# Rozšíření editoru formulářů programového systému NAHOS pro řízení nástrojového hospodářství

Bc. Antonín Výborný

Diplomová práce
2010



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2009/2010

# ZADÁNÍ DIPLOMOVÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

| | |
|---|---|
| Jméno a příjmení: | **Bc. Antonín VÝBORNÝ** |
| Studijní program: | **N 3902 Inženýrská informatika** |
| Studijní obor: | **Informační technologie** |
| Téma práce: | **Rozšíření editoru formulářů programového systému NAHOS pro řízení nástrojového hospodářství.** |

Zásady pro vypracování:

1. **Prostudujte stávající verzi editoru formulářů systému NAHOS**
2. **Navrhněte rozšíření editoru o obsluhu nových prvků formuláře – Menu, Chart, TabPanel, GraphicArea, IconLabel**
3. **Navrhněte doplnění editoru o možnosti grafické editace velikosti jednotlivých prvků a skupin prvků pomocí myši**
4. **Navrhněte doplnění editoru o možnost zobrazení gridu při návrhu formuláře**
5. **Navrhněte doplnění editoru o schopnost zachovat a vložit komentáře do generovaného XML dokumentu**
6. **Navržená rozšíření a doplnění editoru implementujte v jazyce Java a začleňte zpracované programové moduly do stávajícího editoru systému NAHOS**

Rozsah práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:


1. **Horton, Ivor: Java 5, Neocortex, Praha 2005, ISBN 80-86330-12-5**
2. **Virius, Miroslav: Java pro zelenáče, Neocortex, Praha 2005, ISBN 80-86330-17-6**
3. **Brůha, Lubomír: Java Hotová řešení, Computer Press, Brno 2006, ISBN 80-251-0072-3**
4. **Hynar, Martin: Java nástroje, Neocortex, Praha 2004, ISBN 80-86330-16-8**
5. **Vařecha, Martin: Diplomová práce - Programové vybavení pro evidenci nářadí ve strojírenské výrobě - vzdálená správa databáze nářadí, 2003**
6. **Verbovský, Jakub: Diplomová práce - Software for tools records an Engineering industry - communication with user, 2003**
7. **Svoboda, Michal: Diplomová práce - Rozšíření programového systému pro správu nástrojového hospodářství ve strojírenské výrobě, 2004**
8. **Hlaváček, Marek: Diplomová práce - Rozšíření programového systému NAHOS pro řízení nástrojového hospodářství, 2007**


Vedoucí diplomové práce: **doc. Ing. Lubomír Vašek, CSc.**
Ústav automatizace a řídicí techniky

Datum zadání diplomové práce: **19. února 2010**

Termín odevzdání diplomové práce: **8. června 2010**

Ve Zlíně dne 19. února 2010

prof. Ing. Vladimír Vašek, CSc.
*děkan*

L.S.

prof. Ing. Vladimír Vašek, CSc.
*ředitel ústavu*

## ABSTRAKT

Cílem mé diplomové práce je rozšíření editoru formulářů informačního systému nástrojového hospodářství ve strojírenském podniku Tajmac-ZPS Zlín, a.s. Měl jsem za úkol rozšířit editor formulářů o obsluhu nových prvků – *Menu*, *Chart*, *TabPanel*, *GraphicArea* a *IconLabel*, a také doplnit editor o další nové funkce. Jedná se zejména o možnosti grafické editace velikosti prvků formuláře pomocí myši, zobrazování mřížky, k níž jsou jednotlivé prvky formuláře přichycovány a možnost vkládání a uchování komentářů do generovaného XML dokumentu. Rozšíření bylo naprogramováno v programovacím jazyce Java v prostředí *NetBeans*. Práce obsahuje část teoretickou, která popisuje informační systém NAHOS a vyžívané technologie, především samotný jazyk Java. Další částí je část praktická, popisující rozšíření editoru formulářů o nové prvky a funkce z hlediska návrhu i řešení.

Klíčová slova: Informační systém, Java, XML, editor formulářů.

## ABSTRACT

The aim of my thesis is to extend the Form Editor of tools management information system in the engineering company Tajmac-ZPS Zlín, a.s. It was necessary to extend the Form Editor of operation of new elements - *Menu*, *Chart*, *TabPanel*, *GraphicArea* and *IconLabel* and also implement new features. These include the possibilities of graphic editing of size of the elements with the mouse, displaying grid to which are various elements attached and possibility of inserting and keeping comments to the generated XML document. Extension has been programmed in the Java programming language in the *NetBeans* environment. The work contains a theoretical part, which describes the information system NAHOS and technologies, especially the Java language. Another part is the practical part that describes the extension of Form Editor of the new features and functions to design and solution.

