TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky a mezioborových inženýrských studií

Studijní program: M2612 – Elektrotechnika a informatika
Studijní obor: 1802T077 – Informační technologie

Návrh komponent a tříd pro použití ve WYSIWYG grafických editorech

Designing components and classes for using in WYSIWYG graphics editors

Diplomová práce

Autor: **Bc. Vladimír Ostatek**
Vedoucí práce: Ing. Jaroslav Buchta
Konzultant: Ing. Martin Fian

V Liberci 14.05.2008

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé diplomové práce a prohlašuji, že souhlasím s případným užitím mé diplomové práce (prodej, zapůjčení apod.).

Jsem si vědom toho, že užít své diplomové práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce a konzultantem.

Datum:

Podpis:

Poděkování

Rád bych poděkoval všem, kteří mi radou či skutkem pomohli k vypracování tohoto textu, zejména vedoucímu Ing. Jaroslavu Buchtovi za jeho shovívavost a Ing. Martinu Fianovi za spoustu užitečných nápadů a postřehů. Dále bych chtěl poděkovat svým rodičům za podporu, zázemí a lásku. Jejich důvěra mi vždy bude pomáhat k dosažení mých snů a cílů.

Abstrakt

Tato práce se zabývá vývojem graficky orientovaných aplikací. Teoretická část se věnuje objektově orientovanému programování, vytváření vlastních komponent, způsobům kreslení a překreslovacích operací ve vývojovém prostředí Delphi společně s postupy a problémy vycházejících z vlastností operačního systému Windows. Analytická část práce popisuje výběr správných tříd a podpůrných nástrojů pro vlastní realizaci komponent a tříd. Praktická část blíže popisuje navržené komponenty a jejich možné využití v různých aplikacích.

Abstract

This graduation thesis forms the introduction to development of graphical-oriented applications. Theoretical part considers object-oriented programming, developing new components, different ways of drawing and redrawing operations in development environment Delphi, as well as operations and problems made by OS Windows. Analytical part of thesis consists of selection right classes and tools for components and classes realization. Practical part closer describe designed components and theirs usage in different applications.

Klíčová slova

Borland Delphi 2006, Objektově orientované programování, Grafické objekty komponenty, Canvas, Překreslování

Keywords

Borland Delphi 2006, Object oriented programming, Graphic objects, Components, Canvas, Repainting

Obsah

Seznam obrázků

1 Úvod

Počítač je dnes nedílnou součástí jakékoliv firmy i většiny domácností. Časy, kdy se psala většina formálních dokumentů na psacím stroji a obrázky se kreslili ručně nebo vznikali na kreslících prknech, jsou nenávratně pryč. Správa uživatelských dat na počítači sebou nese množství výhod, opravou chyb a jednoduchou editací a distribucí počínaje, elektronickou archivací dat, která odbourává neskladné papírové archivy, konče.

Elektronické dokumenty budete s největší pravděpodobností vytvářet v některém z dostupných editorů. Celá tato práce se bude zabývat jednou třídou editorů z této skupiny a to WYSIWYG grafickými editory. WYSIWYG je akronym anglické věty „What you see is what you get“, česky „co vidíš, to dostaneš“. Tato zkratka označuje způsob editace dokumentů v počítači, při kterém je verze zobrazená na obrazovce vzhledově totožná s výslednou verzí dokumentu.

V tomto okamžiku však také narážíme na problematiku vývoje softwaru nebo jeho částí. Ať už vyvíjíte jakýkoliv systém či jeho část, prochází tento vývoj určitou posloupností, činností a proto by měl být celý proces uvědomělý a metodický. Cílem je vytvoření komponent a tříd použitelných v těchto editorech. Celou toto rozsáhlou strukturu jsem se snažil rozdělit do menších logických celků. Nic by tedy nemělo bránit případnému použití menších částí na základě granularizace celého řešení.

Hlavním cílem je vytvořit grafickou komponentu, která bude mít všechny vlastnosti plnohodnotného vektorového editoru, který dokáže spravovat grafické objekty. Komponent musí podporovat funkcionalitu, na kterou jsme zvyklí z běžných editorů, dále export do některých z obecně podporovaných grafických formátů, a tím zajistit její použitelnost, a dostatečnou pružnost pro další využívání v různorodých aplikacích. Celkové struktuře, konceptu a problematikám pří návrhu se budu věnovat později v této práci.

Detailní popis jednotlivých komponent, tříd a metod, které nabízejí různou funkcionalitu, je nad rámec daného rozsahu, a proto se chci zaměřit na stěžejní problémy, se kterými jsem se potýkal. Hovořit tedy budu nejen o samotném grafickém

gridu, jeho objektech, ale i komponentách a třídách, které slouží pro vytváření tabulek, textovém editoru formátu RTF, nebo linkování databázových dat nad libovolnou databázovou strukturou.

Komponenty v menší či větší míře využívám v některých svých aplikací. Jako ukázkový program jsem vybral aplikaci PaperPalmManager, kde je zastoupení nejširší. Budu v ní také demonstrovat i jednotlivé ukázky.

2 Úvod do objektově orientovaného programování

V několika málo odstavcích se, velice stručně, pokusím vysvětlit alespoň základní pojmy a principy objektově orientovaného programování. Omluvte mou strohost, člověk znalí věci se zde jistě nic nového nedozví, ale případnému méně zkušenému čtenáři osvětlím termíny, které se v celé práci budou hojně vyskytovat.

Třídy a objekty

Třída je předpis, podle kterého se vytváří příslušný objekt. Třída je definovaný typ, který má nějaký stav reprezentaci a chování. Obsahuje své atributy a metody. Atributy jsou proměnné, které si potřebujeme pomatovat. Metody jsou činnosti, které můžeme u objektu volat a jsou zapsány ve formě procedur a funkcí.

Mezi třídami a objekty často dochází k záměnám. Takže pro úplnost objekt je instance třídy. Třídu píše programátor, na rozdíl od objektu, který se vytváří za běhu aplikace a je uložen v operační paměti. Třída je jedna, instancí jedné třídy může být více. Třídy po vypnutí počítače zůstávají uloženy na disku, objekty jsou z paměti uvolněny, nebudeme li uvažovat o některých speciálních případech.

Abstrakce

Můžeme abstrahovat některé nepodstatné vlastnosti objektu a zaměříme se na rysy podstatné, které mají být součástí objektu. Objekt pracuje jako černá skříňka, programátor, potažmo program využívá funkcionalitu objektu, aniž by musel znát způsob, jakým objekt vnitřně pracuje.

Zapouzdření

Každá třída může obsahovat nespočet atributů a metod. Pro správný objektově orientovaný návrh by měla být data skryta (zapouzdřena) uvnitř třídy. Přístup k nim by měl být realizován pomocí rozhraní objektu, tak, aby nedošlo k nekonzistenci. Těmto účelům poslouží specifikátory přístupu:

Private - Tímto klíčovým slovem označujeme atributy a metody třídy, které jsou dostupné pouze zevnitř jednotky (zdrojového souboru).

Public - Tato deklarace umožňuje použití z libovolné části kódu. Volání je možné uvnitř jednotky, všem potomkům třídy i části kódu, který vytváří instanci objektu.

Protected - Jedná se částečně chráněné atributy a metody, použití je možně současnou třídou a všemi jejími potomky.

Published - Tyto atributy a ukazatele na metody jsou dostupné uvnitř jednotky, ale i ve všech ostatních částech kódu. Ve vývojovém prostředí Delphi jsou položky označené jako Published, dostupné v Object Inspectoru a můžeme měnit jejich hodnoty již v době návrhu aplikace.

Dědičnost

Při psaní kódu se často stává, že potřebujeme vytvořit novou třídu, která bude mít podobné vlastnosti jako nějaká existující třída. Nejlepším postupem jak toho dosáhnout je deklarovat si novou třídu, a jako předka uvést třídu, jejíž funkcionalita nám nejvíce vyhovuje. Máme tak definovanou vlastní třídu, která zdědila veškeré atributy a metody rodičovské třídy, a můžeme ji následně libovolně rozšiřovat.

Polymorfismus

Funkce a procedury jsou založeny na takzvané statické vazbě. To znamená, že v době překladu programu je známo místo v paměti, ve kterém je daná procedura nebo funkce uložena. V Object Pascalu můžeme definovat dynamickou (pozdní) vazbu. V tomto případě je adresa volané metody určována až za běhu programu, podle typu příslušného objektu. Tato vlastnost je známa jako polymorfismus.

3 Vývojové prostředí

Jedním z prvních kroků pro úspěšný vývoj aplikace, nebo jako je tomu v tomto případě komponent a tříd je uvědomění si požadavků a kritérií, které si na ně kladete vy, popřípadě zadavatel. Jsou to důležité otázky, protože jejich zodpovězení je nutné pro správný výběr vhodného vývojového prostředí. Tímto krokem si určíte hranice a omezení, za které se budete jenom těžko dostávat. Já jsem zvolil vývojové prostředí od firmy Borland konkrétně Borland Delphi 2006.

Co je to vlastně Delphi? Jedná se o vývojový nástroj, který umožňuje jednoduše vytvářet a navrhovat aplikace pod operačním systémem Windows. Toto vývojové prostředí je založeno na jazyce Object Pascal, jedná se o rozšíření Turbo Pascalu o podporu objektů. V Object Pascalu bude zapsán veškerý výkonný kód aplikace a v prostředí Delphi budeme navrhovat a testovat vizuální vzhled.

V souvislosti s Delphi se také často setkáme s pojmem vizuální programování. Jedná se o postup, při kterém se na formulář v době návrhu pomocí myši nebo klávesnice umísťují jednotlivé komponenty a tím vytváříme požadovaný vzhled aplikace. Skládáním komponent vlastně vytváříme uspořádání a design konkrétního formuláře.

Osobně tento koncept považuji za velmi zdařilý, protože vám ušetří spoustu času, výsledný formulář vidíte již v době návrhu a nestrávíte spoustu času psaním kódu, který formulář vygeneruje. Atributy ovládacích prvků můžete také pohodlně měnit již při návrhu aplikace. Uživatelům, kteří se s prostředím teprve seznamují, výrazně pomůže.

