může definovat vlastní ovládací prvek pro číselné atributy třídy Osoba, případně pro jeden konkrétní atribut, dle jeho jména a ještě jen pro některé případy užití. Lze tedy například implementovat i přístupová práva, kdy uživatel s administrátorským oprávněním (případ užití „Administrátor“) může mít ve stejném formuláři jiné ovládací prvky než standardní uživatel. Stejně tak je možné pro různé případy užití aplikovat rozdílné validační mechanismy. Toho lze využít například u validace uživatelského jména, kdy jsou rozdílné požadavky na formát pro různé typy uživatelů. Protipólem výše zmíněných konkretizací může být zobecnění, kdy je definován formulář pro obecný objekt jakéhokoliv typu. Například za účelem sjednocení uživatelského rozhraní.

Systém poskytuje mnoho cest, jak docílit požadované funkcionality, a proto je potřeba při návrhu cílového informačního systému dopředu vymyslet, jak co nejefektivněji využít jeho vlastností tak, aby se využil potenciál tohoto systému naplno, a tím se ušetřil čas i práce.

4 Návrh systému

V této kapitole budou popsány základní myšlenky a principy systému DynamicForms. Dále zde bude popsána struktura, možnosti a parametry jednotlivých součástí systému, tvořících jádro DynamicForms. Všechny informace v této kapitole jsou nezbytné pro pochopení postupu implementace, která je popsána v kapitole 5. Při návrhu systému DynamicForms bylo vycházeno ze základních požadavků, které byly stanoveny na základě konzultací tohoto projektu a studia Microsoft DynamicData a systému Karat. Výsledný systém DynamicForms by měl splňovat následující požadavky:

1. Nezávislost na okolí.
2. Nezávislost na klientské aplikaci.
3. Editace dat obecně jakékoliv struktury.
4. Procházení kolekcí těchto dat.
5. Podpora dědičnosti dat.
6. Podpora pro různé případy užití (UseCase) a jejich kombinování.
7. Aplikace vlastních validačních mechanismů.

Nezávislostí na okolí je myšleno omezení referencí jádra na knihovny třetích stran. Smyslem tohoto požadavku je snadná distribuce a případně snadné překompilování pro starší verze .NET Frameworku. V této práci je jádro kompilováno pro .NET 4.0, a to z důvodu, že jej využívá i ukázkový informační systém. Ten jej vyžaduje kvůli jeho implementaci technologie Microsoft Entity Framework (MacDonald, Freeman a Szpuszta, 2010), ale o tom dále, v kapitole 6. Jádro samotné by ovšem mělo být kompilovatelné i pro starší verze, počínaje .NET frameworkem 2.0.

Vzhledem k potřebě nezávislosti na klientské části bylo potřeba navrhnout systém dostatečně obecně bez závislosti na nějaké další technologii jako např. Entity Framework či Linq2Sql. To totiž zaručuje silnou univerzálnost celého řešení a jeho použitelnost za všech okolností. S touto univerzálností souvisí i třetí a čtvrtý bod. Systém musí být schopen pracovat obecně s jakýmkoliv strukturovaným datovým typem. Takže musí umět pracovat jak s klasickými třídami .NET frameworku, tak s datovými třídami (tzv. Entity) z Entity frameworku. (MacDonald, Freeman a Szpuszta, 2010)

Dědičnost je jeden ze základních principů OOP, a tak je velmi často využívána. V případě jazyka C# je využívána dokonce vždy, protože jsou v něm úplně všechny třídy potomky elementární třídy Objekt. Z tohoto důvodu také vzešla tato klíčová myšlenka využití dědičnosti i v kontextu uživatelského rozhraní. Tato myšlenka umožňuje využívat formuláře bázových tříd i pro interpretaci tříd z nich odvozených. To samozřejmě ve výsledku vede k rychlejší a jednodušší implementaci.

Požadavek na podporu případů užití vzešel z odborné konzultace s ing. Pokorným. Jeho smyslem je rozlišení různých stavů klientské aplikace z pohledu implementovaných rozšíření jádra. To umožní pro různé stavy aplikace vybrat specifické formuláře, ovládací prvky, překlady a dokonce i specifické validační mechanismy pro daný konkrétní případ užití. V případě, že se implementátor bude řídit pokyny z manuálu, tak systém dokonce umožní kombinování těchto případů užití, což znamená, že předaný

případ užití může být kombinací více stavů a jádro i z této kombinace rozpozná, který formulář či ovládací prvek využít.

Systém dále musí umožnit na různé vstupní ovládací prvky (textová pole apod.) aplikovat vlastní validační mechanismy tak, aby bylo možné kontrolovat správnost informací zadaných uživatelem již na straně klientské aplikace a nemusela se tato logika aplikovat až na straně serveru. Tento požadavek je opět řešen tak, aby eliminoval redundantní práci, takže jeden validační mechanismus lze uplatnit v mnoha situacích.

Je zřejmé, že systém zabírá velmi širokou problematiku, avšak díky kvalitnímu návrhu se v této práci podařilo implementovat všechny vytyčené požadavky a cíle. Návrhem konkrétních řešení se zabývá celá tato kapitola.

4.1 Základní myšlenky

Systém DynamicForms je z principu daty řízený systém (MacDonald, Freeman a Szpuszta, 2010). Většina systémů podobného ražení pracuje s metadaty, nesoucími dodatečné informace a popis předmětných dat, uloženými například v xml souborech. Tato metadata poté slouží pro generování uživatelského rozhraní koncové aplikace. Tento přístup má sice své výhody, ale při sebemenší změně dat je vetšinou potřeba upravit jak metadata, tak často i koncovou aplikaci Proto byl pro systém zvolen odlišný přístup.

Místo popisování dat metadaty bylo vycházeno z myšlenky, že opravdové dynamické rozhraní musí být zcela nezávislé na zpracovávaných datech a je potřeba zbavit je veškeré sémantiky. Jedině tak lze dosáhnout plné dynamičnosti a hlavně znovupoužitelnosti jednotlivých prvků. Této separace dat od jejich sémantiky dosahuje systém DynamicForms použitím modulárního systému, jenž každé datové třídě přiřazuje (mapuje) konkrétní formuláře pro seznam a detail objektu, a sady ovládacích prvků vstupu pro každý z jejích atributů. Tyto formuláře a vstupní ovládací prvky se načítají dynamicky z modulů přiložených k aplikaci. V praxi tedy umožňuje např. v rámci aktualizace nahrát další knihovnu, a tím systém rozšířit o možnosti práce s novými typy dat.

Síla tohoto řešení je především ve výše zmíněném mapování. To totiž zaručuje, že systém bude za všech okolností pracovat s daty jakékoliv struktury i bez přídavných modulů, a také umožňuje velmi snadnou rozšiřitelnost datových struktur pomocí dědičnosti. Zjednodušeně lze říci, že při procesu mapování se pro danou třídu objektů hledá nejvhodnější formulář pro její typ, *nebo některý z jejích rodičovských (bázových) typů*. Stejně tak v závislosti na typu objektu určí mapování nejvhodnější vstupní ovládací prvky pro každý z atributů této třídy. Problematika mapování je poměrně komplexní, a bude proto více rozvedena níže, v kapitole „Vyhodnocení mapování“.

Systém DynamicForms poskytuje mimo samotných formulářů i prostředky pro lokalizaci a validaci jednotlivých ovládacích prvků. Lokalizace umožňuje obejít omezení pojmenovávání jednotlivých atributů datových tříd tím, že v klientských implementacích definujeme lokalizační slovníky. U nich je možno využít stejných mapo-

vacích principů jako u dynamických prvků. Lze tedy definovat názvy pro konkrétní třídy, či dokonce názvy specifikovat pro konkrétní případy užití.

4.2 Struktura systému

V této kapitole budou postupně uvedeny všechny hlavní součásti systému s popisem jejich funkce, vazeb a použití. Pro úplnost je přiložen v příloze A kompletní třídní diagram systému se všemi veřejnými metodami a poli. Pro přehlednost byly v tomto diagramu všechna privátní pole a metody vnitřní implementace skryta.

V souvislosti se strukturou systému DynamicForms je zaveden pojem *Dynamický prvek*, jenž reprezentuje bázové třídy pro vstupní ovládací prvky, bázové třídy pro formuláře detailu, bázové třídy pro formuláře seznamu a všechny jejich potomkovské implementace. Každý typ dynamického prvku má svůj jmenný podprostor, ve kterém se nachází náležící bázová třída a s ní i její výchozí (vestavěné) implementace pro základní datové typy. Jádro systému DynamicForms má tedy následující jmenné podprostory: Forms, Inputs, Columns, Validation, Localization a jestě podprostor Mappings, ve kterém se nacházejí třídy Mapping a MappingList, které budou stručně popsány v tabulce níže a detailněji v kapitole 5.1.

Pro přehled je zde uvedena i tabulka jednotlivých součástí systému s jejich krátkým popisem a příznakem viditelnosti pro klientskou aplikaci. Hlavní součásti, které budou hlouběji rozebrány níže v této kapitole, se nachází v první části tabulky. Bázové třídy budou více popsány v kapitole 5.4.

Třída DynamicFormsCore reprezentuje jádro celého systému DynamicForms, které zajišťuje většinu komunikace mezi klientskou aplikací a systémem. Zpravidla se za běhu klientské aplikace vytváří jedna její instance, nicméně není to podmínkou. Při jejím vytvoření se jí předává název klientské aplikace a volitelně i adresář s knihovnami dynamických prvků. Název aplikace se využívá jednak pro účely logování událostí a chyb, a také jnapř. ej využívají zabudované formuláře pro vytvoření titulků oken. Jádro obsahuje dvě základní metody vystavené klientské aplikace a dvě její zjednodušené nadstavby. Jedná se o metody *DetailForm* a *ListForm*, které při jejich volání vracejí vygenerované formuláře dynamického GUI. Metoda DetailForm slouží pro vygenerování editačního formuláře, ve kterém může uživatel upravovat data předaného objektu. Její zjednodušená nadstavba, metoda Detail, oproti tomu formulář přímo zobrazí a vrací logickou hodnotu informující o případných změnách v upravovaném objektu. Metoda ListForm vrací formulář seznamu, ve kterém lze procházet kolekce dat určitého typu. Její zjednodušená verze List, analogicky k metodě Detail, přímo zobrazí formulář seznamu v novém okně. Vzhledem k tomu, že při procházení seznamu dat může docházet k úpravám listovaných objektů na více místech, používá metoda ListForm mírně odlišný mechanismus informování o změnách. Využívá tzv. událostí (Virius, 2006), kde při každém uložení editované položky vyvolá událost ItemChanged, a tím informuje klientskou aplikaci o změnách.