Keywords: Information system, Java, XML, Form Editor.

Na tomto místě bych rád poděkoval vedoucímu mé diplomové práce panu doc. Ing. Lubomíru Vaškovi a panu Ing. Markovi Hlaváčkovi za odborné vedení, cenné rady a připomínky, které mi poskytovali při řešení diplomové práce.

**Prohlašuji, že**

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

**Prohlašuji,**

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně …….

podpis diplomanta

# OBSAH

# ÚVOD

V dnešní době, kdy se informační technologie rozšířily téměř do všech odvětví lidské činnosti, jsou důležitou součástí všech podniků informační systémy. Ty podnikům umožňují získat vyšší bezpečnost dat, rychlejší odezvu i lepší kontrolu nad všemi automatizovanými procesy. Může se jednat o řízení výroby a obchodu, vedení účetnictví, mezd, ale i správu dokumentů či jiných libovolných dat ukládaných nejčastěji v databázi. Díky tomu firmy získávají výhody na trhu a mohou tak lépe obstát v konkurenci.

Tato práce se zabývá rozšířením informačního systému nástrojového hospodářství ve strojírenském podniku Tajmac-ZPS Zlín, a.s. Součástí systému je editor formulářů, umožňující vytvářet potřebné klientské formuláře, které uživatelům zjednodušují a zpříjemňují práci s daty o nástrojích používaných ve strojírenském výrobním systému.

Mým úkolem je navrhnout a implementovat nové funkce editoru, poskytující uživatelům nejen lepší kontrolu nad vzhledem i funkčností formulářů, ale i vyšší komfort při jejich vytváření a úpravách. Bude tedy umožněno měnit velikost komponent jednoduše pomocí myši a zobrazovat mřížku, ke které mohou být komponenty přichytávány. Dalším úkolem je rozšíření o obsluhu nových prvků. Jedná se například o prvky *Menu*, *Chart*, *GraphicArea* a další. Je také potřeba editor doplnit o možnost vkládání a zachovávání komentářů do generovaného XML dokumentu.

Diplomová práce navazuje na již poměrně rozsáhlou aplikaci, která je výsledkem práce několika předchozích diplomantů. Aplikace je napsána v jazyce Java, data jsou uložena v databázi, která pro jejich správu umožňuje použití dotazovacího jazyka SQL. Pro ukládání formulářů systém využívá technologii moderního značkovacího jazyka XML. Díky těmto technologiím je zajištěno, že je systém nezávislý na cílové platformě, na níž může být nainstalován.

# I. TEORETICKÁ ČÁST

# 1 INFORMAČNÍ SYSTÉMY

Stejně jako pronikají informační technologie do různých oblastí života, stávají se informační systémy velmi důležitou součástí všech firem a organizací, které mají snahu udržet se na trhu a obstát v konkurenčním boji. Díky informačním systémům mohou automatizovat nejrůznější způsoby práce se všemi daty a informacemi, se kterými přichází firma do styku. V nejjednodušších případech se může jednat pouze o telefonní seznamy nebo archivaci určitých dokumentů, jakými jsou například lékařské záznamy. Může jít ale také o kompletní správu účetnictví, objednávkový systém prodejny či automatizovaný systém výroby. Informační systémy jsou však většinou komplexnější a umožňují starat se o velkou část nejrůznějších podnikových i lidských činností. Starají se především o data a informace, jejich uchovávání, analýzu, zpracování, vyhodnocování i jejich prezentaci. Většina dat, se kterými informační systémy pracují, je uchovávána v databázích. Ty často neposkytují uživatelsky přívětivé prostředí, k prezentaci bývá použito převážně jen seznamů a tabulek a neumožňují plnohodnotnou a efektivní práci s nimi. Pro tyto účely je třeba použít distribuované a vícevrstvé aplikace.