Využití objektově orientovaného programování však bývá v Delphi často podceňovanou skutečností. V době návrhu za vás prostředí totiž udělá hodně práce samo. Delphi automaticky definují třídu pro každý nový formulář nebo jakoukoliv jinou komponentu umístěnou do formuláře nebo na jiný ovládací prvek. V nástroji Object Inspektor můžete změnit atributy označených objektů nebo na záložce events definujete metody k různým událostem, na které má objekt reagovat. Delphi opět automaticky vytvoří tělo metody a ukazatel na metodu, vy jenom napíšete kód. Bohužel spousta uživatelů si tyto skryté akce neuvědomuje a proto je pro ně obtížné využít potenciální možnosti tohoto prostředí, které je založeno na komponentách a definici nových typů tříd.

4 Komponenty

Každý uživatel Delphi má nějaké zkušenost s použitím již existujících komponent firmy Borland. Najdete je na záložkách v okně nástrojů jménem Tool Palette. Někdy může být užitečné vytvořit si komponenty vlastní nebo přizpůsobit a upravit chování již některých z existujících komponent.

Instalace nových komponent je v Delphi vcelku jednoduchá záležitost, ale jejich vývoj to je jiná kapitola. Jen stěží se obejdete bez detailních znalostí jazyka Object Pascal, důležité je pochopení konceptů jako je dědičnost, potlačování a přetěžování metod, specifikací přístupu, definice vlastních událostí.

Stěžejní je také prostudování hierarchie tříd VCL (Visual Component Library), která je důležitá pro správný výběr třídy, od které bude vaše budoucí komponenta dědit. Zapomeňte také na vizuální programování, to vám v tuto chvíli nemůže být k dopomoci, celou definici třídy i metod si musíte napsat sami.

4.1 Obecné zásady pro vývoj komponent

Respektujte zavedené konvence, pro pojmenování metod, atributů, i pro psaní samotného kódu. Bude to jednodušší pro vás, protože nemusíte vymýšlet smysluplné názvy metod, které již implementují některé jiné komponenty, ale hlavně pro uživatele, kteří se nebudou muset s rozhraním vašich komponent detailně seznamovat.

Komponenty vytvářejte jednoduché. Pokud vytváříte velkou a komplexní komponentu, rozdělte ji na menší třídy, které mezi sebou navzájem dědí. Tímto způsoben posupně rozšiřujete základní komponentu, až do konečné podoby přidáváním dalších metod a atributů. Váš kód se tak stane přehlednější, můžete lépe identifikovat jednotlivé milníky a zvýší se také znovupoužitelnost vašich komponent. Je lepší mít možnost dědit od některého z předků, než jenom od finální komponenty.

Veškerou alokaci zdrojů provádějte v chráněném bloku try catch. Pokud se něco nepovede, důvodů může být nespočet, měli bychom uvolnit všechny alokované zdroje, popřípadě i vygenerovat chybové hlášení.

Stále používám slova komponenta a třída, přičemž by se mohlo zdát, že se jedná o synonyma. Abych vše uvedl na prvou míru, komponentou můžeme nazývat takovou třídu, která má za předka třídu TComponent. Novou komponentu je tak možné odvodit od jakékoliv existující komponenty nebo od komponenty nevizuální, která neodpovídá žádné již existující komponentě.

4.2 Rozdělení komponent

Komponenty rozdělujeme do tří základních skupin. Každou komponentu, kterou jsme naprogramovali, naprogramujeme nebo která je již součástí vývojového prostředí, zapadne do některé z těchto kategorií.

Třída **TWinControl** je rodičovskou třídou každé komponenty, která je umístěna v okně, má svůj handle, můžeme jí posílat zprávy a může se stát aktivním oknem. Pro vytvoření komponenty odvozené od těchto tříd, nejčastěji použijeme její variantu začínající TCustom, za kterým následuje název třídy. Takže má li být rodičovská třída TEdite použijeme třídu TCustomEdit. Samozřejmě můžeme použít u třídu TEdit, ale připravili bychom se tak o možnost skrývat některé metody a atributy.

Třída **TGraphicControl** je základem vizuálních komponent, nemůže se stát aktivním oknem, a také nemůže být přímým adresátem zpráv zasílaných v systému. Objekty, které jsou instancí této třídy, nemají handle a šetří tak systémové zdroje. Pro vytváření nových grafických komponent nejčastěji použijeme jako předka Třídu TGraphicControl.

Třída **TControl** je rodičovskou třídou všech komponent, a její použití je vhodné pro komponenty které nemají grafickou složku. V době návrhu je jejich přítomnost na formuláři indikována ikonou, přístup k atributům objektu a jeho událostem se nijak neliší od ostatních komponent.

5 Definice vlastních komponent

Jak se komponenty vytvářejí a jaké při tom dodržovat zásady bylo obsahem předchozí kapitoly. Vytvořit takto velký balík komponent není možné provést bez důkladného zvážení všech možností. Celému projektu by měla předcházet počáteční studie, specifikace požadavků a vytvoření modelu tříd, popřípadě funkčního modelu. V našem případě je prvním krokem naprogramování grafického gridu. Je to ústřední komponenta, srdce celého balíku a některé další třídy jsou s ním asociovány. Tato komponenta se stará o vytváření, správu, uchovávání, ukládání, načítání a rušení grafických objektů. Také určuje plochu, v rámci které budeme kreslení obsahu realizovat. Jeho nedílnou součástí jsou sady metod různorodé funkcionality. Nebudu už předbíhat a pojďme se podívat na jednotlivé komponenty a jejich možnosti.

5.1 Bázový grid

Je bázovou třídou a slouží tedy jako základní třída všem odvozeným třídám grafických gridů. Je potomkem třídy TGraphicControl, která obsahuje rozličné virtuální metody. Tyto metody je tedy možné přetěžovat, což nám ušetří spoustu práce, nemusíme sami implementovat metody pro překreslování, změnu velikosti, akce spojené s pohybem myší, aktivací, deaktivací komponenty a mnoho dalších.

Spadá do třídy grafických ovládacích prvků systému Windows. Obsahuje tedy plátno Canvas, na které jsou vykreslovány grafické elementy. Je nutné mít na paměti, že obsah je pouze vykreslen na obrazovku a nedochází k jeho automatickému uchovávání v paměti. Důvodem je šetření systémových zdrojů. Z hlediska systému je lepší zavolat metodu pro překreslení, například po překrytí jiným oknem aplikace nebo změně velikosti, než uchovávat bitmapový obraz celého plátna komponenty.

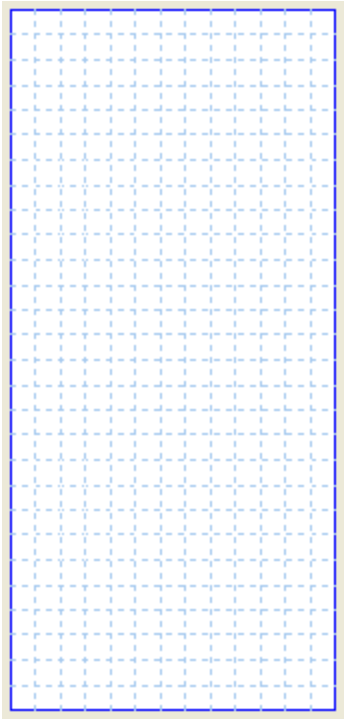
Tyto fakta vedou k logickému závěru přetížit metodu Paint a v ní realizovat překreslení komponenty. Má to však jeden háček. Operace spojené s kreslením jsou pomalé a z hlediska systému náročné operace. Provádí-li se kreslení přímo na plátno, dochází k efektu „problikávání“ obsahu komponenty. Po vyvolání systémového překreslení se nejprve obsah smaže a poté je celý zvonu vykreslen. Mezi těmito stavy je určitá prodleva způsobující tento jev.

Způsobů řešení tohoto problému je více, ale nejčastěji se požívá metoda zvaná „restore and pain“, volný český překlad by mohl znít, ulož a maluj. Jak už název napovídá je obsah nejprve vykreslen v paměti na bitmapu a poté pomocí metody Draw překreslen na komponentu. Vykreslení se provádí jedna ku jedné na souřadnice X a Y, které metoda přijímá jakou atributy. Jedná se vlastně jenom o zkopírování celého obsahu, není tedy nutné provádět přepočet velikosti, jako je tomu například u metody StrechDraw a překreslení paměťové bitmapy na plátno komponenty je velice rychlé.

Pro zrychlení celého postupu je v určitých případech možné provést další efektivní optimalizace. Přesahuje-li vaše komponenta viditelné pole formuláře nebo nepoužíváte-li pro kreslení celý obsah komponenty, je velice užitečné generovat bitmapu v paměti jenom pro překreslovanou část. Tuto část komponenty můžete například zjistit vlastností plátna ClipRect. Návratová hodnota tedy definuje velikost bitmapy, které změníte počáteční souřadnice plátna voláním funkce MoveWindowOrg a tím zajistíte správné vykreslení pro definovanou oblast komponenty. Postupů je samozřejmě víc, není pouze jedno univerzální řešení a vždy je nutné zohlednit všechny aspekty problému.

Nyní když už víme, jak budeme kreslit, se nabízí otázka, co vlastně budeme kreslit. Plocha komponenty je rozdělena do mřížky. Uživatel nebo programátor definuje počet řádků, sloupců a jejich vzhled. Mřížka slouží k rozdělení aktivní plochy komponenty, je zde také možnost ji vůbec nezobrazovat, přičemž zůstanou zachovány veškeré její vlastnosti. Důležitější je ovšem sada funkcí, které s mřížkou úzce spolupracují. Funkce nabízí podporu pro přichytávání k mřížce, posun po polích mřížky jak po horizontále tak vertikále a konverzní funkce.

Zřejmě nejobtížnější je podpora funkce zoom. Je nutné implementovat dva systémy jednotek. Jeden z nich slouží k vykreslování objektů na obrazovku a je tedy v pixelech, druhý je ekvivalentní hodnota v nanometrech. Druhá soustava je přesnější a vyrovnává tak nuance při zvětšování obsahu a nevznikají tak disproporcionální rozdíly.



Obrázek 1 - Mřížka komponenty

5.2 Grid s podporou grafických objektů

Je potomkem třídy předcházející, a rozšiřuje ji o podporu grafických objektů třídy TGraphicBase. Grafickým objektem v tuto chvíli myslím jednotlivé elementy kreslené v rámci grafického gridu. Mám-li být konkrétní, jsou to objekty úsečka, orientovaná úsečka, kruh, elipsa, čtverec, obdélník. Poslední čtyři jmenované objekty nabízejí různou funkcionalitu a podporu grafických formátů. Samotným třídám, do nichž tyto objekty spadají, se budu věnovat později. Chtěl jsem se pouze vyhnout nejasnostem a tento dále hojně používaný termín objasnit již na začátku.