Mapping Manager zastřešuje klíčovou část systému, a to načítání uživatelských knihoven. Načítá nalezené mapování a k nim příslušející třídy do speciálních seznamů *MappingList*, které budou hlouběji popsány v kapitole 5.1. Tyto seznamy umožňují v kolekci mapovacích vazeb vybrat tu nejvhodnější pro daný datový typ.

Tabulka 1: Přehled důležitých tříd systému DynamicForms

Název	Viditelnost	Popis
DynamicFormsCore	ANO	Jádro systému DynamicForms zajišťující komunikaci s klientskou aplikací.
MappingManager	NE	Třída řešící kompletní problematiku mapování od načítání klientských knihoven po výběr nejvhodnějšího mapování.
InterfaceBuilder	NE	Třída obsluhující proces tvorby uživatelského rozhraní.
LocalizationManager	NE	Vzhledem ke komplikovanosti problematiky lokalizace byla pro tento účel vytvořena samostatná řídící třída.
Báze pro dynamické prvky a lokalizaci		
DynamicDetailBase	ANO	Bázová třída pro formuláře detailu.
DynamicListBase	ANO	Bázová třída pro formuláře seznamu.
DynamicColumnBase	ANO	Bázová třída pro dynamické prvky sloupců pro formulář seznamu.
DynamicInputBase	ANO	Bázová třída pro vstupní dynamické prvky pro formulář detailu.
DynamicValidatorBase	ANO	Bázová třída pro validátory vstupních dynamických prvků.
DynamicLocalizationBase	ANO	Bázová třída pro lokalizační dynamické prvky.
Pomocné třídy		
Mapping	ANO	Atribut definující vazby dynamických prvků na data. Je to základní stavební kámen celého systému.
MappingList	NE	Třída rozšiřující systémový generický seznam System.Collections.Generic.List<Mapping> o specifickou třídící funkcionalitu umožňující vybrat pro daný typ objektu či jeho atributu nejvhodnější mapování.

Mapping Manager rozlišuje šest základních typů mapování pro jednotlivé báze (viz tabulka výše). Pro každý typ mapování má vyhrazen samostatný MappingList. Ostatní třídy v rámci systému DynamicForms k těmto seznamům přistupují přes příslušné atributy instance MappingManageru. Primárně MappingManager využívá třída InterfaceBuilder při hledání vhodných dynamických prvků.

Třída InterfaceBuilder se stará o budování uživatelského rozhraní. Je tedy klíčovou součástí celého systému, která implementuje logiku tvorby dynamických prvků, příslušných validátorů a jejich lokalizací. Vrací surové neinicializované formuláře. Jejich inicializaci má na starost třída DynamicFormsCore, která vytvořené formuláře dále zpracovává a předává klientské aplikaci.

5 Implementace jádra systému

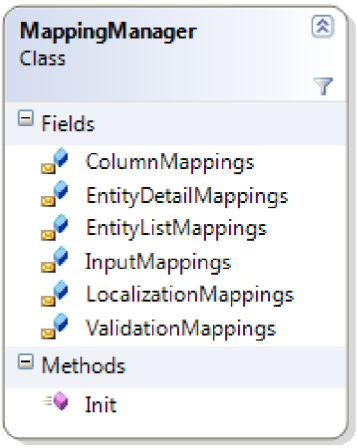
5.1 Metadata

Metadata v klasickém podání znamenají data popisující data. Ovšem v kontextu DynamicForms znamenají definování vazby od dynamického prvku (viz kapitola 4.2) k danému datovému typu, nikoliv naopak. Proto byl v této práci zvolen výstižnější název pro popis metadat, a to *Mapování*. Vzhledem k povaze systému se tedy mapují dynamické prvky k datům.

Jako nejvhodnější formu popisu mapovacích metadat jsem zvolil jeden z původních prostředků platformy .NET, jímž jsou *atributy* (Virius, 2006, s. 19). Tato metoda umožňuje přidat obecně jakákoliv metadata k jakékoliv třídě či některému z jejích polí (atributů[1]). Navíc těchto atributů lze pro jednu třídu definovat neomezené množství, což se pro tento projekt hodí. Umožní to každý dynamický prvek mapovat na neomezené množství tříd. Pro potřeby systému DynamicForms byla tedy vytvořena třída Mapping odvozená od .NET třídy Attribute reprezentující informaci o mapování.

Inicializace Mapping Manageru

MappingManager je z hlediska implementace poměrně triviální třídou, jelikož většina mapovací logiky byla v průběhu vývoje systému abstrahována do tříd Mapping.Comparator a MappingList kvůli kvalitě výsledného kódu.



Obrázek 1: Třída MappingManager

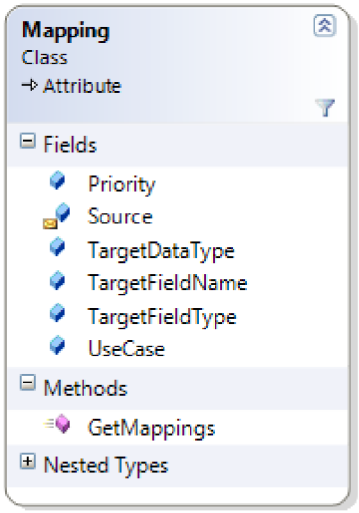
K inicializaci Mapping Manageru (dále jen MM) slouží metoda Init. K jejímu volání dochází při vytvoření instance třídy DynamicFormManager. Načítají se při ní všechny podklady pro fungování systému DynamicForms. Na to je potřeba brát zřetel především u komplikovanějších systémů, kde se mohou načítací časy neúměrně prodlužovat, a s tím může vznikat i nepříjemné „zmražení“ uživatelského rozhraní.

[1]Společnost Microsoft zde vytvořila mírný zmatek v názvosloví, jelikož termín atribut se již dlouhá léta používá v kontextu OOP pro pojmenování datových složek objektových tříd.

Samotná inicializace probíhá tak, že se postupně načítají dodané klientské knihovny a i samotná knihovna DynamicForms (Core). V každé z těchto knihoven MM hledá veřejné třídy dynamických prvků, jež jsou potomky některého z bázových dynamických prvků (viz kapitola 4.2). Pro každý takovýto nalezený dynamický prvek jsou načteny jemu definovaná mapování a ty zaznamenány do vnitřních seznamů sloužících pro proces mapování. Mimo to jsou ještě mapování lokalizačních dynamických prvků zaslány do LocalizationManageru, který si je dále zpracovává.

Atribut Mapping

Jak již bylo řečeno výše, třída Mapping slouží pro definování vztahu dynamických prvků k jednotlivým prvkům. Přidává se do hlaviček jednotlivých dynamických prvků v rámci systému DynamicForms, a tím k danému prvku přidává vazbu k danému datovému typu či atributu. Implementačně vychází ze systémové třídy Attribute. Třída mapping má následující strukturu:



Obrázek 2: Třída Mapping

Všechna její pole jsou nepovinná, nicméně pro přesnější zacílení daného dynamického prvku je potřeba alespoň některé pole zadat. Jednotlivá pole mají různou prioritu při vyhodnocování. To je detailněji rozebráno v následující podkapitole „Vyhodnocení mapování“. V zásadě ale platí přímá úměra mezi vyplněnými položkami a přesností zacílení daného mapování. V diagramu třídy jsou položky seřazeny abecedně, ovšem zde budou popisovány dle logického pořadí.

Položka Source je interním polem. Je tedy pro implementátora skryta, jelikož ji vyplňuje MappingManager při načítání dynamických prvků. Její význam je v tom, že uchovává informaci o zdrojové třídě dynamického prvku, který dané mapování definuje. Slouží pro pozdější vytvoření instance daného dynamického prvku InterfaceBuilderem.

Pole TargetDataType zacíluje mapování pro konkrétní datové třídy. Nese informaci o datovém typu (třídě) cílové položky.

TargetFieldType a TargetFieldName oproti tomu specifikují mapování pro jednotlivá pole datových položek. TargetFieldType umožňuje zacílit dle datového typu pole a TargetFieldName podle názvu pole. Tyto dvě položky zůstávají nevyužité při mapování formulářů a lokalizací. Jejich využití je při mapování sloupcových a vstupních dynamických prvků a jejich validátorů.

Priority a UseCase slouží pro dodatečné rozlišení jednotlivých mapování. Priority umožňují upravovat pořadí mapování při jeho vyhodnocování. Pole UseCase slouží pro definici případu užití daného dynamického prvku. Ten je společný vždy pro celý formulář, takže jej přebírají i dynamické prvky vstupu a sloupce, validátory i lokalizátory. Toto pole má na rozdíl od priority velmi striktní logiku, takže pokud se nenajde žádné odpovídající mapování pro daný případ užití, použije se výchozí případ užití. Tím je zaručeno, že například formuláře určené pro administrátory systému neuvidí řadoví zaměstnanci.

Třída Mapping dále obsahuje statickou metodu GetMappings, která nalezne z předaného datového typu všechna asociovaná mapování a vrátí je volajícímu objektu.

S třídou Mapping jsou úzce spojeny další dvě neméně důležité třídy Mapping.Comparer a MappingList. V následujícím textu budou krátce rozebrány, jelikož jsou základem pro pochopení následující podkapitoly Vyhodnocení mapování.

Mapping.Comparer slouží jako porovnávací třída (komparátor) při řazení seznamů mapování pro daný dotazovaný typ dat. Je tedy základním kamenem pro vyhodnocování nejvhodnějšího mapování pro hledaný datový typ. Implementuje standardní systémové rozhraní IComparer pro třídu Mapping. Jeho implementace je ovšem poněkud atypická. Oproti klasickým komparátorům je tento potřeba inicializovat datovým typem třídy, ze kterého se vygeneruje jeho dědičnostní linie.[2] V této linii je každé generaci přiřazen odpovídající index, tzv. *InheritanceIndex* neboli Index dědičnosti. Podle tohoto indexu se určuje náležitost dotazované třídy k danému mapování.

Třída MappingList, jak již název vypovídá, slouží pro uchovávání seznamů mapování. Mapping manažer ji využívá pro každý typ dynamického prvku. Vychází ze systémové třídy pro generické kolekce System.Collections.Generic.List pro prvky typu Mapping. Avšak oproti této systémové třídě poskytuje poměrně komplexní funkcionalitu mezipaměti. Bez ní by se při každém vyhodnocování mapování pro hledaný datový typ musel celý seznam setřídit a profiltrovat, což by u rozsáhlejších systémů mohlo stát poměrně dost výkonu v případě častého opakování. Proto třída MappingList všechny dotazy na vyhodnocení mapování ukládá do příslušných seznamů, a při opětovném dotazu už jej znovu nevyhodnocuje.