S postupem času a s přicházejícími novými technologiemi rostou zároveň i požadavky na možnosti informačních systémů. Nejkomplexnějším řešením s největšími možnostmi bývají komerční systémy, které však nejsou vždy nejvhodnější, zejména kvůli jejich vysoké ceně a složitosti, která vyžaduje nákladné školení uživatelů. Proto může být vhodnou volbou vytvoření nového systému, který splňuje všechny zadané požadavky a zároveň je dostatečně flexibilní, aby umožňoval případné rozšíření a úpravy. Jedním z takových systémů je informační systém NAHOS, který je zatím systémem vcelku novým a mladým a při jeho provozu jsou zjišťovány nedostatky, které je třeba vyřešit, a také požadavky nové, o které je systém třeba rozšířit.

## 1.1 Architektura informačních systémů

Při vývoji téměř všech rozsáhlých aplikací je nejdůležitějším aspektem podrobná analýza zadání, jejíž součástí je volba typu architektury. Jedná se o nejzásadnější proces celého vývoje aplikace. Pokud je návrh nevhodný, může se stát, že bude vývoj zdlouhavý a náročný, v horším případě může celý projekt ztroskotat. Architektura vychází zejména z požadavků zadavatele projektu. Naprostá většina informačních systémů má architekturu rozdělenou do několika samostatných celků – vrstev, díky nimž jsou systémy snadněji udržované a přenositelné.

### 1.1.1 Rozdělení vrstev

#### *1.1.1.1 Prezentační vrstva*

Prezentační vrstva (klientská) se stará zejména o vstup dat a zobrazování výsledků. Jedná se o uživatelské rozhraní, které umožňuje komunikovat s aplikační vrstvou a má v sobě zabudovanou určitou logiku tohoto rozhraní.

#### *1.1.1.2 Aplikační vrstva*

Tato vrstva má za úkol komunikaci mezi prezenční vrstvou a vrstvou datovou. Zabývá se také zpracováním dat na požadovanou formu, výpočty a logikou práce s daty. Pokud je systém postaven na dvouvrstvé architektuře, stává se běžně součástí prezentační, nebo někdy i datové vrstvy jako jeden celek.

#### *1.1.1.3 Datová vrstva*

Hlavním účelem je logika skladování dat, především jejich ukládání a čtení. Většinou jsou data uchovávána v relačních databázích, nebo přímo na disku počítače či jiného externího zařízení.

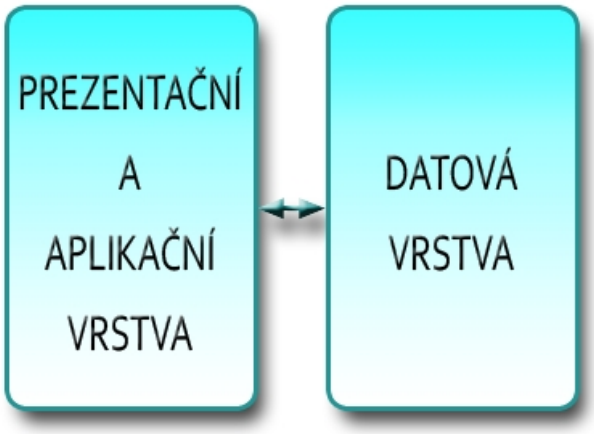
Hranice mezi těmito třemi částmi nejsou vždy jasné a je třeba počítat s tím, že veškerá aplikační logika nemusí být přímo v aplikační vrstvě, ale měla by se tam nacházet alespoň většina. Nejvýhodnější bývá použití architektury třívrstvé, lze se však setkat i se spíše starší dvouvrstvou. [4, 8, 10]

### 1.1.2 Dvouvrstvá architektura

Dvouvrstvá architektura vychází z vrstvy klientské (prezentační) a vrstvy serverové (datové). Aplikační logika je většinou obsažena v klientské části a pracuje přímo nad datovým zdrojem. V mnoha scénářích je zdroj dat reprezentován relační databází a klient využívá jazyk SQL pro práci s daty. Mezi další typy datového zdroje patří soubory anebo jiné aplikace např. informační systémy. [9]