5.2.1 Grafické listy

Objekty jsou ukládány do grafického listu. Určitě nás napadlo použití pole. To by bylo také možné, ale deklarace tříd na základě listů má nespočet výhod jako například dynamická alokace velikosti a společně s nejrůznějšími metodami k jeho obsluze je pro dané použití vhodnější. Bázovou třídu listů pro grafické objekty můžeme nazvat jako základní list.

Základní List

Dědí od třídy TObjectList. Potlačuje metody předka, které jsou svázány s třídou TObject a překrývá je stejnými metodami ovšem pro třídu TGraphicBase. Důvodem je kontrola typů při práci s listem, pokud se snažíme vložit objekt, který nesplňuje daný typ, překladač ohlásí chybu. Ať už by byla příčinou nepozornost nebo neznalost hierarchie tříd, tyto chyby se špatně hledají a ušetříme si tak spoustu práce.

List s akcemi

Je potomkem předchozí třídy a úzce spolupracuje s grafickými objekty v listu. Obsahuje sadu metod, které využívá grafický grid s podporou objektů. List například umožňuje export grafických objektů do formát wmf, generuje xml dokument pomocí kterého je možno objekty replikovat, čte i nastavuje vlastnosti objektů a grafických stylů, dále jsou k dispozici funkce pro hledání objektů pomocí pointrů, typů tříd atd.

Základní list a list s akcemi nejsou v projektu přímo využívány, to jsou až jejich potomci. Používány jsou až třídy TPPGraphicList a TPPSelectList výhodou ovšem zůstává společná funkční základna, kterou rozšiřují jenom o metody specifické pro danou třídu.

Grafický list a list označených objektů

Grafický list je použitý v grafickém gridu a agreguje v sobě list označených položek. Tento list tedy obsahuje pouze objekty vybrané uživatelem. Grafický list implementuje několik metod pro hledání objektů. Po předložení bodu nebo regionu hledá ty grafické objekty, které vyhovují definovanému požadavku. Po nalezení těchto objektů jsou jejich ukazatele přiřazeny do listu označených položek a uživatel nebo programátor s nimi může pohodlněji pracovat.

Nyní už ale zpět ke komponentě grafický grid s podporou objeků. V předchozích odstavcích jsme si klázali způsob správy a uchovávání grafických objektů. Nyní se budu věnovat další funkcionalitě, kterou tato komponenta nabízí.

5.2.2 Ukládání obsahu

Mám li se přiznat, nenapadá mě jediný rozšířený editor, který by neumožňoval ukládání a opětovné načítáni svého obsahu. Použití některého z již existujících formátů jsem po krátké úvaze úplně vyloučil. Možná je to mojí neznalostí, ale podle mého názoru neexistuje dostatečné komplexní řešení, které by mi mohlo nabídnout dostatečný základ, pro zamýšlenou funkcionalitu a zároveň mě neomezovalo svými limity a konceptuálním omezením.

Vymyslel jsem si tedy svůj vlastní formát, samozřejmě jsem použil již některé existující technologie a nestavěl jsem celý základ, jak se říká „na zelené louce“. Pro ukládání atributů objektů jsem se rozhodl použít formát XML. Zvažoval jsem i jiné formáty velmi úsporné jsou například ini soubory, které se hojně využívají v systému Windows. Bohužel ale v jazyce Object Pascal je použití těchto souborů dosti limitováno a existující uřída TIniFile je pro zamýšlené účely nedostačující a musel bych si napsat vlastní parser a generátor.

XML je značkovací jazyk, to znamená, že dokument obsahuje nejen vlastní data, ale i jejich značky určující jejich význam. Bez těchto značek je dokument sice stále čitelný, ale ztrácí svoji informační hodnotu. V Delphi je pro podporu XML zajištěna dostatečně silná podpora jak pro Dom tak i Sax a je možně využít i free produkty třetích stran.

Společně s jednotlivými atributy objektů je důležité ukládal i jejich obsah. Mám na mysli obrázky v různých formátech a tabulky, které jsou vlastně také obrázky určitého druhu a vlastností. Podporu ukládání hodnot objektů a jejich pozdější obnovu do původního stavu v Delphi zajišťuje třída TPerzistetn. Je sice poněkud zvláštní obsahuje jenom velice málo kódu a nelze ji použít přímo. Poskytuje však také dostatečný základ protože perzistence je klíčovým prvkem vizuální programování. Tedy XML i grafiku jsem uložil do paměťového streamu. Tato binární forma reprezentace dat nám nabízí spoustu možností. Všechny streamy se společně s jejich identifikátory načítají do komponenty THKStreams. Výsledný stream celého formuláře nejenže můžete uložit na disk, ale také ho můžeme komprimovat pomocí kompresního algoritmu LHA nebo zašifrovat celosvětově rozšířeným symetrickým šifrovacím algoritmem Blowfish.

5.2.3 Načítání obsahu

Je téměř reverzní postup. V prvním kroku je čten a následně pársován XML soubor nesoucí atributy Samotného grafického gridu. Poté jsou vytvořeny jednotlivé grafické objekty. Po nastavení jejich stylu, který je také definován jako množina atributů, se do paměti načítá přidružená grafika v podobě obrázků.

5.2.4 Ukládání akcí

Mám na mysli kroky zpět (undo) a vpřed (redo). Zmíněná funkcionalit vám dává možnost dostat se do stejného stavu, který existoval v minulosti, před provedením následujících změn a zpět. Počet kroků je omezen z hlediska uvolňování paměťových zdrojů. Vytvořil jsem si tedy jednoduchou třídu, která obsahuje zapouzdřený StringList, do kterého můžeme připisovat nebo z něj vyčítat, dále pak nabízí metody pro pohyb po krocích. Vím, že ve složitějších aplikacích není tato podpora jednoduchou záležitostí a implementaci zajišťuje sada netriviálních algoritmů.

Také jsem přemýšlel o vhodném systému, který by ukládal pouze provedené změny. Nakonec jsem od této myšlenky upustil. Do changelistu jak jsem nazval dříve zmiňovanou třídu, ukládám pouze XML popisy objektů. Jedná se o textové dokumenty, které mohou mít sice stovky řádků, ale paměťová náročnost není značná a zmiňované řešení mi přišlo jako dobrý kompromis.

5.2.5 Grafický export

Obsah celého zobrazeného dokumentu chceme později využívat a je nutné generovat jeho grafickou reprezentaci. K tomu slouží metoda CreateMetaFile, která jako svoji návratovou hodnotu vrací ukazatel na objekt, který je instancí třídy TMetafile. Jedná se o formát vektorové grafiku podporovaný firmou Microsoft. Výsledný grafický obraz není složen z jednotlivých rastrů tedy bodu určité barvy ale množinou úseček bodů a obrázků. Jeho největší devízou konstantní kvalita obrazu při jeho zvětšení popřípadě i zmenšování. Možné je samozřejmě i neproporcionální zvětšování. Podpora ostatních formátů tak nepředstavuj žádný problém, pouze vytvoříme novou instanci

požadovaného formátu a objekt třídy TMetaFile na něj překreslíme. Je to vlastně konverze mezi formáty.

5.3 Grid s definovanými akcemi

Je poslední komponentou ze třídy grafických gridů a v hierarchii tříd je na nejnižším místě. Implementovány jsou reakce na pohyb a akce myši. Chování je úzce spjato s grafickými objekty. Můžeme s nimi bezproblémově pracovat, měnit jejich velikost pozici či provádět vícenásobné výběry, jak bývá dobrým zvykem u běžných aplikací. Tím pádem je definováno jedno z možných chování komponenty. Změnit implicitní chování komponenty není obtížnou záležitostí. Existuje více možností jak toho dosáhnout. Za prvé můžeme vytvořit úplně novou komponentu a jako předka použijeme rodičovskou třídu této komponenty, získáme tak kompletní podporu grafických objektů, bez jakýchkoliv reakcí komponenty s okolím, ty pak upravíme podle své libosti. Drobných úprav můžeme dosáhnout přetížením virtuálních metod komponenty. Funkční kód tak můžeme úplně potlačit, nebo zavoláme metodu předka pomocí klíčového slova inherited tím zachováme původní funkcionalitu, kterou následně v těle metody rozšíříme.

V předcházejících odstavcích jsme si ukázali několik komponent grafických gridů. V této souvislosti jsem často používal trošku abstraktní pojem grafické objekty, který jsem jenom v krátkosti definoval. Také jsem sliboval, že se těmto objektům budu později věnovat podrobněji. Pojďme se tedy podívat na grafické objekty a jejich možnosti.

6 Grafické objekty

Jsou to jednotlivé elementy, se kterými dokážou předchozí komponenty grafických gridů spolupracovat. Mají klíčovou úlohu, protože jejich funkcionalita určuje dosavadní meze, které je samozřejmě v budoucnu možné rozšířit dalšími metodami či přidáním úplně nových grafických objektů. Bez samotných grafických objektů by veškeré předchozí snažení pozbývalo většího smyslu. Byla by to jenom hezká myšlenka, kterou někdo nedotáhl do konce. Jak jedno moudré přísloví praví „dobrý nápad, který nedokážeme realizovat, není dobrý nápad“. Vím, že jsem trošku odběhl od tématu, ale myslím, že případnému čtenáři musí jít ze všech těch objektů, tříd s rozličným přívlastkem, komponent a podobně hlava kolem. Moderní programovací jazyky však staví na těchto konceptech a jsou jejich nedílnou součástí.

6.1 Bázová třída

Po krátkém zamyšlení určitě dospějeme k názoru, že různé grafické elementy mají společné některé vlastnosti, sdílejí i část své funkcionality a dozajista i část svých atributů. Mám li uvést ilustrativní příklad tak úsečka, čtverec i elipsa budou patřit k nějakému gridu, který je jejich vlastníkem, mohou být označeny, bezesporu je můžete posouvat měnit jejich velikost nebo zjišťovat polohu k určitému bodu či dané oblasti. Jedná se téměř o ilustrativní příklad, kde vše mluví pro napsání a definování vlastní bázové třídy. Jiná řešení jsou určitě možná, ale svědčí o nepříliš veliké šikovnosti programátora a nepochopení základních principů objektově orientovaného programování.