Vyhodnocení mapování

Mapování je klíčovým procesem celého systému DynamicForms. Je implementováno ve třídě MappingList, za spolupráce s komparátorem Mapping.Comparer. Obě tyto třídy jsou popsány v předešlé kapitole. Účelem tohoto procesu je nalézt v seznamu

[2] V případě obecného OOP jazyka by se jednalo o strom, nicméně jazyk C# podporuje pouze jednoduchou dědičnost, takže je implementována jako linie (seznam).

mapování (třída Mapping) to nejvhodnější pro daný typ datového objektu, případně pro některý z jeho atributů. Celá tato procedura je implementována na úrovni třídy MappingList.

Hledání nejvhodnějšího mapování pro pole datového objektu je mírně komplikovanější, proto je třeba jej vysvětlit jako první. Nechť je dán lineární seznam mapování S a vyhodnocovaný typ T. Hledáme mapování pro pole A typu TA. V prvním kroku se vytvoří komparátor typu Mapping.Comparator pro typ T. Tímto komparátorem se seznam S setřídí dle následujících kritérií:

1. Název pole (A)
2. Index dědičnosti datového typu pole (TA)
3. Index dědičnosti datového typu třídy (T)
4. Priorita mapování

Kritéria jsou seřazena dle jejich důležitosti při určování pořadí prvků v seznamu. Z tohoto pořadí vyplývá, že nejvyšší přednost mají mapování se specifikovaným názvem pole. Pokud mají dvě porovnávaná mapování všechny atributy shodné, přichází na řadu poslední kritérium Priorita. Hledání nejvhodnějšího mapování pro atribut datového objektu probíhá obdobně, jen s tím rozdílem, že první dvě třídicí kritéria jsou vynechána a řazení se tak řídí jen podle indexu dědičnosti a priority.

Po seřazení tohoto pole se následně ošetřují případy užití. V případě, že si volající aplikace vyžádala konkrétní případ užití (UseCase), tak se ze seznamu S, který je v tuto chvíli již seřazený, odeberou všechna neodpovídající mapování. Pokud je po tomto kroku seznam prázdný, tak se procedura opakuje, ovšem s volám výchozího případu užití (hodnota 0). V případě že existují položky odpovídající zadanému případu užití, vybere se z odfiltrovaného seznamu první položka.

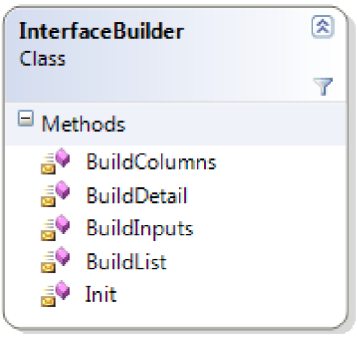
Díky této implementaci zajišťuje mapování hned několik základních požadavků stanovených v návrhu, a to nezávislost na klientské aplikaci, podporu dědičnosti dat a podporu pro různé případy užití (UseCase) a jejich kombinování.

5.2 Generování GUI

Proces generování uživatelského rozhraní má na starost třída InterfaceBuilder. V kontextu DynamicForms se jedná o vytváření nových instancí zdrojových dynamických prvků za běhu aplikace. To je realizováno systémovou třídou System.Activator, která za pomoci reflexe dokáže vytvořit novou instanci třídy daného typu. (Virius, 2006)

Proces generování je řízen jádrem systému, ale jednotlivé kroky jsou implementovány ve třídě InterfaceBuilder. Ta, jak je patrné z obrázku 3, zpřístupňuje jádru čtyři základní prostředky pro budování dynamického uživatelského rozhraní na základě mapování. Jedná se o metody BuildDetail, BuildList, BuildInputs a BuildColumns.

Metody BuildDetail a BuildList jsou poměrně triviální. Spočívají v nalezení nejvhodnějšího mapování pro daný typ a případ užití za pomoci odpovídajícího seznamu MappingList v MappingManageru. Z nalezeného mapování se přečte typ zdrojového dynamického prvku a pomocí výše zmíněné třídy System.Activator se vytvoří její instance, kterou metoda navrací.



Obrázek 3: Třída InterfaceBuilder

U metod BuildInputs a BuildColumns je problematika o poznání komplexnější, jelikož zahrnuje lokalizaci jednotlivých prvků a také připojení validátorů. U obou metod se nejdříve pomocí reflexe zjistí všechna pole datové třídy pro kterou se vstupy či sloupce generují. Poté se pro každé z nalezených polí musí v prvním kroku najít nejvhodnější mapování, ze kterého se, stejně jako v případě formulářů, vytvoří instance dynamického prvku.

V dalším kroku probíhá hledání nejvhodnějšího mapování pro validátor vstupního prvku, ze kterého se následně se vytvoří instance dynamického prvku validátoru. Ten se předá do vytvořeného vstupního prvku při jeho inicializaci. V případě, že vhodné mapování není nalezeno, tak se do vstupního prvku předává výchozí validátor, který při jakýchkoliv vstupních datech vrací kladnou odpověď. Tento krok je aplikován pouze na vstupní prvky, jelikož systém nepočítá s tím, že by se ze seznamů přímo upravovala data.

V dalším kroku se vstupním prvkům i sloupcům musí najít vhodný překlad jejich nadpisu. Ten dohledává InterfaceBuilder v LocalizationManageru, který poskládá výsledný název po průchodu všech použitelných mapování lokalizačních dynamických prvků. Na rozdíl od ostatních procesů pracujících s mapováním, které hledají pouze jedno nejvhodnější mapování, LocalizationManager pátrá napříč celým systémem a každé mapování, které jakkoliv vzdáleně odpovídá danému vstupnímu prvku či sloupci, zahrne do lokalizace. Poté prochází od těch nejvzdálenějších po ty nejvíce odpovídající a každého lokalizačního prvku se ptá na překlad názvu jeho atributu. Toto opatření je z důvodu požadavku podpory dědičnosti, jelikož umožňuje při překladu rozšiřující (dědící) třídy vynechat názvy jeho rodičovské třídy. A zároveň tento postup umožňuje konkretizovat pojmenování položek u zděděných tříd. Proces lokalizace zároveň bere v potaz různé případy užití.

Generování uživatelského rozhraní probíhá ve dvou úrovních. V první úrovni jádro vygeneruje příslušný formulář seznamu či detailu. Ten má, v obou případech, ve své bázové třídě definovánu druhou úroveň, která spočívá ve vygenerování odpovídajících vstupních či sloupcových dynamických prvků. Toto rozdělení umožňuje implementátorům větší volnost v generování obsahu formulářů a možnost kontroly nad tímto procesem.

5.3 Případy užití

Pojem případ užití (UseCase) souvisí s RUP metodikou vývoje softwarových produktů. Jedna z definic (Virius, 2006, s. 47) jej popisuje tak, že „případ užití je chápán jako funkce, kterou systém vykonává jménem jednotlivých účastníků nebo v jejich prospěch“. V kontextu informačních systémů jsou případy užití implementovány jako specifické scénáře odehrávající se na úrovni uživatelského rozhraní. Často jsou spojeny i s problematikou uživatelských oprávnění a přístupu k datům.

Případy užití jsou tedy podstatnou součástí informačních systémů. Proto byla tato poměrně komplexní problematika také zahrnuta do systému DynamicForms tak, aby bylo možné všechny dynamické prvky zacílit jen pro konkrétní případy užití. Díky obecnosti systému je tak možno definovat různé formuláře, vstupní prvky, sloupce, validátory a dokonce i lokalizace pro konkrétní případy užití.

Pro zjednodušení zápisu mapování byla také implementována možnost „skládání“ případů užití. To umožňuje definovat jedno mapování hned pro několik případů užití a také naopak definovat jeden složený případ užití a k němu najít mapování odpovídající alespoň jednomu ze skládaných mapování. Tento problém se podařilo poměrně elegantně vyřešit pomocí metody bitového maskování implementovaného binárním součinem (Herout, 2003, s. 255–256). Předpokladem pro jeho využití je ovšem definice hodnot jednotlivých případů užití v binární posloupnosti. Tedy například 1, 2, 4, 8, 16, ... s tím, že hodnota 0 je rezervována jako výchozí případ užití. V jazyce C# lze implementovat případy užití jako výčtový typ Enum, což velmi usnadní práci v rámci aplikace díky přehledné syntaxi a nativní podpoře pro operace bitového součtu. Například lze tedy zapsat definici mapování pro dva různé případy užití: (PřípadyUžití.Případ1 | PřípadyUžití.Případ2). Jediný problém, který s tímto systémem může potenciálně vyvstat, je omezení maximálního počtu případů užití. Ten je limitován velikostí (počtem bitů) použitého datového typu, který systém používá pro označení případu užití. V rámci této práce byl zvolen největší dostupný celočíselný typ, jež jazyk C# poskytuje. Tím je UInt64, neboli bezznaménkové 64bitové celé číslo. Ve výchozí implementaci tedy systém DynamicForms poskytuje prostor pro 64 různých případů užití. V případě potřeby jejich rozšíření by bylo zapotřebí definovat vlastní datový typ s větším rozsahem.

5.4 Implementace vlastních rozšíření

Při implementaci uživatelského rozhraní nad systémem DynamicForms se postupuje tak, že se postupně vytváří vlastní rozšíření systému, tzv. dynamické prvky, odvozené od příslušných bázových tříd. Bázové třídy poskytují vybrané metody svým odvozeným třídám pro přepsání (tzv. override), čímž jim umožňují vlastní implementace daných procesů. Mírným problémem je to, že implementátor musí znát bázové třídy, aby věděl, které metody poskytují pro přepsání, a proto tato kapitola poskytuje shrnutí všech možností a povinností implementátora při implementaci jednotlivých typů rozšíření nad bázovými třídami. Sloupec Typ reprezentuje rozlišení mezi metodami určenými pro přepis a metodami, které jsou pouze prostředky, jež bázová třída poskytuje. Také může obsahovat typ Událost. Sloupec povinná imple-

mentace značí, zda bázová třída vyžaduje implementování dané metody. V případě jejího neimplementování nebo zavolání bázové implementace dané metody generuje bázová třída výjimku.