#### *1.1.2.1 Klient – server se vzdálenými daty*

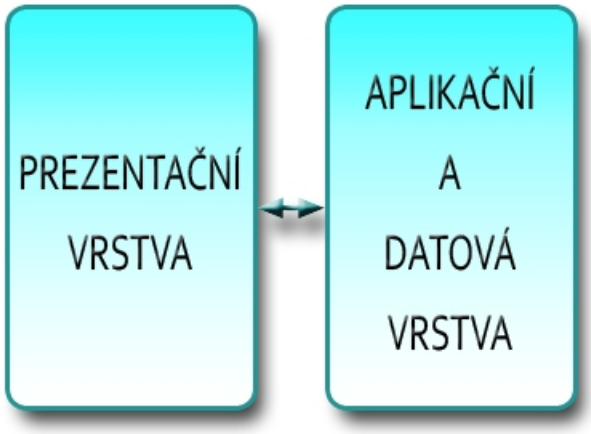
Jedná se o takzvaný souborový server a dodnes se používá především v případech, kdy je logika práce s daty jednoduchá a univerzální. Největší nevýhodou je velká zátěž klientské části a přenosového kanálu mezi vrstvami, naopak výhodou je menší zatížení serveru. [11]




*Obr. 1 – Klient – server se vzdálenými daty*

#### *1.1.2.2 Klient – server se vzdálenou prezentací*

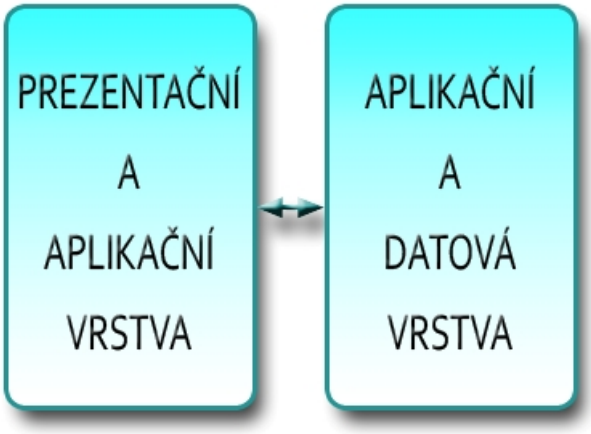
Oproti předchozí variantě méně zatěžuje klienta a přenosový kanál, ale kvůli tomu, že je aplikační vrstva součástí vrstvy datové (serverové), podstatně více zatěžuje server. Zajímavostí je, že je z klientské části na první pohled nerozlišitelná od architektury třívrstvé. Používá se především u jednoduchých webových aplikací. [11]




*Obr. 2 – Klient – server se vzdálenou prezentací*

#### *1.1.2.3 Klient – server s rozdělenou logikou*

Zátěž klientské i serverové části je rozložena, což může být v některých případech výhodou, příliš se však nepoužívá kvůli horší rozšiřitelnosti a přenositelnosti. [11]




*Obr. 3 – Klient – server s rozdělenou logikou*

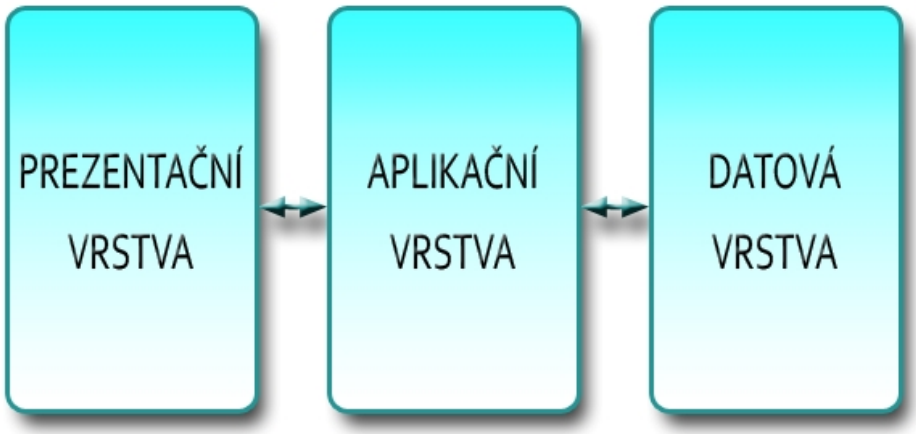
Při vzrůstající komplexnosti klientských aplikací vyšly na povrch nevýhody dvouvrstvých modelů. Se složitostí vzrůstaly výkonové nároky na klientské počítače, což byl jeden faktor. Další faktory přímo vyplynuly z masivního rozšíření aplikací, softwarové firmy byly nuceny pružně reagovat na poptávku a tím pádem se snažit operativně plnit přání zákazníků. Tak byly definovány požadavky na budoucí aplikace. Například nutnost sdílení

zdrojů, omezení datového přenosu atd., díky kterým se začalo uvažovat o architektuře, která by tyto požadavky pokryla.