Výhod, které přináší použití bázové třídy je mnoho. Pokud vhodně upravíte TList, TObjectList nebo jinou třídu spravující vaše objekty tak, že neumožníte pracovat s jinou než bázovou třídou. Zajistíte si jednak typovou kontrolu, ale také můžete volat metody bázové třídy, které budou ve většině případů virtuální, abstraktní, následně přetěžované samotnými grafickými objekty. Bázová třída si svými virtuálními, abstraktními metodami vynucuje přetížení u svým potomkům a jejich specifické provedení danému grafickému prvku.

Do budoucna si také ušetříte spoustu práce se psaním kódu. Jak jsem již řekl bázová třída je společná všem grafickým objektům a její funkce můžete volat přímo. Jinak by musela předcházet kontrola typu s následným přetypováním a to pro každý jednotlivý grafický objekt. Není složité si představit, jak by se tělo metody lineárně zvětšovalo v závislosti na počtu používaných objektů. Už konec povídání o bázových třídách a pojďme se podívat na zoubek jednotlivým grafickým objektům.

I když to není na první pohled patrné, rozděluji své doposud vytvořené grafické objekty do dvou základních skupin. Jsou to objekty definované na základně dvou bodů, a objekty jejichž velikost je určená obdélníkem. Do první skupiny spadají třídy úsečka a orientovaná úsečka, do druhé skupiny pak všechny doposud vytvořené grafické objekty.

Veškerá funkcionalita důležitá pro samotné kreslení, ať již mám na mysli grafickou reprezentaci či jejího vlastního obsahu není součásti této unity. Z toho také vyplývá, že není implementována přímo v potomcích třídy TGraphiBase, protože v jazyce Object Pascal musí být deklarace metody a její implementace ve stejné programové jednotce, nazývané unity.

Důvodem extrahování těchto operací je lepší znovupoužitelnost. Grafické objekty jsou pevně svázány s gridem, který je jejich vlastníkem. Navíc atributy určující polohu jsou v jiné měrné soustavě a musí být proveden přepočet na obrazové pixely, který provádí konverzní funkce gridu vzhledem k jeho vnitřnímu nastavení určujícím potřebný konverzní poměr. Grafické objekty tedy v sobě agregují další objekt zprostředkující jejich samotné vykreslování. Možná se to může zdát zbytečně matoucí a nesmyslné, štěpit takto funkcionalitu do dvou různých programových jednotek, přičemž se zdá, že vše patří do stejné třídy. Souhlasil bych s vámi, ale jenom pouze kdybych tímto postupem, nezískal náramný benefit použitelný v mnoha různorodých situacích. Později v této práci uvedu některé příklady, které celý tento krok osvětlí, ale nejprve se budu věnovat grafickým objektů samotným.

Všechny tyto třídy i ty které jsem ještě nenapsal, mají společné rysy, protože musí mít jako předka bázovou třídu TGraphicBase. Objekt, který je instancí některé z těchto tříd můžete vykreslit, označit, odznačit, pohybovat s ním, měnit jeho velikost, přichytit ho

k mřížce, načíst ho z XML uzlu popřípadě tento uzel vygenerovat, změnit některou jeho vlastnost, zjistit jeho polohu vůči bodu či výřezu plochy definované jako obdélník. V tuto chvíli si představíme samotné grafické třídy, jejichž instance se generují za běhu programu a budeme se věnovat pouze jejich vzájemným odlišnostem.

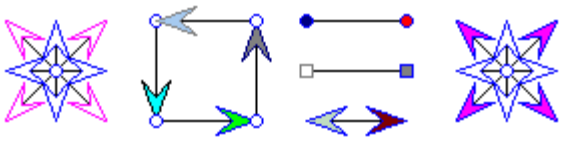
6.2 Úsečka

Jedná se pouze o úsečku z bodu a do bodu b, její grafický vzhled určuje třída TPen můžeme tedy měnit její barvu, styl a šířku. Pero používám geometrické, protože defaultní isometrické pero třídy Canvas se nechová příliš korektně. Pokud je šířka pera větší než jedna, je styl pera ignorován a chování objektů je přinejmenším nevhodné a vše se tváří jako by šlo o chybu programátora.

Obrázek 2 - Úsečka

6.3 Orientovaná úsečka

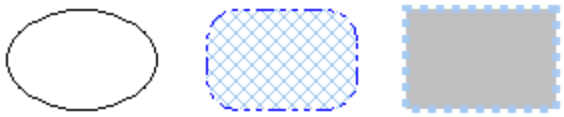
Dědí od předchozí třídy. Nabízí navíc možnost vykreslit grafické zakončení úsečky. Máme na výběr ze třech typů zakončení. Jedná se o zakončení čtvercem, kruhem, šipkou nebo můžeme zakončení ponechat nedefinované. Můžete tak například určit směr šipky a ve spolupráci s dalšími grafickými objekty nakreslit vývojový diagram, popřípadě nějaký jiný orientovaný graf.



Obrázek 3 - Orientovaná úsečka

6.4 Obdélník se zakulacenými rohy

V této třídě jsem použil, trik jak si v budoucnosti ušetřit trošku práce. Objekt, který je instancí této třídy neumožňuje kreslit pouze čtverec, nebo obdélník, ale i elipsu, kruh a čtverec se zakulacenými rohy. Pro kreslení nepoužívám metodu plátna Canvas Rectangle, ale metodu RoundRect, která vykreslí obdélník, ale na rozdíl od předchozí metody se zakulacením rohů, které se metodě předává jak parametr. Pokud je zakulacení nulové, je vykreslen klasický obdélník, pokud zakulacení přesahuje délku delší strany, vykreslí se elipsa v rámci daného obdélníku. V druhém případě, tedy je li zakulacení nenulové, je důležité vygenerovat na plátně region a jenom v rámci toho regionu bude plátno akceptovat a provádět kreslící operace. Pero se jako v předchozím případě požívá geometrické, ale navíc přibyla výplň objektu zprostředkované pomocí třídy TBrush.



Obrázek 4 - Obdélník se zakulacenými rohy

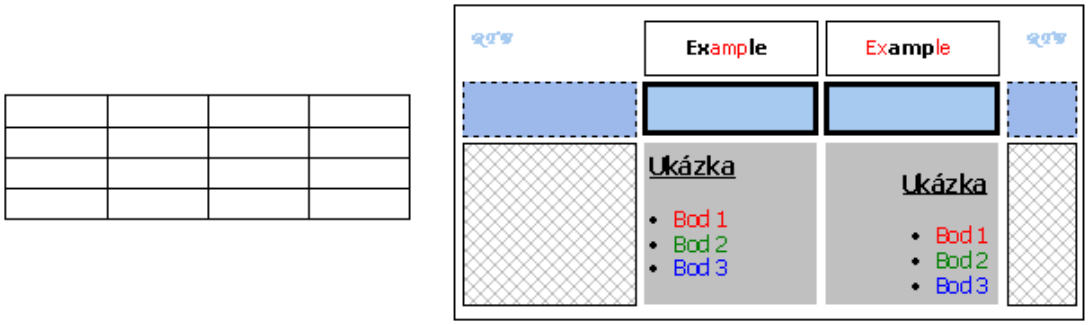
6.5 Obdélník s podporou obrázků

Je potomkem předchozí třídy a rozšiřuje ji o podporu obrázků. Podporovány jsou nejběžnější formáty bmp, jpg, jpeg, wmf, ico, png. Pro jejich správné vykreslení v rámci uživatelem definovaného objektu jsou zapotřebí funkce regionů. Pixely, které přesahují tento region, nebudou vykresleny, takže obrázek nepřesáhne okraje grafického objektu a bude tedy vykreslen pouze v rámci jeho výplně. Obrázek můžete v objektu roztáhnout, přičemž jeho velikost může zůstat proporcionální či nikoliv. Můžete také zvolit zarovnání obrázku vlevo, na střed, vpravo nebo pozici obrázku nahoře, uprostřed, vespod.

Obrázek 5 - Obdélník s podporou obrázků

6.6 Obdélník s podporou tabulek

Dlouho jsem přemýšlel jak vyřešit podporu tabulek. Docela elegantní mi nakonec přišlo, vygenerovat tabulku do formátu Windows Meta File. O jeho nesporných výhodách jsem se zmínil již dříve, a proto se nebudu opakovat. Popis tabulky pro její zpětnou replikaci pak uložit do textového popisu a bylo. V tuhle chvíli jsem sice ještě žádný tabulkový editor neměl, ale zmiňované řešení mi přinášelo do budoucna volné ruce k jeho realizaci. Tabulka tedy bude jenom prostý obrázek. Třídu pro podporu obrázků jsem měl již hotovou, takže stačilo dědit od této třídy přidat jeden textový atribut pro uložení tabulky pro její replikaci a celá věc byla dílem okamžiku.



Obrázek 6 - Obdélník s podporou tabulek

6.7 Obdélník s podporou obrázků a textů

Je potomkem obdélníku s podporou obrázků. Ať už máme čtverec, obdélník, elipsu, kruh, obdélník se zakulacenými rohy, s obrázkem nebo bez tak funkcionalita této třída přidává podporu textů ve formátu rtf nebo txt. Kreslená oblast je opět omezena pomocí regionů. Na rozdíl od obrázků můžete určit odsazení textu od každé strany. Přičemž nastavení pozice a zarovnání textu jsou totožné. Chtěl jsem, aby se text formátoval

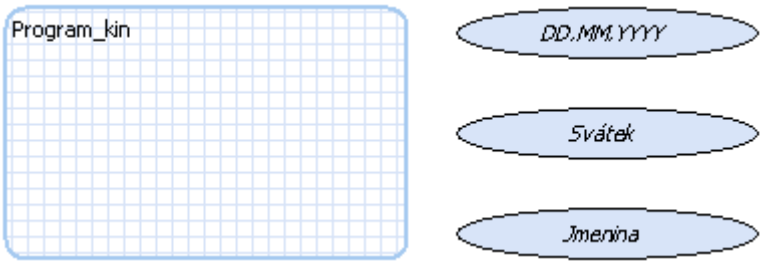
v závislosti na tvaru vnitřní oblasti, bohužel to bylo možné jenom pro text ve formátu txt. Doposud jsem nepřišel na způsob jak stejného výsledku dosáhnout i pro formát rtf. Zkoušel jsem i text sázet ručně po jednotlivých písmenech, ale nikdy jsem nedosáhl uspokojivého výsledku. Stejně jako u tabulkového rectu se text vykreslí na formát Windows Meta File, aby byla zajištěna jeho konstantní kvalita například při změně zoomu.