Tabulka 2: Přehled bázových tříd

Název	Typ	Povinná	Popis
Forms.DynamicDetailBase			
Init	Přepis	NE	Inicializace formuláře, vytvoření dyn. prvků vstupu a jejich přidání na formulář voláním metody AddInput.
AddInput	Přepis	ANO	Přidání dyn. prvků na formulář.
AddInputForPlaceholder	Přepis	NE	Přidání dynamického prvku na místo určené tzv. placeholderem (viz níže). Ve výchozí implementaci přidává dyn. prvek na místo placeholderu a zobrazí jej.
Forms.DynamicListBase			
Init	Přepis	NE	Inicializace formuláře, vytvoření dyn. prvků sloupce a jejich přidání na formulář voláním metody AddColumn.
AddColumn	Přepis	ANO	Přidání sloupce do seznamu. Po přidaní všech sloupců volá báze metodu OnAllColumnsAdded.
AddRow	Přepis	ANO	Přidání řádku do seznamu. Po přídaní všech řádků volá báze metodu OnAllRowsAdded.
ClearRows	Přepis	ANO	Odstranění všech řádků seznamu.
OnAllColumnsAdded	Přepis	NE	Informace pro odvozené třídy, že sloupce byly přidány.
OnAllRowsAdded	Přepis	NE	Informace pro odvozené třídy, že řádky přidány.
FilterData	Prostř	–	Metoda umožňující vyhledávání v seznamu dat pomocí pokročilého filtrování, kompatibilního s češtinou.
EditItem	Prostř	–	Metoda umožňující upravit vybraný objekt.
ItemChanged	Udál.	–	Notifikace o změně objektu, po jeho úpravě m. EditItem
ItemBeginEdit	Udál.	–	Stornovatelná notifikace o počátku úpravy objektu. Lze využít například i pro výběr položky ze seznamu.
Inputs.DynamicInputBase			
Init	Přepis	NE	Inicializace dynamického prvku vstupu.
SetValue	Přepis	NE	Nastavení hodnoty dynamického prvku.
GetValue	Přepis	NE	Přečtení hodnoty z dynamického prvku.
ImplicitValidation	Přepis	NE	Vlastní validační mechanismus daného dynamického prvku, nezávislý na externích validátorech.
TitleVisibilityChanged	Přepis	NE	Informování odvozené třídy o změně viditelnosti jejího popisku. K té dochází při změně veřejného pole TitleIsVisible, které je definováno v bázové třídě DynamicInputBase.
Localization.DynamicLocalizationBase			
GetProperties	Přepis	NE	Metoda vracející překladový slovník pro názvy atributů.
GetClasses	Přepis	NE	Metoda vracející překladový slovník pro názvy tříd.
Validation.DynamicValidatorBase			
Validate	Přepis	NE	Implementace vlastního validačního mechanismu.

Při implementaci vlastních formulářů má implementátor zcela volnou ruku co se týče designu těchto formulářů, avšak co se týče jejich kódu jsou bázové třídy poměrně striktní. Z tabulky 2 vyplývá, že formuláře detailu a seznamu jsou jedinými bázovými třídami, které vyžadují implementování některých metod.

Při návrhu uživatelského rozhraní pro formuláře detailu poskytuje systém DynamicForms velmi zajímavou možnost definice rozložení jednotlivých dynamických prvků. Eliminuje tak potřebu programovat rozložení prvků v metodě AddInput. Jedná se o tzv. placeholdery, tedy konkrétně o třídu InputPlaceholder ze jmenného

podprostoru Inputs. Ta umožňuje definovat pozici pro vybrané dynamické prvky umístěním placeholderu na formulář. Tomuto placeholderu se následně přiřadí seznam polí, která může zastupovat. Tento seznam je implementován jako seznam řetězců, kde každá položka reprezentuje jeden název pole, případně i s jeho prioritou ve formátu „NázevPole:Priorita“. Placeholdery tedy umožňují aplikaci i pro více dynamických prvků vstupu. V nastavení placeholderu lze také definovat, zda na jednom formuláři zastupuje pouze jeden prvek, či může zastupovat více prvků. Ve výchozí implementaci formulářové báze se (v případě zástupu více prvků) tyto prvky řadí pod sebe.

5.5 Zabudovaná rozšíření, SDK

V původním návrhu systémové jádro DynamicForms implementovalo základní formuláře a dynamické prvky pro standardní datové typy (čísla, řetězce, datum a čas). To ovšem působilo matečně vzhledem k tomu, že se ve jmenných prostorech mísily bázové třídy společně s jejich implementacemi. Navíc jako každý vývojový framework by systém DynamicForms měl mí své SDK, neboli Software Development Kit. Proto bylo potřeba původní implementace dynamických prvků z jádra separovat do samostatného projektu dynamické knihovny DynamicForms.SDK. Tento krok vychází vstříc případným implementátorům informačních systémů nad platformou DynamicForms. Toto SDK urychluje pochopení systému DynamicForms tím, že se implementátor nemusí probírat poměrně komplexním ukázkovým projektem a vidí jen surové implementace bez zbytečných okras a pokročilých funkcí jako například případy užití.

Implementace obsažená v SDK obsahuje základní prostředky pro zobrazení generických dat nad základními datovými typy .NET frameworku. Obsahuje následující součásti:

Tabulka 3: Přehled tříd obsažených v SDK

Název	Jmenný podprostor	Datové typy
DefaultDetail	SDK.Forms	všechny[3]
DefaultList	SDK.Forms	všechny
TextInput	SDK.Inputs	řetězce, Číselné typy
CheckBoxInput	SDK.Inputs	logická hodnota (boolean)
DateInput	SDK.Inputs	datum a čas (DateTime, DateTime?)[4]
TextColumn	SDK.Columns	všechny
DefaultValidator	SDK.Validation	všechny

[3] Všechny datové typy reprezentuje mapování na datový typ System.Object, ze kterého vycházejí úplně všechny třídy jazyka C#.

[4] Otazník za názvem hodnotového datového typu značí jeho speciální implementaci v podobě třídy Nullable<T>, která umožňuje takto typovaným proměnným nabývat hodnoty null.

DefaultDetail je výchozí implementace formuláře pro detail datového objektu jakéhokoliv typu. Základem tohoto formuláře je ovládací prvek FlowLayoutPanel, který je součástí základních systémových tříd pro uživatelské rozhraní. Ten poměrně jednoduše zajišťuje rozmístění jednotlivých vstupních ovládacích prvků jejich postupným zařazováním pod sebe do sloupce. Přidávání vstupních ovládacích prvků je zajištěno přepisem bázové metody AddInput, případně AddInputForPlaceholder. Formulář DefaultDetail také implementátorovi ukazuje, jak řešit uložení formuláře pomocí bázové metody SaveItem.

DefaultList je výchozí implementace formuláře pro seznam datových objektů. Vychází ze systémového ovládacího prvku DataGridView, který reprezentuje tabulku dat s jednotlivými sloupci a řádky. Ukazuje implementátorovi základní práci s řádky a sloupci, která se implementuje přepisem bázových metod AddRow, AddColumn, OnAllRowsAdded a OnAllColumnsAdded. Práci s vestavěným filtrem dat, využitím metody FilterData, kterou poskytuje bázová třída. Editaci dat v tabulce pomocí metody báze EditItem.

Vstupní ovládací prvky ze jmenného podprostoru Inputs jsou všechny založeny na standardních systémových ovládacích prvcích (TextBox, CheckBox, DateTimePicker). Přepisují bázovou metodu Init, aby nastavily svůj popisek. Dále řeší nastavení dat a jejich zpětné přečtení. To je implementováno v přepisu bázových metod GetValue a SetValue. Prvek TextInput implementuje vlastní validační mechanismus, tzv. implicitní validaci (viz kapitola 5.4). Ta slouží pro ověření platnosti zadání v případě přiřazení k číselnému datovému typu.

TextColumn slouží jako dynamický prvek sloupce pro formuláře seznamu. Využívá opět systémovou třídu, a to DataGridViewTextBoxCell. Ta funguje jako prostá textová buňka v tabulce. To jí umožňuje zobrazit jakýkoliv datový typ pomocí jeho textové reprezentace.[5] Z tohoto důvodu je mapování pro TextColumn nastaveno pro všechny datové typy.

DefaultValidator je výchozí validační prostředek, který ukazuje, jak pomocí přepisu bázové metody Validate implementovat vlastní validační mechanismy. On sám ovšem při jakýchkoliv vstupních datech vrací kladný výsledek validace.

[5]Každý datový typ v jazyku C# musí implementovat metodu ToString, která vrací jeho textovou reprezentaci. To je zajištěno tím, že ji implementuje už třída System.Object.

6 Ukázkový IS

Pro ukázku práce se systémem DynamicForms bude uvedena jen částečná implementace informačního systému pro zadávání úkolů zaměstnancům a jejich zpětnou kontrolu. Tento systém musí demonstrovat vlastní implementace formulářů, vlastní ovládací prvky, využití validačních mechanismů a případů užití.

6.1 Popis

Projekt je rozdělen na dvě hlavní části. První částí je klientská aplikace TestProject. Druhou částí je dynamická knihovna vlastních rozšíření TestProject.Dynamics.

Klientská aplikace představuje základní uživatelské rozhraní, do kterého byla integrována i datová vrstva v podobě databázového systému Microsoft SQL Server Compact Edition a ORM systému EntityFramework. Tato datová vrstva by za standardních okolností byla separovaná v podobě nějakého aplikačního serveru, což ovšem z pohledu jádra DynamicForms nic nemění. Tato klientská část také řeší přihlašování a uživatelská práva, která aplikuje přes případy užití do uživatelského rozhraní DynamicForms. Datový model ukázkového systému je k dispozici v přílohách. Klientská aplikace pro své grafické rozhraní využívá volně šiřitelné knihovny Krypton Toolkit společnosti ComponentFactory, která poskytuje základní prvky uživatelského rozhraní jako např. tlačítka nebo textová pole.

Knihovna Dynamics obsahuje rozšíření systému DynamicForms v podobě vlastních implementací dynamických prvků. Ty jsou kvůli přehlednosti rozděleny do příslušných jmenných podprostorů a jim odpovídajících složek. Obsažené implementace jsou shrnuty v tabulce 4.