Řešení se našlo v podobě přidání třetí vrstvy. Samozřejmě se nejedná pouze o její přidání, je nutno specifikovat role jednotlivých vrstev v architektuře. [9]

### 1.1.3 Třívrstvá architektura

Dvouvrstvý model není příliš výhodný, protože aplikace je přímo závislá na datech. Při jakékoliv změně způsobu ukládání dat je potřeba přeprogramovat nastavení aplikace. S příchodem třívrstvé architektury se změnil význam jednotlivých vrstev. Díky nově definované aplikační vrstvě je možné aplikační logiku, která do té doby ležela na klientu, přesunout na aplikační server. To zapříčiní výrazné zvýšení efektivity, neboť veškerý výpočetní výkon je přesunut na výkonné servery. Dalším významným krokem je redukce datového přenosu, jehož je těžiště přesunuto na trasu mezi aplikačním serverem a datovým zdrojem. Spojení aplikačního serveru a datového zdroje je možno realizovat vhodným přenosovým připojením. Díky oddělení prezentační a aplikační logiky je navíc možné částečně předcházet nekvalitní implementaci.




*Obr. 4 – Třívrstvá architektura*

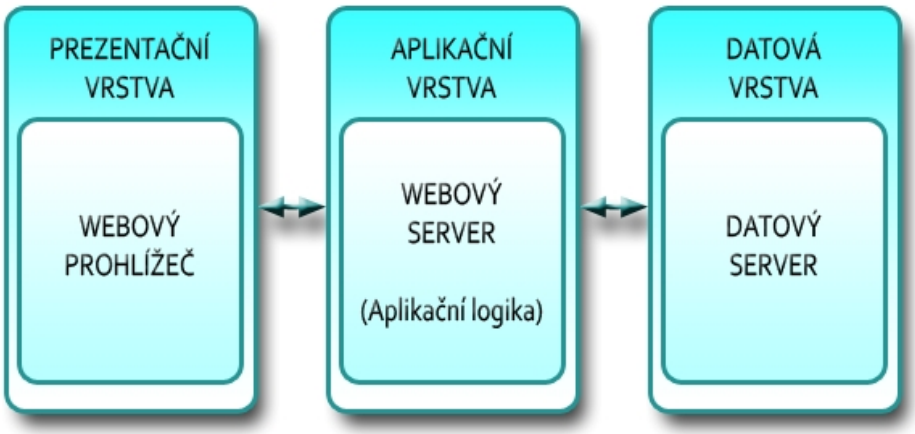
Výhody třívrstvé aplikace jsou zřejmé hlavně z dlouhodobého hlediska, kdy je potřeba aplikaci neustále vyvíjet a aktualizovat, neboť tento model je oproti dvouvrstvému o mnoho pružnější, lépe se udržuje a je také bezpečnější. To zejména proto, že oddělení

prezentace od zbytku aplikace umožňuje vyšší úroveň zabezpečení, kdy nemají koncoví uživatelé k datům přímý přístup. [8, 9]

#### *1.1.3.1 Tenký klient*

Jedná se o nejjednodušší typ klienta. V principu jde o to, že prakticky vše se zpracovává na straně serveru. Aplikace na klientské straně má pouze omezené možnosti, zpravidla se jedná o webové rozhraní, které se stará o zobrazování dat zpracovaných serverem. Výkon aplikace je pak dán především jen výkonem serveru a hardware na klientském počítači neklade žádná omezení, což může být z klientského hlediska výhodné. Další výhodou je jednoduchost instalace. Téměř všechny operační systémy mají webové prohlížeče zabudované a není tedy potřeba prezentační vrstvu implementovat.

Mezi největší nedostatky patří nutnost časté komunikace s aplikační vrstvou. Při jakékoliv uživatelské akci je zaslán požadavek na server, který po vyhodnocení vrací odpověď prohlížeči. Z toho plyne pomalejší chod celé aplikace a nutnost delšího čekání na odezvu. Grafické rozhraní je značně omezené a neumožňuje dosahovat takových výsledků, jako umožňují grafické knihovny programovacího jazyka. [4]