Obrázek 7 - Obdélník s podporou obrázků a textů

6.8 Obdélník s podporou databázového obsahu

Rodičovskou třídou je obdélník s podporou obrázků a textů, třída plně využívá funkcionalitu nabízenou svými předky. Obsahy však nevytváří uživatel, ale jsou linkována z externích databázových zdrojů, které nejsou pevně dané a lze je libovolně modifikovat. V žádné z tříd, které jsem napsal, proto nenajdete jediný pevně napsaný select či pevně definované připojení na databázi vše se vytváří plně dynamicky.

Snažil jsem se tak dosáhnout co největší flexibility, různorodého použití a zároveň jsem se nechtěl omezovat pouze na jednu databázovou platformu. Podle mého názoru je v tomto směru projekt do určité míry originální. Byla to také nejobtížnější věc, o kterou jsem se kdy pokusil. Nebudu předbíhat a celou věc si ukážeme hezky od začátku.



Obrázek 8 - Obdélník s podporou databázového obsahu

7 Selekce databázových dat

Po několikadenním hledání, kdy jsem obracel internet vzhůru nohama, jsem narazil na komponenty Zeos. Jak sami autoři uvádějí, zní to prostě dobře a žádný hlubší smysl se prý za názvem neskrývá. Komponenty jsou volně ke stažení a neváže se na ně žádné licenční ujednání. Možnosti a funkcionalita jsou skutečně velice bohaté. Komponenty nabízí připojení na zhruba deset různých databázových systému. Nechybí velké a rozsáhlé systémy pro řízení báze dat jakými je například Oracle, Microsoft SQL server, DB2, či velmi rozšířené MySQL, popřípadě méně rozšířené, ale přesto velmi silné a stabilní systémy řízení báze dat jako například Interbase, Firebird, PosgreSQL, Sybase, až po některá alternativní řešení jako je SQLite, které také nalezne své uplatnění například pro méně náročné aplikace. Další nesporným kladem je, že spojení s databází je přímé a není tedy nutná instalace doprovodných ovladačů, jako je například ODBC od firmy Microsoft. Instalace a používání ve vývojovém prostředí Delphi je bezproblémová. Komponenty jsou svou strukturou a pojmenováním jak samotných tříd, tak i metod téměř shodné s databázovými komponentami dodávanými s Delphi a jejich používání tak činí velmi intuitivní. Jedná se tedy o mocný nástroj a pokusil jsem se maximálně využít jeho možnosti.

7.1 Databázové připojení

Vytvořil jsem si vlastní třídu nazvanou TPPConnection, která v sobě agreguje dvě databázové připojení. Dovolují uživateli dynamicky generovat objekty, které provádějí dotazy nad databází, ale také navázat spojení po sestavení connectionstringu. Jedná se o jednoduchý XML dokument, který jsem si definoval, a jeho minimální struktura je pevně daná, protože pro navázání spojení s databází je nutná určitá množina informace, bez které toto spojení není možné uskutečnit. XML dokumenty mohou nést nějakou další informaci, využitelnou aplikací na vyšších úrovních záleží na konkrétní implementaci a zamýšleném účelu. Vím, že se pohybuji v abstraktní rovině a pro nezainteresovaného čtenáře jsou to téměř prázdná slova. Proto uvedu konkrétní příklad využití v aplikaci PaperPalmManager.

Jak jsem již řekl třída TPPConnection obsahuje dvě databázová připojení. Jedno se jmenuje Linker a druhé Result. Linker se připojí na databázi avšak XML dokument,

který toto spojení definuje, také odkazuje na tabulku s dalšími XML dokumenty. Podle nich se sestaví sekce v jednotlivých nabídkách aplikace, které má uživatel k dispozici. Samotné položky těchto nabídek pak odkazují na další dokumenty, které již určují cílovou databázi i typ selectované položky. Protože připojení na databázi je časově náročná operace je k dispozici druhé připojení Result. Jelikož jsou nabídky soustavně využívány, je Linker stále připojen na databázi, která tyto informace obsahuje, zatímco připojení Result je používáno pro selectování informací z ostatních databází.

Chcete li změnit databázovou platformu, která definuje nabídky aplikace. Není problém, pouze změníte XML dokument, který na ni odkazuje. Tento dokument může mít uživatel na svém disku nebo ho vaše aplikace může automaticky stahovat z nějakého zdroje na internetu například z prostého URL nebo FTP, možností je spousta.

Chcete li rozšířit či modifikovat nabízené nabídky. Pouze ve své databázi upravíte XML dokumenty, pomocí kterých se tyto nabídky generují a vytvoříte odpovídající struktury. Od tohoto okamžiku mají k dispozici tuto novou nabídku i všichni uživatelé vaší aplikace po celém světě.

Ukázali jsme si způsob vytváření uživatelských nabídek. Nevyřčeno však stále zůstává, co tyto nabídky vlastně nabízejí a jaká je jejich užitková hodnota pro potencionální uživatele. I když databáze nabízejí atributy typu BLOB, jejichž obsahem může mýt prakticky cokoliv mající binární reprezentaci, tak v největším počtu případů bude výstup obsahovat tabulku či jediné pole s textovým obsahem.

7.2 Datové vstupy

Konektivitu k různým systémům báze dat máme již vyřešenou, jak jsem říkal XML dokument, který toto spojení definuje, je možné rozšiřovat, a proto dokumenty obsahují jednak atribut určující typ cílové položky tedy tabulka a textové pole, ale i popis pro její dosažení. Samozřejmě má tento popis jistou strukturu její dodržení je podmínkou pro správné fungování celého procesu. Validaci je možné provést pomocí DTD dokumentu. Pro aplikaci PaperPalmManager je navíc implementována podpora linkování kalendářových dat. Jsou to položky vztahující se k určitému datu. Může to být svátek,

den v týdnu, datum atd. Využití je však dost specifické, a proto se mu nebudu podrobněji věnovat, i když je součástí grafických objektů.

7.2.1 Textové pole

V případě, kdy chceme z databáze selektovat pouze textovou položku je situace mnohem jednodušší. XML dokument obsahuje navíc pouze SQL dotaz a také název atributu, která se z výsledného dotazu použije. Po předložení XML dokumentu se tedy provede dotaz a výsledný text je ihned načten do textového pole. Nečeká se tedy na žádný další vnější podmět. Podpora rtf textu je implementována v třídě TTextRect, jejíž atributy a metody můžeme libovolně využívat, neboť je předkem třídy TLinkRect, která nabízí podporu jak pro textové pole, tak pro tabulky.

7.2.2 Tabulky

S tabulkami je to ovšem daleko obtížnější. Stejně jako v případě textových polí, není známa struktura databáze a pokud chcete například zjistit program ve vašem oblíbeném kině, tak napsání samotného dotazu nad databází by nebylo složitou záležitostí, my ovšem nevíme, jaké kino si uživatel vybere z jakého je města popřípadě země či na jaký den si návštěvu naplánoval. Všechny tyto atributy nemůže uživatel pouze vepsat do textových polí, ale musí být vázány na konkrétní existující datovou strukturu a také musí být mezi jednotlivými položkami definovány logické závislosti. Mám li se držet příkladu s programem kin, tak poté co uživatel vybere město, ve kterém se nachází, by měl obsah dalších ovládacích prvků na tento výběr zareagovat a omezit nabídku kin pouze na vybrané město. Počet ovládacích prvků se bude také lišit i u tohoto příkladu je více možností. Bude záležet třeba na tom, jak detailně budeme chtít výběr specifikovat a do určité míry i na struktuře databáze pro daný výběr.

Jinými slovy jediné co víme je, že výstup bude tabulka, která obsahuje libovolný počet polí z libovolné databáze na libovolném systému řízení báze dat a je závislá na výběru 0 až N položek, které jsou mezi sebou v určité logické závislosti a navzájem se ovlivňují. Celý formulář musí být dynamicky vygenerován z popisu, uloženém v externí databázi.

7.3 Dynamicky generované formuláře

Nebudeme se zatěžovat všemi slepými uličkami a milníky, ale ukážeme si až finální řešení celého problému. Textový popis pro vygenerování celého formuláře jsem se rozhodl ukládat do XML. Každý generovaný ovládací prvek má definovanou pozici na formuláři pomocí koordinátů Left, Top, With, Height. Nad levým horním okrajem bude umístěn popisek ovládacího prvku.

```
<controltype>
     <title>text</title>
     <left>číslo</left>
     <top>číslo</top>
     <width>číslo</width>
     <height>číslo</height>
</controltype>
```

Tuto minimální strukturu bude obsahovat každý generovaný ovládací prvek, přičemž atribut controltype v závislosti na svém názvu určí typ ovládacího prvku. Generovat komponenty na formuláři tedy není obtížnou záležitostí. Všechny ovládací prvky jsou součástí třídy, která je spravuje a zapouzdřuje, přičemž je nutná specifická vnitřní implementace pro různé ovládací prvky, avšak navenek je poskytováno rozhraní, které se využívá pro řízení a komunikaci s nadřazenou třídou. Nadřazenou třídou mám na mysli centrální prvek, který komponenty generuje, nastavuje, řídí jejich vzájemnou komunikaci atd. Ovládací prvky podle účelu spadají do několika skupin:

Bez databázové konektivity - Sem patří například edit nebo date time picker. A jejich vstup je plně v moci uživatele. Třídu, ve které jsou zapouzdřeny, můžeme požádat o textovou reprezentaci obsahu či části obsahu například rok u ovládacího prvku date time picker.

S databázovou konektivitou a jednou výstupní položkou - Do této skupiny spadají ovládací prvky jako combo box, list box a radio group. Položky těchto ovládacích prvků se načtou z jednoho sloupce sql dotazu. Opět můžeme volat funkci, které předáme jako parametr název pole a návratová hodnota bude odpovídat aktuálně vybrané položce.

S databázovou konektivitou a více výstupními položkami - Do této skupinky patří tabulky. XML soubor tedy navíc obsahuje seznam polí, které se mají zobrazovat, jejich

velikost a titulek. Formulář může obsahovat libovolný počet tabulek a právě jednu tabulku, kterou označujeme jako hlavní. Je potomkem obyčejné tabulky a obsahuje navíc zaškrtávací tlačítka jak pro řádky (selekce), tak i pro sloupce (projekce), tabulku můžeme exportovat do formátu csv. Třída, která spravuje všechny vygenerované objekty, ji také publikuje jako výstupní a tedy se předpokládá, že se jedná cílovou tabulku obsahující data, která budou dále využívána.