Tabulka 4: Přehled vlastních implementací v ukázkovém IS

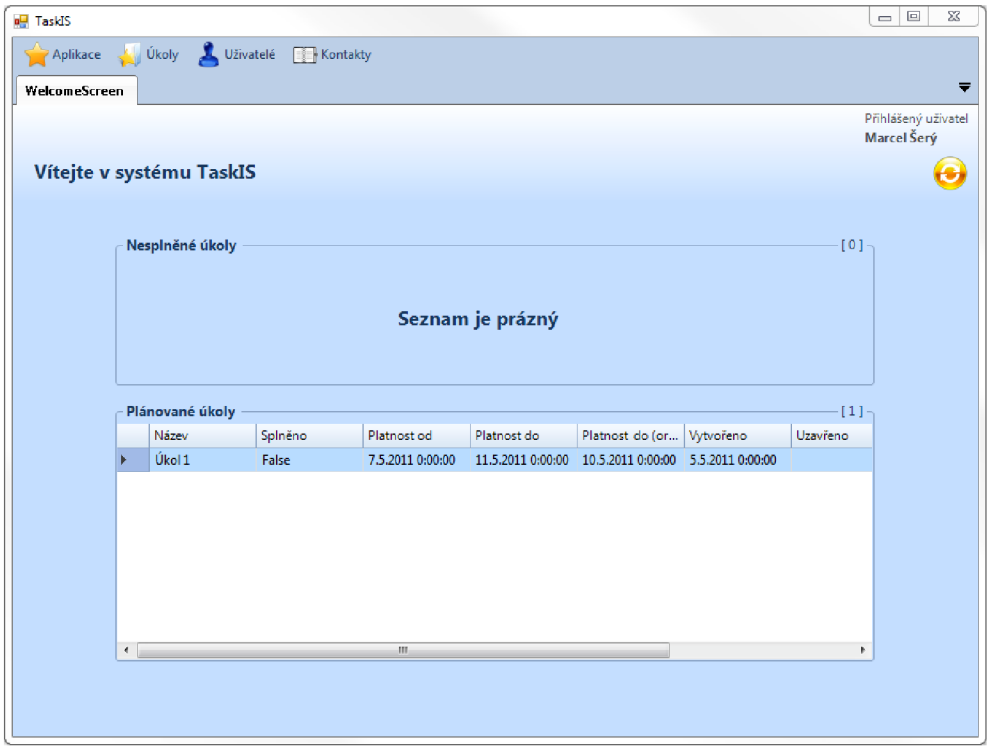
Název	Jmenný podp.	Popis
DetailForm	Forms	Základní formulář detailu.
ListForm	Forms	Základní formulář seznamu.
TextInput	Inputs	Textový vstup.
CheckBoxInput	Inputs	Zaškrtávací vstup.
DateInput	Inputs	Vstupní prvek pro datum.
UserInput	Inputs	Vstupní prvek pro datový typ Uživatel (viz dat. model)
HiddenInput	Inputs	Skrytý vstup. Pro omezení přístupu bez náležitých práv.
ReadOnlyInput	Inputs	Vstup umožňující pouze zobrazení hodnoty.
TextColumn	Columns	Sloupec textové hodnoty.
HiddenColumn	Columns	Skrytý sloupec. Pro omezení zobrazení bez uživ. práv.
EmailValidator	Validation	Validátor pro kontrolu formátu emailové adresy.
NonEmptyValidator	Validation	Validátor pro kontrolu neprázdného řetězce na vstupu.
Czech	Localization	Obecný slovník pro společné názvosloví v rámci ISu.
Czech_User	Localization	Slovník mapovaný ke třídě uživatel.
Czech_Address	Localization	Slovník mapovaný ke třídě adresa.

6.2 Výsledná aplikace

Jako nejefektivnější způsob prezentace výsledné aplikace byly zvoleny snímky jednotlivých obrazovek s doprovodným komentářem. Na těchto snímcích budou demonstrovány jak vlastní ovládací prvky, tak aplikování případů užití z pohledu uživatelských oprávnění.

Při spuštění se aplikace dotáže uživatele na jeho přihlašovací údaje, které posléze ověřuje vůči databázi uživatelů. Na této obrazovce není využit systém DynamicForms, proto není nutné ji uvádět. Je ovšem k nahlédnutí na přiloženém CD.

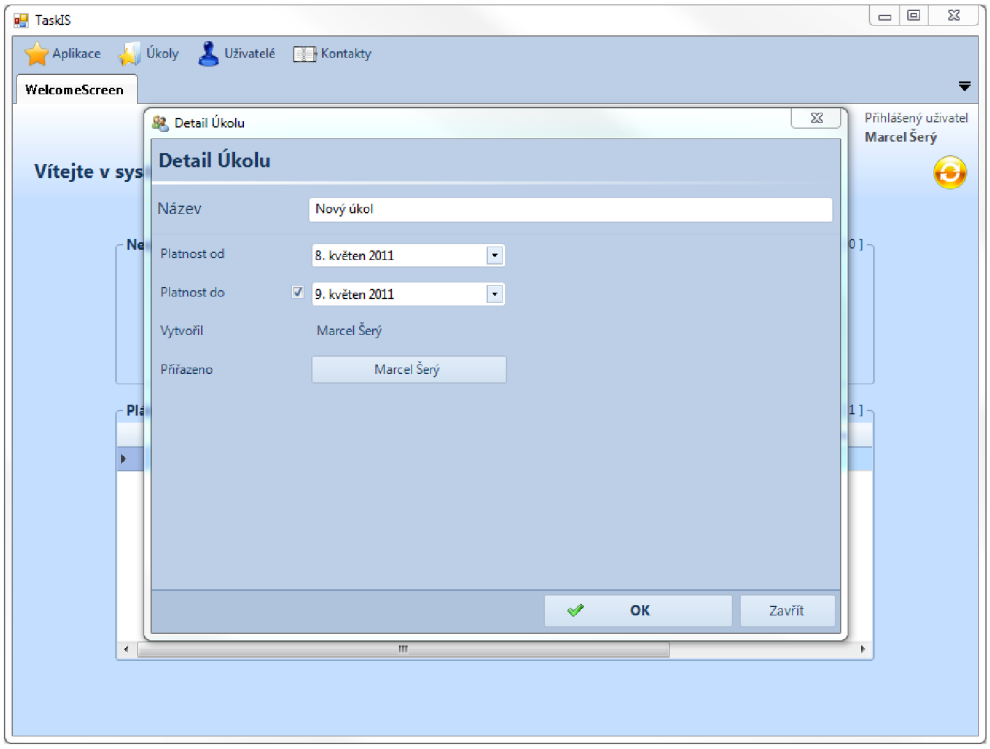
Po úspěšném přihlášení vidí uživatel tzv. uvítací obrazovku, kde se nachází přehled plánovaných a nesplněných úkolů (obrázek 4). V horní části obrazovky je vidět přihlášený uživatel a pod ním tlačítko pro nové načtení dat. Toto tlačítko má význam spíše dočasný, jelikož ukázkový systém neimplementuje notifikace o změnách mezi jednotlivými záložkami. Oba seznamy použité na této uvítací obrazovce jsou kompletně generovány systémem DynamicForms. Klientská aplikace jim pouze předává informaci o konkrétním případu užití TASK_OVERVIEW. Na ten už reaguje formulář seznamu ListForm skrytím horního panelu s vyhledáváním. Tato obrazovka je tedy kombinací standardně programovaného formuláře, do kterého jsou zasazeny formuláře dynamické.



Obrázek 4: Uvítací obrazovka

Nový úkol je možné vytvořit po vybrání položky „Nový úkol“ z hlavního menu Úkoly. Pro jeho vytvoření se uživateli otevře vygenerovaný formulář pro případ užití CREATING_TASK. Tento případ užití skrývá některá nepotřebná pole při vytváření nového úkolu jako například položky „Splněno“, originální „Platnost do“ či

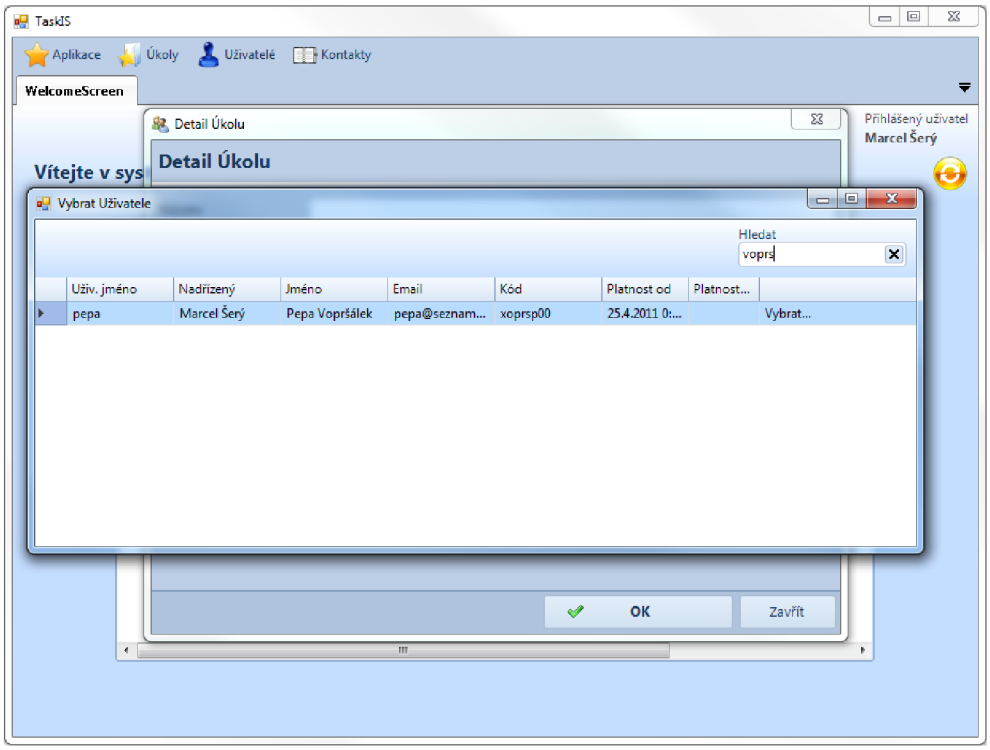
„Vytvořeno“. Pole „Vytvořil“ je pro tento případ užití zobrazeno dynamickým prvkem ReadOnlyInput, který ze zřejmých důvodů znemožní změnu uživatele, který úkol vytváří. Na obrázku 5 je také vidět využití placeholderů, kdy atribut „Název“ byl dán nad všechny ostatní atributy a byl roztažen přes celou šířku formuláře.