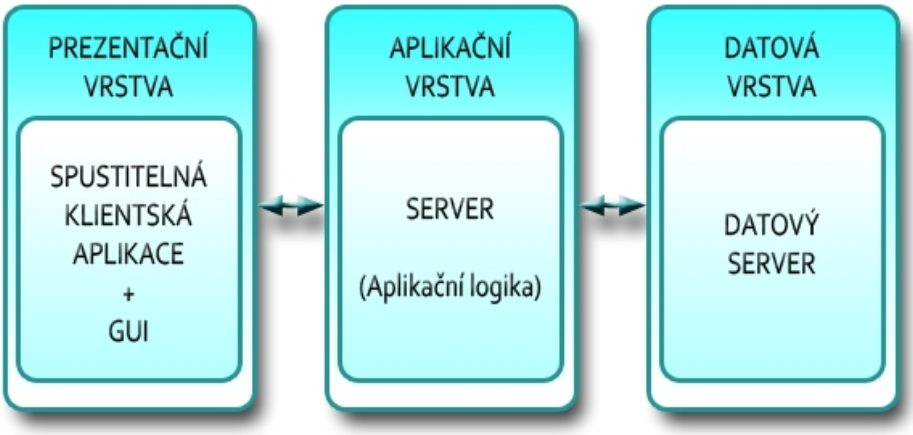



*Obr. 5 – Tenký klient*

#### *1.1.3.2 Tlustý klient*

Tlustý klient představuje aplikaci, která běží na straně hardwaru klienta. Jako taková je pak omezována možnostmi operačního systému a výkonem hardwaru. To už postupem času ale přestává být problémem, neboť výkon hardwaru již většinou převyšuje softwarové

nároky. Tlustý klient má vlastní logiku, která s aplikační vrstvou komunikuje jen při potřebě získání nebo uložení údajů. Dále umožňuje využít všech dostupných možností grafických knihoven, například v případě jazyka Javy a její knihovny Swing jsou možnosti opravdu rozsáhlé. Nutnosti je však vytvoření klientské aplikace a její následná instalace. [4]




*Obr. 6 – Tlustý klient*

# 2 POUŽITÉ TECHNOLOGIE

## 2.1 Java

Programovací jazyk Java je dnes již plně vyzrálý a objektově orientovaný jazyk, vyvinutý společností Sun Microsystems. Ta jej představila v roce 1995 a za dobu své existence si vytvořila velkou řadu příznivců z řad programátorů, ale také mnoho odpůrců. O oblíbenosti Javy svědčí mimo jiné fakt, že se jazyk velmi rychle rozšířil a již dlouhou dobu se řadí mezi nejpoužívanější programovací jazyky na světě. O původu názvu Java, panuje mnoho různých pověr, nejznámější a nejpravděpodobnější je ta, která říká, že se jedná o tmavý, kávě podobný nápoj, který pijí programátoři, aby se v nocích, kdy píšou nekonečné řádky kódu, udrželi při vědomí.

Díky své přenositelnosti je Java používána nejen pro klasické počítačové aplikace, ale je také často využívána u nejrůznějších systémů, jako jsou mobilní telefony, čipové karty nebo dokonce i pro programování internetových aplikací. Není žádným tajemstvím, že právě Java bývá často používána i pro bankovní systémy, zejména pro její vysokou bezpečnost. [1, 9]

### 2.1.1 Výhody

- **Jednoduchost** – Java je jednoduchý jazyk, částečně vycházející ze syntaxe jazyka C++. Díky tomu je přechod mezi těmito programovacími jazyky jednodušší. Jsou odstraněny nehezké příkazy jako *goto*, hlavičkové soubory i ukazatelová aritmetika. Java tak předchází velkému množství chyb vyskytujících se často při programování v jiných jazycích. Naopak přibylo mnoho užitečných konstrukcí a rozšíření.
- **Objektová orientace** – Od počátku je Java vyvíjena jako objektově orientovaný jazyk. Třídy jsou uspořádány v hierarchii od obecných ke konkrétnějším.
- **Distribuovanost** – Programovací jazyk podporuje různé úrovně síťového spojení i práci se vzdálenými objekty, kterým je poskytována prakticky stejná funkčnost jako při práci s lokálně uloženými daty.
- **Interpretace** – Zdrojové kódy nejsou překládány přímo do strojového kódu, nýbrž do tzv. bajtového kódu. Takto vytvořené soubory jsou nezávislé na jakékoliv architektuře či zařízení. Tím je zajištěná plná přenositelnost programu. S programem

tak lze pracovat na jakékoliv platformě, na jakémkoliv počítači nebo zařízení, na kterém je nainstalován tzv. virtuální stroj Javy.