Pokud ovládací prvky obsahují databázovou konektivitu, vygenerují si vlastní objekt, který je instancí třídy TZQuery připojí se na globální objekt connection a provedou na databázi příslušný dotaz. U ovládacích prvků s jednou výstupní položkou se jednotlivé řádky načítají. U ovládacích prvků s více výstupními položkami je přístup přímí pomocí objektu DataSource.

Generování ovládacích prvků, máme tedy již hotové. Stále však mezi nimi není zajištěna závislost a komunikace. Uvedeme si příklad jednoduchého XML dokumentu, na kterém si objasníme řešení tohoto problému, záměrně z něj vypustíme nepotřebné atributy, abychom zmenšili jeho velikost. U každého ovládacího prvku tedy nebudeme uvádět atributy title, left, top, width a height, které souží jako koordináty a nastavení titulku a pro tento příklad nejsou důležité. Ze stejných důvodů nebudu také uvádět atributy pro nastavení formuláře.

```
<structure type="table">
      <combobox>
                  <sql use="nazev">
                  Select id_mesto,nazev from mesto
            </sql>
      </combobox>

      <combobox>
                  <sql use="nazev">
                        Select id_kino,nazev from kino where id_mesto =
                        item0
                  </sql>
                  <atributs>
                        <atribut item="0" field="id_mesto"/>
                  </atributs>
      </combobox>

      <mastergrid>
                  <sql>
                        Select * from mesto
                        join kino on mesto.id_mesto = kino.id_mesto
                        join film on kino.id_kino = film.id_kino
                        where kino.id_kino = item1 >
                  </sql>
                  <atributs>
                        <atribut item="1" field="id_kino"/>
                  </atributs>
                  <fields>
                        <field use="nazev" title="Město" width="100"/>
                        <field use="nazev_1" title="Kino" width="100"/>
                        <field use="nazev_2" title="Film" width="100"/>
                        <field use="datum" title="Datum" width="100"/>
                  </fields>
      </mastergrid>

</structure>
```

Prvním ovládacím prvkem je tedy combo box provede na databázi dotaz, který je určen obsahem prvku sql a svůj obsah načte z atributu, use tohoto prvku. Pokud nechceme vygenerovat combo box, ale nějaký jiný ovládací prvek ze skupiny s databázovou konektivitou a jednou výstupní položkou stačí pouze změnit název prvku z combobox na listbox či gadiogroup.

Druhým ovládacím prvkem je také combo box, ale na rozdíl od prvního případu se v sql dotazu vyskytuje rezervované slovo item bezprostředně následované číslovkou. Je to místo na které se vloží hodnota pole u ovládacího prvku, který leží v listu ovládacích prvků na nultém místě. V Object Pascalu začínají pole i listy u indexu 0 a na nulté pozici je tedy první položka. V XML dokumentu pro tento combo box také najdeme položku atribut, která v sobě obsahuje seznam atributů, které se musí načíst a následně

jejich hodnotou nahradit příslušnou položku v SQL dotazu. Celá tato konstrukce tedy říká, požádej ovládací prvek, který je v listu uložen na pozici nula o hodnotu položky id_mesto a tu pak vlož do sql dotazu na pozici item0 a dotaz proveď.

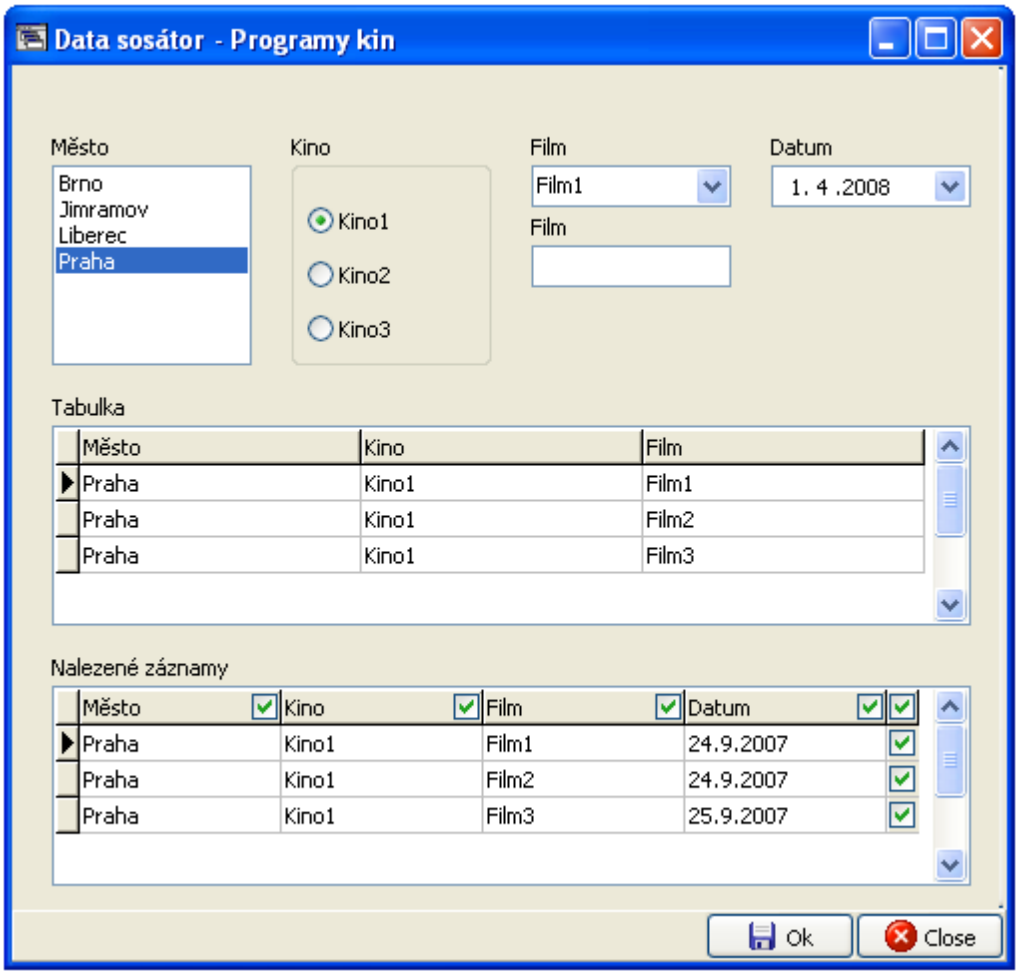
Obsahem prvního combo boxu jsou všechny názvy měst uložené v tabulce města. Druhý combo box obsahuje názvy kin, ale pouze v daném městě, protože se z prvního combo boxu použije položka id_mesto a výběr se tak omezí. Musíme mít na zřeteli, že se tímto způsobem nemůžeme odkazovat na dosud neexistující položky a vytváření ovládacích prvků musí probíhat v určitém sledu. Toto omezení ale nepředstavuje žádný markantní problém a struktura XML dokumentu není těžké přizpůsobit.

Po vygenerování celého formuláře jsou tedy obsahy ovládacích prvků načteny podle daných závislostí. U všech ovládacích prvků jsou odchytávány události, které souvisí se změnou vybrané položky. Pokud tato událost nastane, znamená to, že uživatel změnil vybranou položku. Ovládací prvek pošle zprávu o svojí změně objektu, který je odpovědný za vnitřní komunikaci a který zná všechny ovládací prvky, které byly v rámci formuláře vygenerovány. Může tedy provést optimalizaci a plán jak na změnu zareagovat. Nemůžeme pouze provést celkový refresh, protože v tom případě přijdeme o změnu provedenou uživatelem a dostaneme se tak pokaždé do konstantního stavu, který je identický jako po inicializaci formuláře. Musíme tedy najít a vyčlenit pouze ty položky, kterých se změna týká. Najdeme je jednoduše, jsou to takové ovládací prvky, které při skládání sql dotazu odkazují na změněnou položku. Také to ovšem znamená, že se jejich obsah bude také měnit a musí tedy také zaslat událost o své budoucí změně, protože na nich mohou být závislé zase jiné ovládací prvky. Celý proces se tak opakuje, k zacyklení nemůže dojít, protože jak jsem se již zmínil, odkazovat se můžeme pouze na dříve vytvořené ovládací prvky a nemůže tedy vzniknout smyčka. Na změnu by bylo možné reagovat hned, ale na jednu položku může odkazovat více ovládacích prvků a zbytečně by se tak provádělo více dotazů než je nezbytně nutné. Proto je lepší provést optimalizaci, odměnou nám bude vyšší rychlost a menší vytížení komunikačního kanálu.

Nic nám tedy nebrání dynamicky generovat formuláře, které se svojí strukturou, vzhledem ani chováním nijak výrazně neliší od běžných formulářů databázových aplikací. Jistě by obsahem dokumentu mohli být i další ovládaví prvky, jako například

panely, splitery, záložky či jiné ovládací prvku systému Windows. Velikost dokumentu by se pak zvětšovala. Podle mého názoru by však nebyl přínos tak markantní. Pokud se ovšem v budoucnu objeví dobré důvody pro rozšíření, nebudu váhat a celou sadu tříd rozšířím k zamýšleným účelům.

Na následujícím obrázku si můžete prohlédnout jeden ukázkový formulář. Celou dobu jsem jednotlivé kroky vysvětloval na příkladu s programem kin, a proto v tom budu pokračovat. Udělal jsem si jednoduchou databázi s tabulkami mesto, kino, film. Na formuláři jsou všechny komponenty, které můžete vygenerovat. Tento příklad je ilustrativní a proto grid s textovým popisem a tabulka zobrazuje pouze duplicitní informace, ovládací prvek edit s textovým popisem film není využíván. Jak jsem však řekl, příklad má pouze ilustrativní charakter a chtěl jsem demonstrovat možnosti a správnou funkcionalitu.