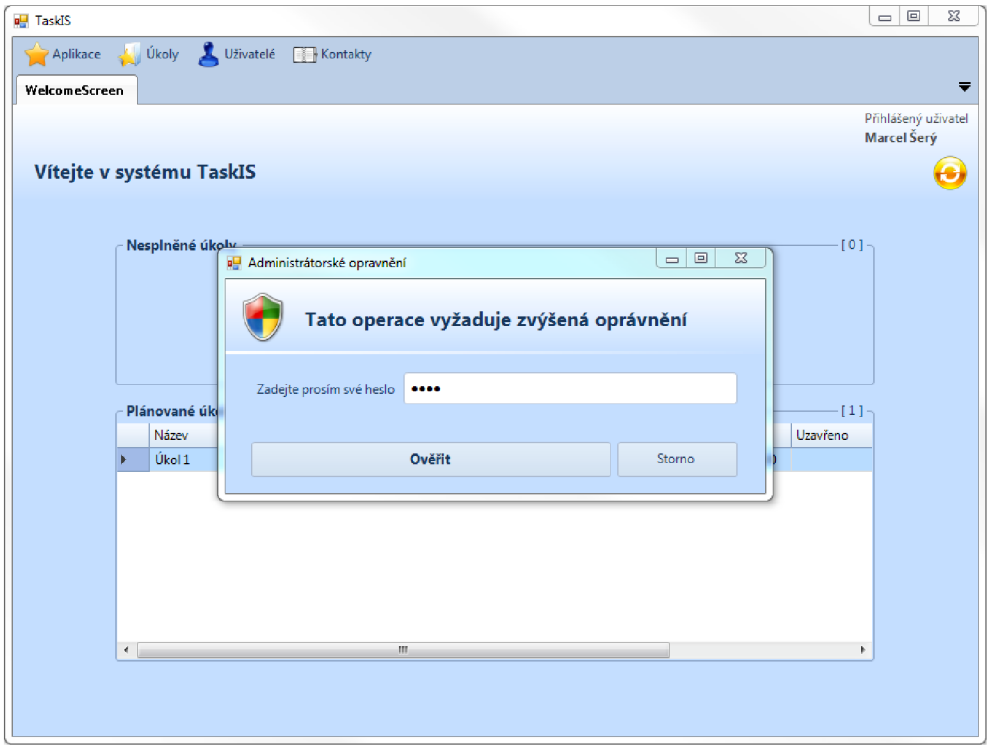
Obrázek 5: Nový úkol

Dalším demonstrovaným prvkem je již výše zmíněný dynamický prvek UserInput. Ten umožňuje pomocí dynamického formuláře seznamu vybrat, kterému uživateli bude úkol přiřazen. V této konkrétní implementaci k tomu využívá volání do klientské aplikace, která seznam naplní potřebnými daty. Avšak teoreticky je možné tuto logiku implementovat přímo do dynamického prvku UserInput vzhledem k tomu, že při implementaci tohoto prvku je znám jak celý zpracovávaný objekt, tak konkrétní případ užití. Na obrázku 6 je také vidět nativní filtrování dat v praxi. V horním panelu je rozepsaný hledaný výraz (tedy „Vopršálek“, bez diakritiky) a v seznamu již vyfiltrované položky. Výběr přiřazeného uživatele se provede buď dvojitým kliknutím myší na daný řádek, nebo obyčejným kliknutím na buňku s textem „Vybrat...“.

Další zajímavou problematikou jsou uživatelská oprávnění (obrázek 7). Ta byla vzhledem k čistě demonstrativnímu účelu tohoto systému řešena zjednodušenou formou. Tato zjednodušená forma spočívá v opětovném zadání přihlašovacího hesla aktuálně přihlášeného uživatele. Tento proces byl aplikován na položku „Seznam uživatelů“ v hlavním menu „Uživatelé“. V případě, že uživatel odmítne vyplnit heslo a vybere tlačítko „Storno“, zobrazí se základní seznam uživatelů, kde nemá přihlášený uživatel právo cokoliv měnit. Pokud se ovšem uživatele povede ověřit, zobrazí se pokročilý seznam uživatelů, ve kterém jsou vidět i jejich hesla.

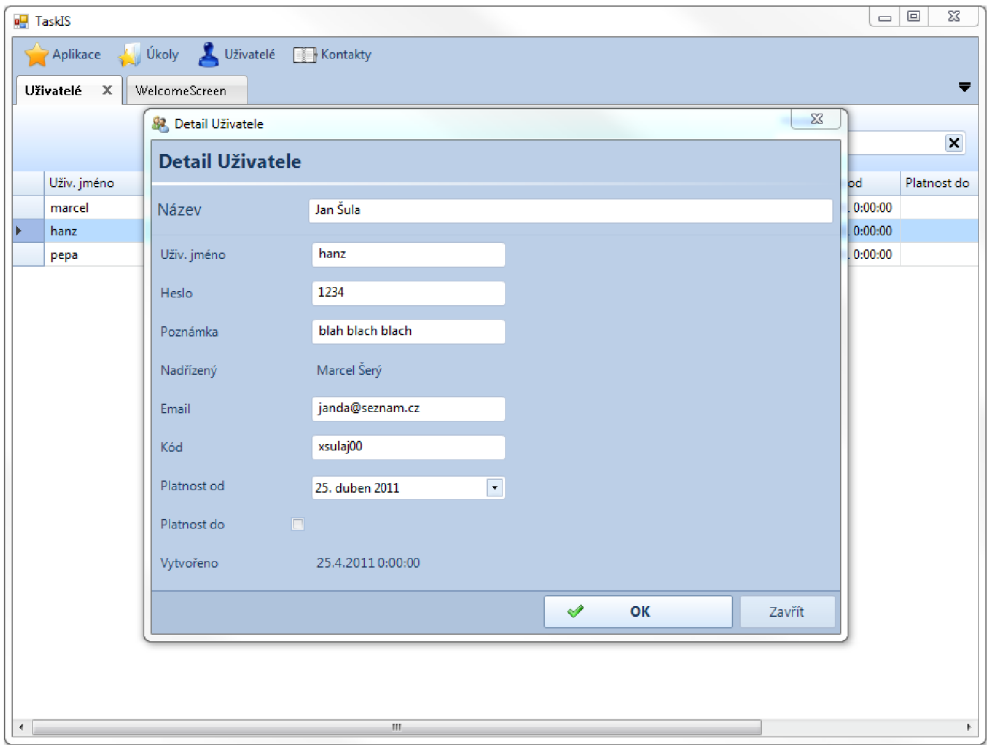


Obrázek 6: Výběr uživatele



Obrázek 7: Zvýšená oprávnění

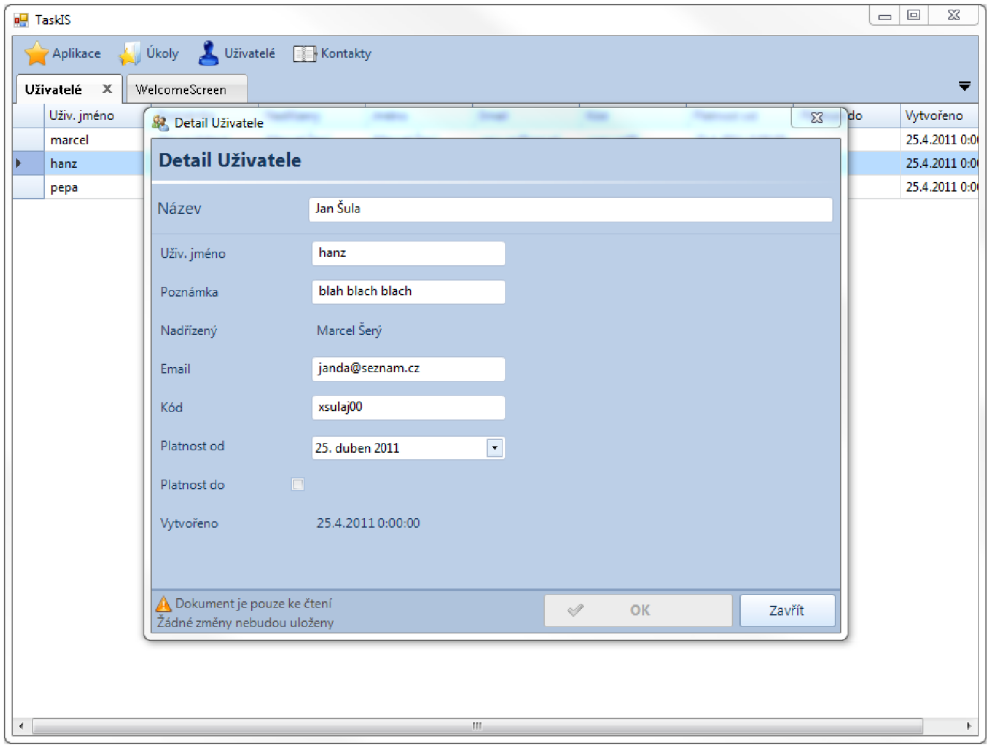
Formuláře detailu vyvolané z výše zmíněných seznamů se stejně jako samy seznamy odlišují. V případě zvýšených oprávnění formulář detailu umožní měnit atributy zobrazeného uživatele včetně jeho hesla a tyto změny posléze uložit (obrázek 8).



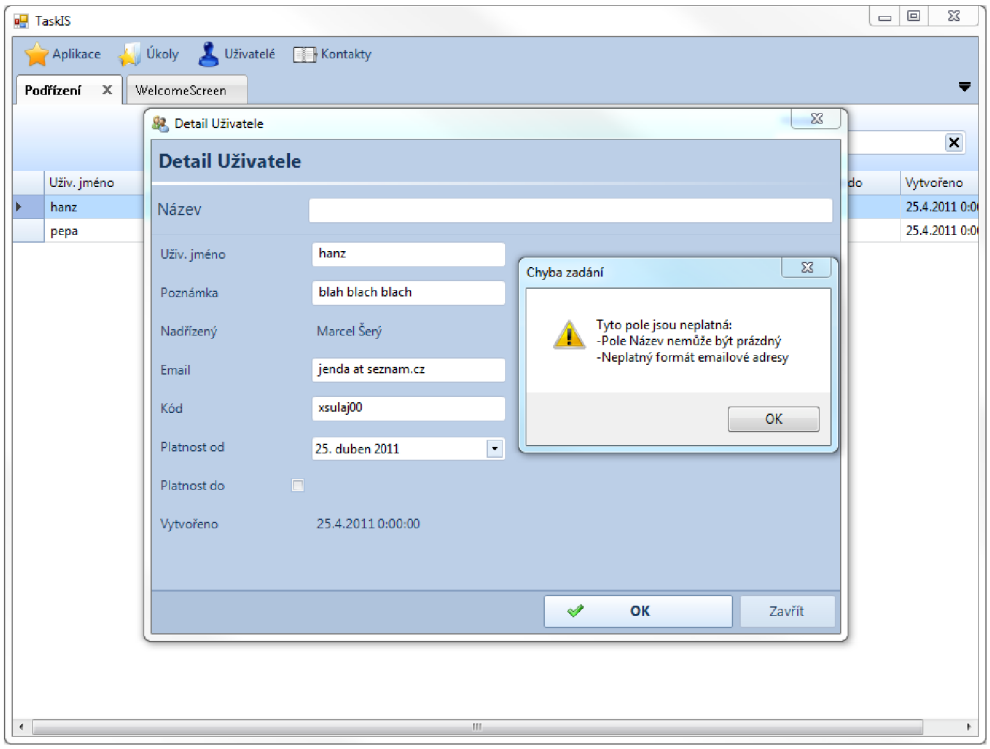
Obrázek 8: Detail uživatele se zvýšenými oprávněními

A naopak v případě základních oprávnění se vstupní prvek pro heslo vůbec nezobrazí a celý formulář detailu je pouze pro čtení. To je indikováno jednak vypnutým tlačítkem „OK“ a jednak varovným nápisem ve spodní části formuláře, jak je patrné z obrázku 9.