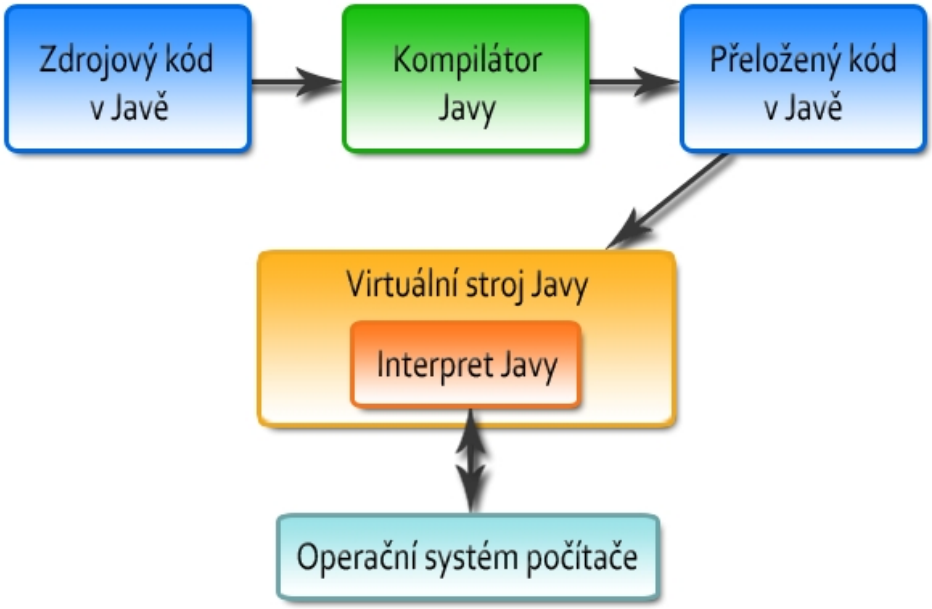
- **Robustnost** – Jazyk je navržen tak, aby umožňoval psaní vysoce spolehlivého a robustního softwaru. Java je silně typovým jazykem, kdy veškeré proměnné musí mít přesně definovaný datový typ, díky čemuž je možné odhalit chyby již během překladu do bajtového kódu. Velkou výhodou je také existence správce paměti (garbage collector) který automaticky vyhledává již nepoužívané části paměti a uvolňuje je pro další použití.
- **Bezpečnost** – Java obsahuje bezpečnostní mechanismy, chránící počítač v síťovém prostředí, na kterém je program zpracováván před nebezpečným kódem pracujícím jako virus nebo červ.
- **Nezávislost na architektuře** – díky překladu do bajtového kódu běží vytvořená aplikace na libovolném operačním systému nebo libovolné architektuře. Pro spuštění programu je ale potřeba, aby byl na dané platformě instalován virtuální stroj. Vzhled aplikací zle přizpůsobit cílové platformě.
- **Výkonnost** – Přesto, že je Java jazykem interpretovaným, rychlost programů je dostačující a je umožněno dynamicky překládat za běhu aplikace pouze tu část programu, která je zrovna potřeba.
- **Podpora vláken** – Tvorba více-vláknových aplikací je dnes běžná záležitost a Java má podporu vláken již ve svém základu.

### 2.1.2 Nevýhody

- **Pomalost** – Oproti programovacím jazykům, které provádějí překlad do strojového kódu, je start běhu programů napsaných v Javě pomalejší. Dnes však již existují mechanismy, které určité části kódu přeloží do strojového kódu a díky tomu se program zrychlí.
- **Paměťová náročnost** – Programy napsané v Javě spotřebovávají při běhu aplikace větší množství paměti, neboť je nutné, aby v paměti počítače bylo také běhové prostředí Javy. Jedná se zejména o drobnější aplikace, u těch rozsáhlejších již rozdíly nejsou tak markantní. [2, 9]

### 2.1.3 Virtuální stroj Javy (JVM)

Na obrázku (Obr. 7) je zobrazen celý proces překladu, až po komunikaci s operačním systémem. Nejprve je napsán zdrojový kód, který posléze kompilátor přeloží do bajtového kódu. Takovému kódu rozumí interpret Javy, jenž je součástí virtuálního stroje JVM (Java Virtual Machine). Interpret zkontroluje a rozšifruje bajtový kód, a potom ve virtuálním stroji provede akce, kódem určené. Veškerá komunikace s počítačem (operačním systémem) je zajištěna interpretem. [1]




*Obr. 7 – Způsob práce Javy*

## 2.2 XML

Technologie XML (eXtensible Markup Language, česky rozšiřitelný značkovací jazyk) je univerzálním jazykem, který slouží především ke komunikaci mezi různými aplikacemi a pro publikování dokumentů. Vychází ze staršího značkovacího jazyka SGML a byl vyvinut uznávaným konsorciem W3C.