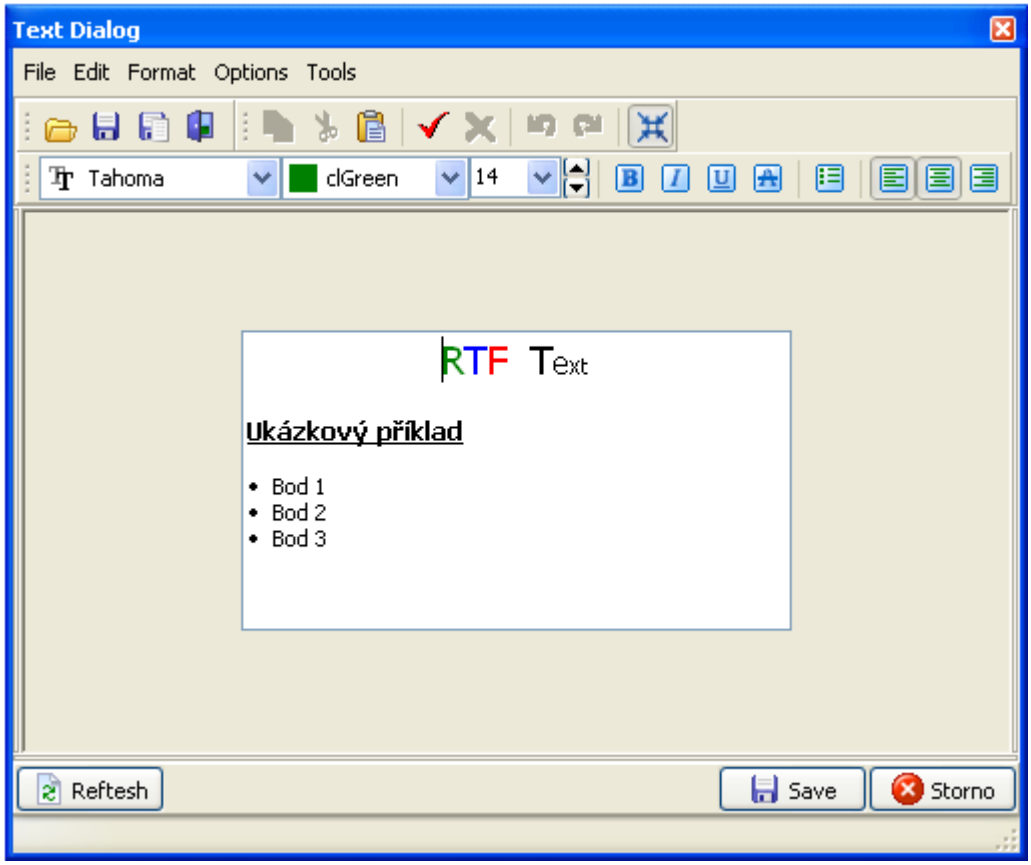


Obrázek 9 - Dynamicky vygenerovaný formulář

8 Editory

Celá sada komponent také obsahuje dva editory. Grafické objekty sice umí pracovat s rtf texty a připravená je i podpora pro tabulky ovšem elegantní řešení je vytvářet odpovídající vstupy v rámci jedné aplikace. Naprogramoval jsem tedy následující dva editory. Jejich grafické rozhraní jsem se snažil udělat intuitivní a v podobném duchu, aby se s ním uživatel nemusel dlouho seznamovat.

8.1 Textový editor



Obrázek 10 - Textový editor

Je to klasický rtf editor, který obsahuje jako hlavní komponentu RichEdit, který patří do základních ovládacích prvků ve windows. Práce s textem a jeho formátování není obtížnou záležitostí.

Pokud víme něco o používání a možnostech ActionManageru jsou základní funkce editoru dílem okamžiku. Rozhodl jsem se také přidat některé další běžné funkce, které uživatel jistě ocení je to ukládání i načítání ze souboru, list změn, bullet, změna velikosti plochy a funkce refresh pro navrácení se do počátečního stavu. Editor není nijak asociován s ostatními komponentami a je ho možné využít v kterékoliv jiné aplikaci, či jako aplikaci samotnou.

8.2 Tabulkový editor

Komponentu pro editaci tabulek jsem si chtěl zpočátku napsat sám. Nakonec jsem od tohoto nápadu upustil. Nejenže by to bylo časově náročné, ale definicí nové třídy, která má jako předka třídu TStringGrid, upravením, předefinování jejich vlastností a přidáním dalších metod lze dosáhnout stejného výsledku. Také si nedělám iluze, že by mou vytvořená tabulka dosahovala alespoň částečných kvalit komponenty od firmy Borland, je také známo, že tyto komponenty nepatří k těm nejjednodušším.

Nejdříve jsem vypnul defaultní vykreslování tabulky. Dalším krokem je vytvořit sede baar komponenty. Tyto ovládací prvky slouží pro označení sloupců a řádků a také pro jejich vypínání. Pokud řádek nebo sloupec tabulky vypnete, bude stále obsahem tabulky, budete ho moci editovat a jeho chování se v rámci rodičovského formuláře vůbec nezmění. Změní se ovšem výstup, který tabulka vygeneruje, vypnuté řádky a sloupce nebudou jeho součástí, ale přitom nepoužité informace zůstávají zachovány a uživatel je může, ale i nemusí použít později.

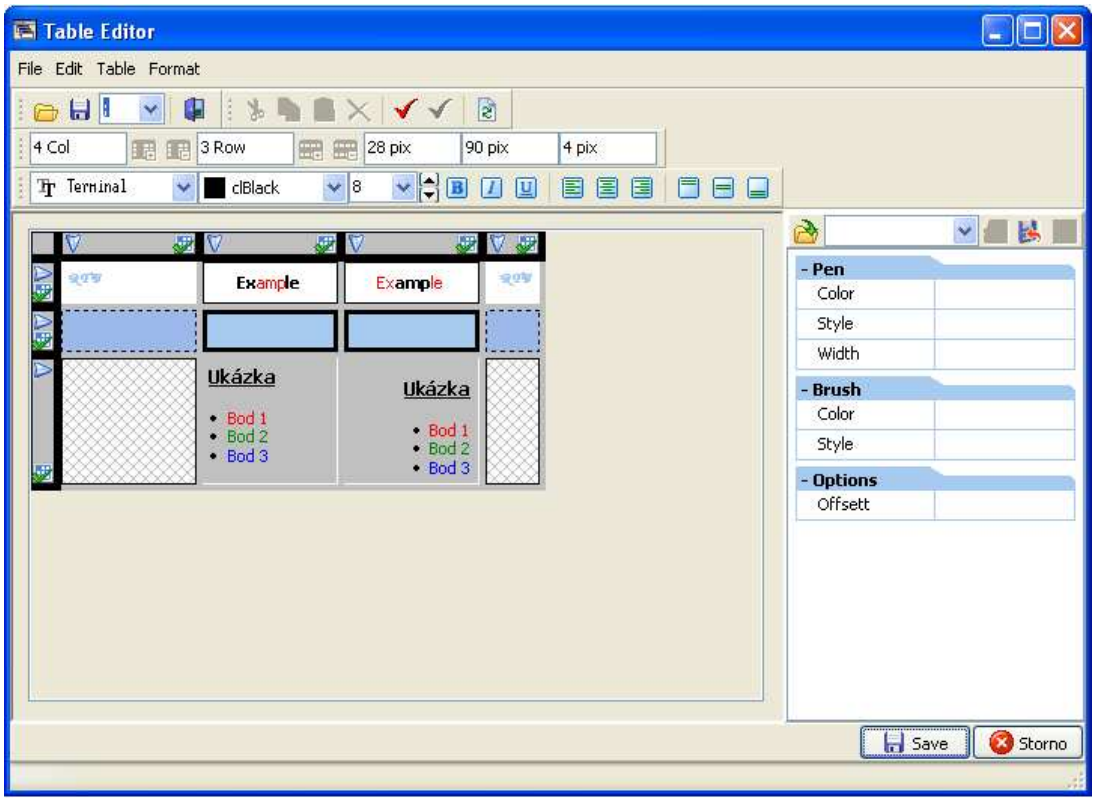
Tabulka při vykreslování svých políček volá virtuální metodu, kde se překreslení vyžádá. V rámci těla této metody si můžete výsledný vzhled tabulky libovolně přizpůsobit. Na side bar komponenty jsem tedy vykreslil ikony simulující tlačítka. Po kliknutí myší můžete lehce dopočítat, zda událost nenastala v rámci této arey. Pokud je tomu tak provedete odpovídající operaci. Výsledný efekt je téměř k nerozpoznání, rozdíl je pouze v tom, že tlačítko má jako předka TWinControl je to tedy ovládací prvek založený na okně a může získat zaměření Focus. Opětovnou akci pak můžete vyvolat například po stisknutí tlačítka enter, to ale v tomto případě není možné.

I když by se i toto chování dalo simulovat, popřípadě by komponenta mohla generovat skutečná tlačítka, které by poté umístila na odpovídající pozice a odchytávala jejich události, je přínos takových to postupů minimální a pole mého názoru si většina uživatelů rozdílů ani nevšimne.

Nyní již přistoupíme k samotným políčkům tabulky. Jestli si ještě vzpomínáte, jak jsem veškeré operace spojené s kreslením extrahoval z grafických objektů. Tak v tuhle chvíli se celý přístup náramně zúročí. Políčko tabulky se vlastně nijak neliší od vzhledu grafického objektu. Musel jsem sice celou třídu podědit a lehce upravit její chováni, ale jinak jsem měl kompletní základnu, kterou jsem potřeboval.

Každé políčko tabulky může mít vlastní styl pera, kterým je vykresleno, libovolnou výplň, je zajištěna podpora trf textů bylo by také možné používání obrázků v rámci jednotlivých polí, ale tuto podporu jsem nakonec neimplementoval. Velikost řádků a sloupců je plně v rukou uživatele a není nijak limitována. Editor podporuje označení více políček tabulky najednou, obsah je možné uložit do souboru a později ho opět načíst, políčka je také možno kopírovat, a formulář také podporuje ukládání stylů. Jak jsem řekl grafické objekty i políčka tabulky používají pro své kreslení stejný kód a styly jsou tedy použitelné v obou případech. Protože se veškerý obsah tabulky kreslí, není problém vygenerovat vektorový grafický formát Windows Meta File, který je používán v grafických objektech a výsledné chování tabulky pak vypadá hodně zdařile.

Do tohoto editoru se také kopírují vybraná data z dynamicky generovaných formulářů, které mají jako výstup tabulku. Uživatel tak získá kýžená data a celou tabulku si pak pomocí grafických stylů či editací jednotlivých polí upraví k obrazu svému.