Poslední prvek systému DynamicForms, který zde bude prezentován, je funkce validátorů. Jejich výstup je v jádře implementován jako systémové hlášení, tzv. MessageBox, který uživatele upozorní na nevalidní položky. Na obrázku 10 je vidět využití NonEmptyValidatoru, který byl zmíněn v předchozím textu. Ten kontroluje název Úkolu a hlásí chybu, pokud není vyplněn.



Obrázek 9: Detail uživatele s běžnými oprávněními



Obrázek 10: Funkce validátorů

7 Zhodnocení

V průběhu vývoje byla objevena jediná významná slabina tohoto systému, a tou je jeho kaskádový efekt, který je paradoxně také jeho hlavní výhodou. Tedy i malá změna dynamických prvků na nízké úrovni může mít veliké dopady na vyšší úrovně. To může vývoj urychlit, ale zároveň způsobit komplikace např. u rozsáhlejších projektů. Jako preventivní opatření proti neúmyslným změnám v systému lze vidět důslednou dokumentaci aplikovaných mapovacích pravidel, dobrou komunikaci mezi členy implementátorského týmu a především kvalitní a důsledný návrh struktury implementovaných rozšíření.

Tento odstavec věnovaný možným rozšířením systému DynamicForms měl původně obsahovat návrh rozšíření systému o lokalizace a případy užití. Ty byly ovšem vzhledem k požadavkům ukázkového systému nutno zahrnout již do obsahu této práce. Jedno z témat, které by mohl systém DynamicForms do budoucna řešit, by mohla být rychlost reflexe. Ta je na platformě .NET kamenem úrazu mnoha systémů, jelikož pro složitější systémy může být velmi náročná. Zajímavým rozšířením by tedy mohla být nějaká optimalizace tohoto procesu. Například při reflexi použít více vláken, a tedy naplno využít možnosti víceprocesorových systémů.

Také by mohla být vítaným zlepšením systému pokročilá mezipaměť pro celé vygenerované formuláře. Ty se ve stávající implementaci při každém dotazu nově generují. S tím by ovšem vyvstala potřeba komplexního správce této mezipaměti, který by byl schopný pružně reagovat na aktuální paměťové prostředky a nepoužívané formuláře případně odstraňovat.

Pokud jde o komerční využití systému DynamicForms, tak po jeho zdárné implementaci může být pohled na něj mírně skeptický. Hlavním problém je ve zcela novém přístupu k návrhu uživatelského rozhraní, který vyžaduje od implementátora notnou dávku abstrakce, zkušeností a hlavně zaškolení pro implementování systémů na platformě DynamicForms. Nicméně i přes toto všechno je pravděpodobné, že projekt DynamicForms by si v komerční sféře své místo našel. Jednalo by se především menší projekty, v rozsahu 2–3 implementátorů s dobrým vedením a kvalitním návrhem. V takové sestavě dokáže systém DynamicForms velmi zrychlit celkový vývoj a následnou údržbu. Problém by ovšem mohl nastat v případě, kdy by měl implementaci na starost dvacetičlenný tým s chabým vedením.

Pro komerční využití systému DynamicForms lze vidět jako zásadní požadavek propracovat ještě hlouběji problematiku validačních mechanismů a možnosti designování formulářů. Zajímavým rozšířením by byl například vlastní designer pro formuláře.

Dalším zásadním požadavkem pro komerční nasazení je zaintegrování jádra DynamicForms do budovaného informačního systému, a tím pádem i posílení vnější interakce generovaných formulářů a ovládacích prvků. V ukázkovém systému je již nástin jednoho z možných rozšíření zanesen. Tím je třída FormConfig umožňující definovat vlastní tlačítka určené pro horní panel formuláře seznamu. Každé toto tlačítko má svůj obrázek, popis a hlavně handler události kliknutí, který by spouštěl definovanou aplikační logiku v klientské aplikaci. Do praxe v ukázkovém projektu nebyl uveden.

U rozsáhlejších systémů by také mohl nastat problém s omezeným počtem možných případů užití. Ty jsou v aktuální implementaci limitovány počtem 63 vlastních případů užití a jejich kombinacemi. Možné řešení tohoto problému bylo nastíněno již v kapitole 5.3. Tento hendikep by rovněž mohl kompenzovat kvalitní návrh případů užití tak, aby se daly dobře kombinovat a pracovat posléze s těmito kombinacemi. Krajním řešením by bylo přeprogramovat jádro tak, aby neřešilo skládané případy užití a řídilo se pouze číselným vyjádřením. Tím by se zpřístupnilo teoreticky až $2^{64} - 1$ případů užití, avšak systém by tímto krokem přišel o poměrně podstatnou a užitečnou část.

8 Závěr

Tato práce si stanovila za cíl vytvořit jeden z pilířů platformy umožňující rychlý vývoj informačních systémů, které jsou do jisté míry nezávislé na aplikovaném datovém modelu. Přinesla také zcela nový přístup k dynamickému uživatelskému rozhraní, který snad poskytne jistý nadhled při dalším rozvoji této problematiky.

Požadavky a očekávání od systému DynamicForms byly zpočátku jednoduché. To se ovšem změnilo s postupnou implementací ukázkového systému. Při implementování této demonstrace došlo k rozšíření původních požadavků o případy užití a validační mechanismy, což ve výsledku znamenalo odhadem 30% nárůst implementační náročnosti jádra systému. Také bylo zajímavé v průběhu vývoje systému sledovat, jak se mění jeho struktura v souvislosti s přidáváním dalších funkcionalit a nárůstu komplexity systému. Došlo ke strukturálním změnám v oblasti práce s mapováním, generováním uživatelského rozhraní a k celkové optimalizaci rozložení funkcionality mezi jednotlivé třídy.

I přes neočekávaný nárůst rozsahu se však podařilo systém implementovat se splněním všech stanovených požadavků a výsledkem je funkční, dostatečně obecná a flexibilní platforma pro využití v informačních systémech. Výsledek této práce by mohl mít potenciál i při využití v komerčním sektoru.

Jako největší přínos této práce lze uvést především demonstraci nových přístupů a myšlenek v oblasti informačních systémů. Na toto téma bude v budoucnu navazovat diplomová práce, která jej bude dále rozšiřovat a více přibližovat reálným a praktickým požadavkům komerční sféry.

9 Literatura

Herout, P. *Učebnice jazyka C, 1. díl.* 2003. 265 s. ISBN 978-80-7232-383-8.

Křena, B., Kočí, R. *Studijní opora předmětu IUS.* 2010. 103 s.

MacDonald, M., Freeman, A., Szpuszta, M. *Pro ASP.Net 4 in C# 2011.* New York: Apress, 2010. ISBN 14-302-2529-7.

Pokorný, J. *Databázové systémy a jejich použití v informačních systémech.* Praha: Academia, 1992. 313 s., ISBN 80-200-0177-8.

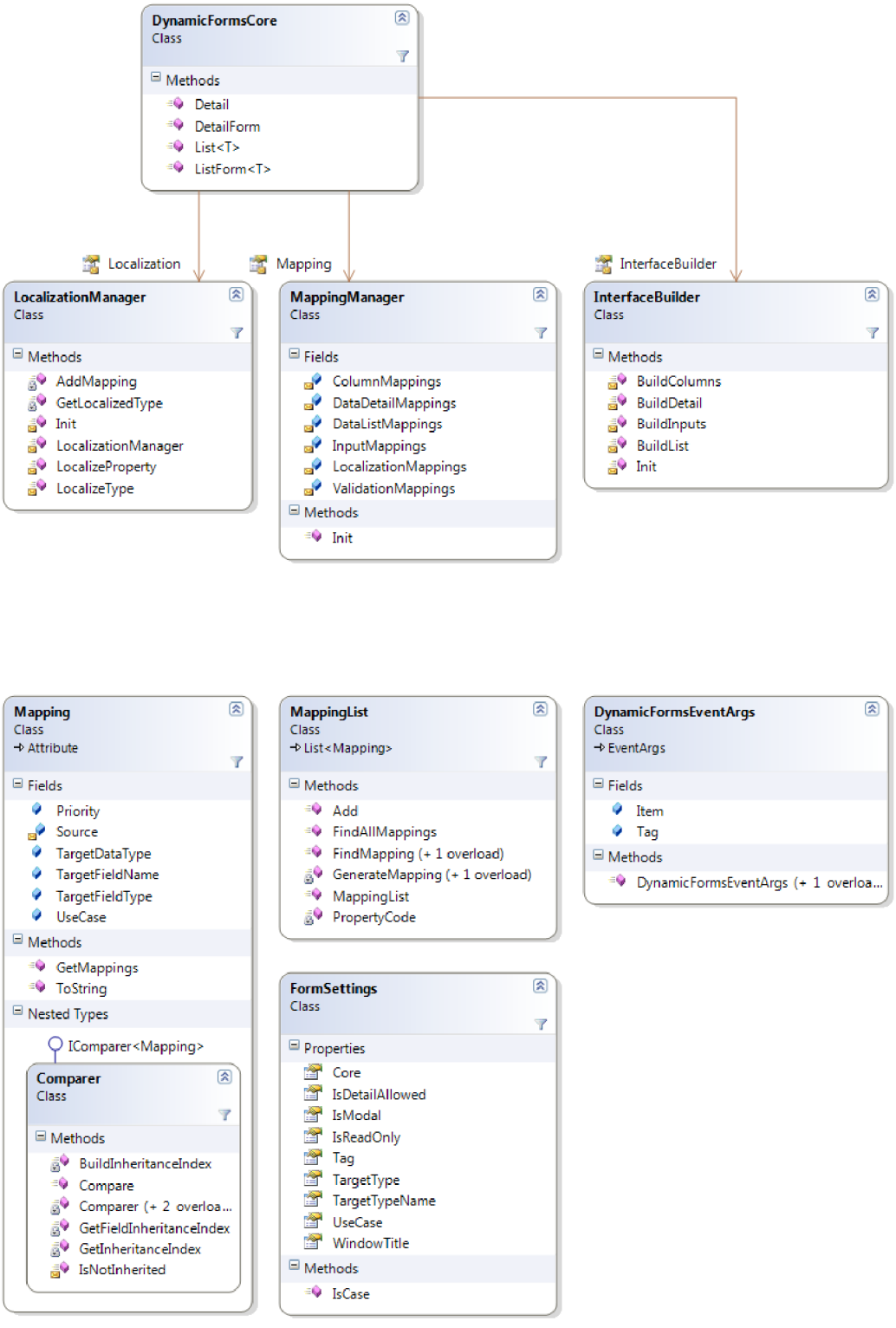
Virius, M. *C# Hotová řešení.* Brno: Computer Press, a. s., 2006. 341 s., ISBN 80-251-1084-2.