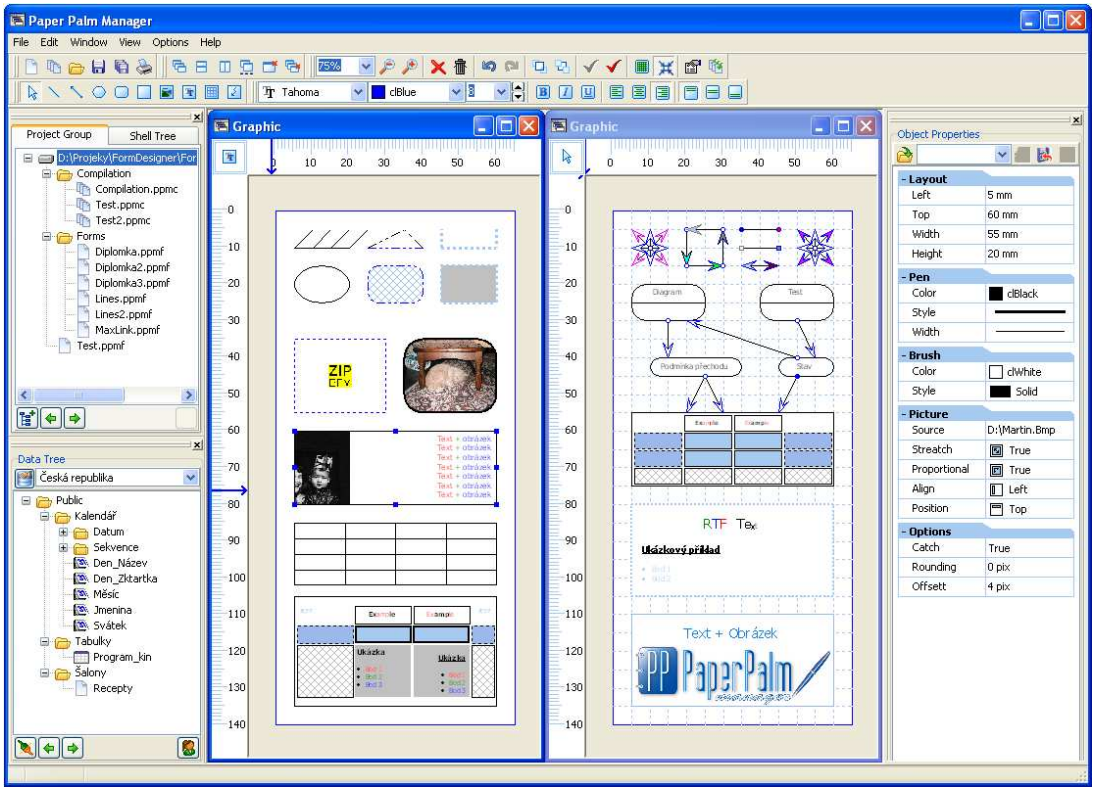
Obrázek 11 - Tabulkový editor

9 Použití komponent a tříd

Ukázali jsme si části celé struktury komponent a tříd. Pro názornost přikládám některé své aplikace, jež tyto komponenty ve větší či menší míře využívají.

9.1 PaparPalmManager

Je to MDI vektorový grafický editor, který vyvíjí moje firma a je součástí produktu PaperPalm. Projekt je nyní ve své finální fázi, a měl by být již brzy volně ke stažení.

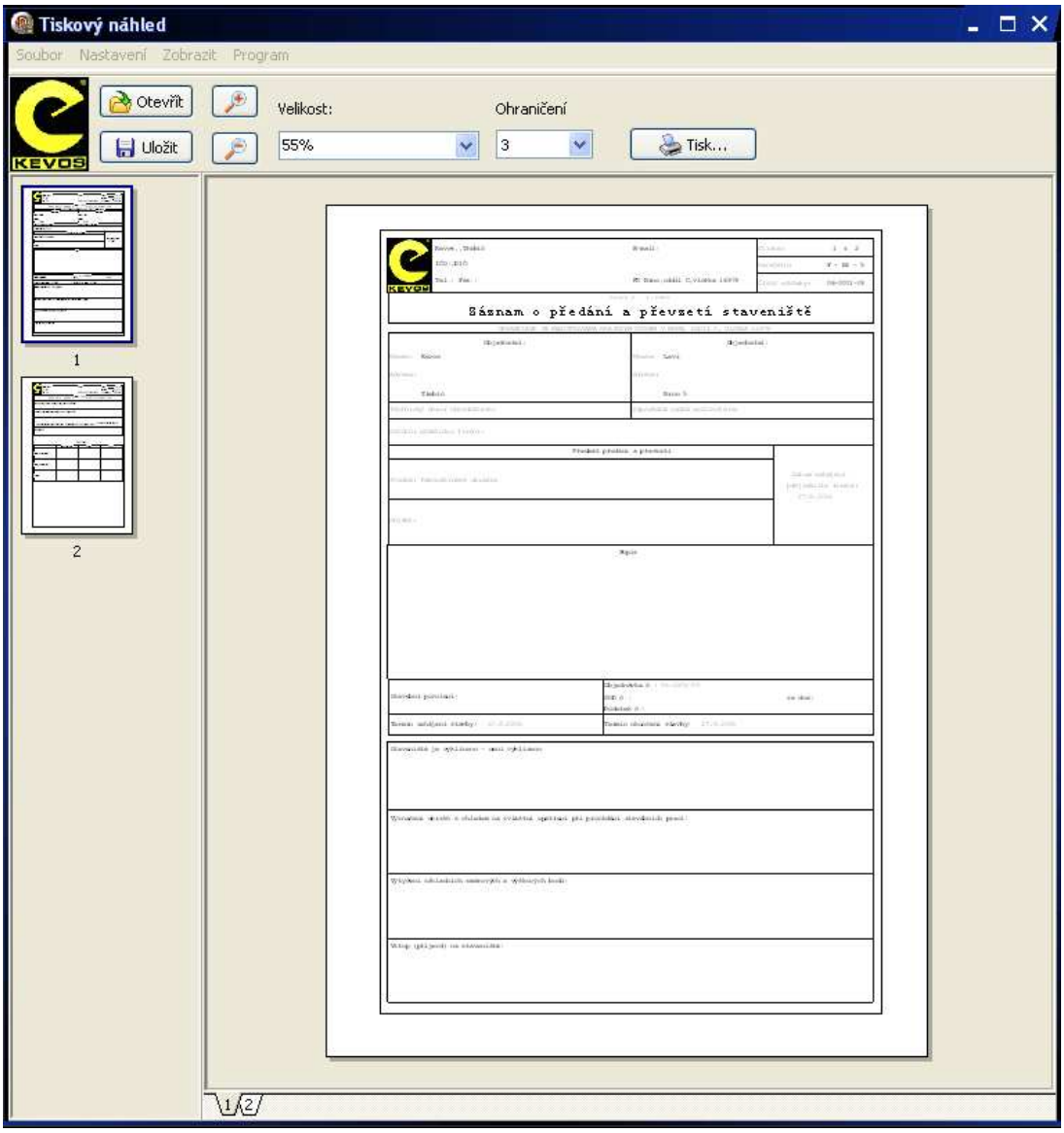


Obrázek 12 - PaperPalmManager

V levé spodní části je okno nástrojů které se jmenuje Data Tree. Pokud si vzpomeneme na generování nabídek, tak právě jeho obsah se vytváří v závislosti na strukturách umístěných v externí databázi. V aktivní části aplikace jsou dva formluláře s grafickými gridy. Každý obsahuje několik grafických objeků, které tvoří jeho obsah.

9.2 Zakázkovník

Jedná se o informační databázový systém pro stavební firmy. Tiskové výstupy mají neměnnou strukturu, kterou definuje norma ISO 9001. Tiskový obsah se kreslí na komponentu, která následně odešle svůj obsah tiskárně. Pro vytváření náhledu používám třídy pro kreslení textových polí, obrázků a tabulek. Obsahy se načítají z databáze, ale v tomto případě je vše jednodušší o to, že mají pevně danou a neměnnou strukturu.



Obrázek 13 - Zakázkovník – Tiskový náhled

10 Závěr

Objektově orientovaný návrh není jednoduchou záležitostí, bez dodržení obecně platných postupů by vás na jeho konci čekalo trpké zklamání, nebo by utrpěla znovupoužitelnost vašich tříd. Výhodnější je postupovat metodicky již od samého začátku. V dalších projektech bych chtěl zlepšit analytickou a návrhovou část při tvorbě aplikací, které jsem dříve nevěnoval tolik pozornosti. Nyní už vím, že samotná implementace může tvořit například jen 20% celkového času při tvorbě aplikace, přičemž zbylý čas věnujeme analýze, návrhu, testování a dokumentaci.

Práce odpovídá stanoveným cílům, tedy vytvoření komponent a tříd pro WYSIWYG grafické editory. V některých případech jsem zamýšlenou funkcionalitu rozšířil a implementoval další třídy a metody i některé nové grafické objekty. Spolupráce grafického gridu a grafických objektů je bezproblémová a bez obtíží lze využít jenom část z nabízené funkcionality. Linkování databázových dat do grafických objektů jsem úspěšně ověřil na několika rozdílných případech. Do budoucna jsou možnosti téměř neomezené a komponenty chci i dále rozšiřovat. Příštím krokem je naprogramování rastrového editoru a jeho spolupráce s grafickým objektem s podporou obrázků. Rozšířením práce mohou být nové grafické objekty například beziérovy křivky a mnohoúhelníky.

Možnost použití komponent a tříd v rozličných typech aplikací a dobré předpoklady pro rozšíření do budoucna jsou hlavní rysy této práce.

Literatura

[1] Mistrovství v Delphi 2, Marco Cantú, Computer Press 1996

[2] Myslíme v jazyku Delphi 6, 1. Díl, Marco Cantú, Grada 2002

[3] Myslíme v jazyku Delphi 6, 2. Díl, Marco Cantú, Grada 2002

[4] Myslíme v jazyku Delphi 7, Marco Cantú, Grada 2003

[5] Delphi hotová řešení, Václav Kadlec, Computer Press 2003

[6] Komponenty v Delphi, Josef Pirkl, Computer Press 2002

[7] Interbase / Firebird, Pavel Císař, Computer Press 2003

[8] Microsoft SQL server 2005, Computer Press 2006

[9] MySQL Průvodce základy databázového systému, Luke Welling a Laura Thompson, Computer Press 2005

[10] MySQL Oficiální průvodce tvorbou, správou a laděním databáze, Robert D. Schneider, Grada 2005

[11] 1001 Tipů a triků pro Delhi, Svoboda, Voneš, Konšal, Mareš, Computer Press 2003

[12] Základy objektově orientovaného návrhu v UML, Meilir Page–Jones, Grada 2002

[13] Delphi průvodce vývojáře, Steve Teixeira & Xavier Pachco, UNIS Publishing 1999