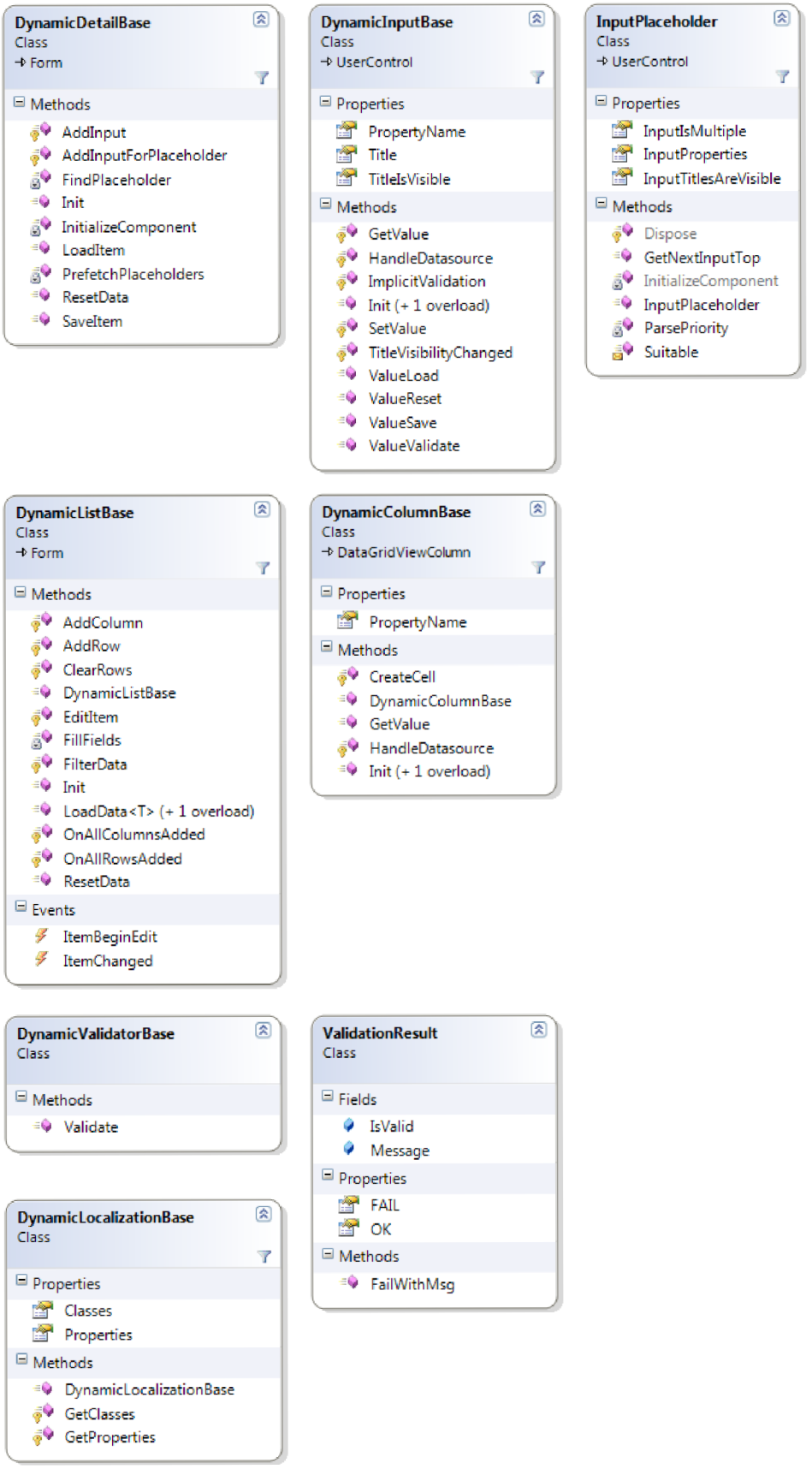
Seznam obrázků

Seznam tabulek

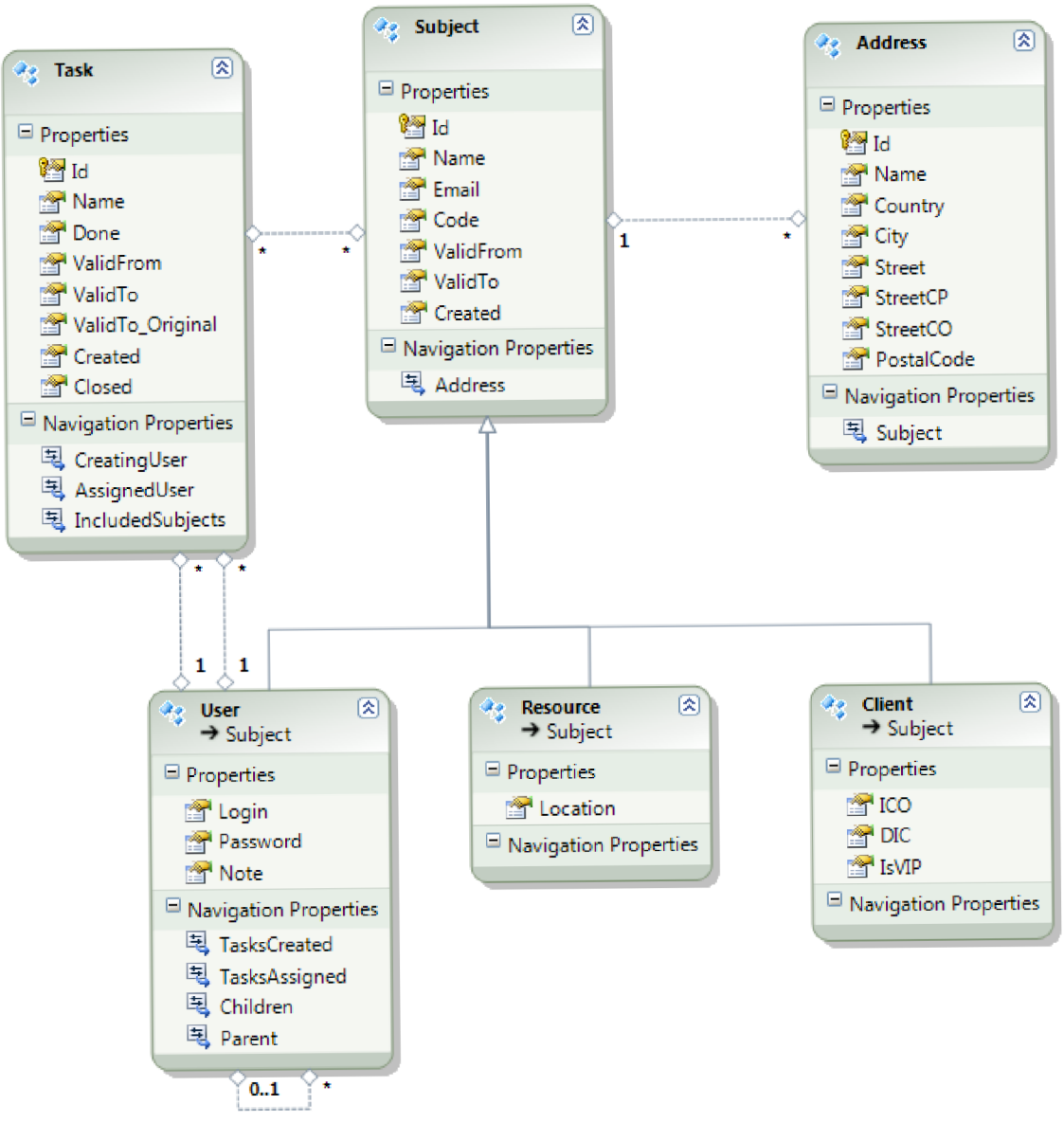
Přílohy

A Diagram tříd DynamicForms




DynamicDetailBase
Class
Form
Methods
AddInput
AddInputForPlaceholder
FindPlaceholder
Init
InitializeComponent
LoadItem
PrefetchPlaceholders
ResetData
SaveItem
DynamicInputBase
Class
UserControl
Properties
PropertyName
Title
TitleIsVisible
Methods
GetValue
HandleDatasource
ImplicitValidation
Init (+ 1 overload)
SetValue
TitleVisibilityChanged
ValueLoad
ValueReset
ValueSave
ValueValidate
InputPlaceholder
Class
UserControl
Properties
InputIsMultiple
InputProperties
InputTitlesAreVisible
Methods
Dispose
GetNextInputTop
InitializeComponent
InputPlaceholder
ParsePriority
Suitable
DynamicListBase
Class
Form
Methods
AddColumn
AddRow
ClearRows
DynamicListBase
EditItem
FillFields
FilterData
Init
LoadData<T> (+ 1 overload)
OnAllColumnsAdded
OnAllRowsAdded
ResetData
Events
ItemBeginEdit
ItemChanged
DynamicColumnBase
Class
DataGridViewColumn
Properties
PropertyName
Methods
CreateCell
DynamicColumnBase
GetValue
HandleDatasource
Init (+ 1 overload)
DynamicValidatorBase
Class
Methods
Validate
ValidationResult
Class
Fields
IsValid
Message
Properties
FAIL
OK
Methods
FailWithMsg
DynamicLocalizationBase
Class
Properties
Classes
Properties
Methods
DynamicLocalizationBase
GetClasses
GetProperties

B Diagram databázového modelu ukázkového systému



C Metriky kódu

Projekt	Počet souborů	Počet řádků zdrojového kódu
DynamicForms.Core	24	2 003
DynamicForms.SDK	13	896
DynamicForms.TestProject	24	4 510
DynamicForms.TestProject.Dynamics	26	1 868
Celkem	87	9 277

D CD/DVD

K bakalářské práci je přiloženo CD se zdrojovými soubory informačního systému, jehož struktura je následující:

`/manual.txt` – stručný přehled obsahu CD
`/Aplikace/` – složka obsahující zdrojové kódy
`/Demo/` – složka se zkompilovanou aplikací sloužící pro demonstrační účely
`/Zprava/Obrazky.zip` – obrazové materiály obsažené v této práci
`/Zprava/Technická zpráva.pdf` – elektronická podoba tohoto dokumentu