XML popisuje, co data jsou a jaký je jejich obsah, ale jak mají být zobrazeny, přímo neurčuje. Může však obsahovat informace, například pomocí atributů, mající částečnou kontrolu nad výsledným vzhledem a prezentací dat.

Struktura dokumentu je popsána pomocí tzv. tagů (značek). Na rozdíl od příbuzného jazyka HTML nejsou tagy pevně definovány a při tvorbě dokumentů máme

zcela volnou ruku. Jakýkoliv obsah dokumentu popsaného pomocí XML musí být tagy obalen, musí tedy mít počáteční i koncovou značku. Obě tyto značky spolu s obsahem tvoří tzv. element. Obsahem elementu můžou být jakákoliv data, včetně dalších elementů. Díky takovému vnořování vzniká hierarchická struktura a je přesně dáno, který element je rodičem a který potomkem. Je však nutné, aby se jednotlivé elementy nikdy nepřekrývaly. Elementy mohou zahrnovat také libovolný počet atributů, které mohou upřesňovat a rozšiřovat informace o svém obsahu. [1, 9]

Důležitou součástí XML dokumentů je také hlavička. Ta obsahuje XML deklaraci, která oznamuje, že se jedná právě o dokument XML, a musí být umístěna hned na začátku souboru. Následovat může nepovinná část, a sice deklarace typu dokumentu – DOCTYPE, jenž přiřazuje souhrn pravidel, které definují tagy a jejich vztahy.

Ukázka jednoduchého XML dokumentu může vypadat například takto:

```
<?xml version="1.0" encoding="Windows-1250"?>
<clanek>
  <!-- ukázka komentáře -->
  <autor>Antonín Výborný</autor>
  <datum>14.3.2008</datum>
  <nazev>Struktura dokumentu XML</nazev>
  <odstavec zarovnani="vlevo">Text článku, zarovnaný vlevo</odstavec>
</clanek>
```

Pomocí XML jsou v aplikaci nástrojového hospodářství uchovávány struktury jednotlivých uživatelských formulářů v databázi nebo XML souborech a mohou být poté znovu načteny a zobrazovány v aplikaci ve vizuální i funkční podobě. V editoru formulářů jsou tyto dokumenty vytvářeny a editovány. Součástí mé diplomové práce je umožnit vkládání a zachovávání komentářů do generovaného XML dokumentu, a také úprava jejich generování, neboť bude přidáno několik nových prvků a funkcí formulářů.

### 2.2.1 Parsování XML

Parsováním nazýváme činnost, která zpracovává libovolný dokument a převádí jej na datovou strukturu, se kterou programovací jazyky dovedou pracovat a rozumí jí. Zpracovávaný dokument je kontrolován z hlediska jeho skladby a prvky v něm obsažené jsou vybírány dle určité předem stanovené normy. Parser je program, nebo programový

modul, který umožňuje parsování. Musí rozumět struktuře dokumentu a mít přesně nadefinované, jak se má chovat při nalezení určité části dokumentu (může se jednat o značku, textovou část nebo i předem definovaný symbol).

V zásadě existují dva hlavní postupy, jak lze dokument parsovat. Postupným procházením (*SAX*), nebo uložením dokumentu do stromové struktury (*DOM*). [1]

#### *2.2.1.1 SAX*

V případě velmi rozsáhlých dokumentů, je prakticky nemožné nebo nevýhodné načítat celý dokument do paměti. V takových situacích je nejvhodnějším způsobem použít řešení založené na událostech. Nejpoužívanějším API s takovýmto přístupem je *Simple API for XML (SAX)*. Umožňuje sériový přístup k XML, kdy je dokument rozdělen na jednotlivé části (značky, obsahy elementů, komentáře…). Parser při zpracování dokumentu vygeneruje sérii událostí, jejichž zpracování je pak plně v kompetenci programátora. Výhody tohoto způsobu spočívají zejména v nízké paměťové náročnosti a rychlosti zpracování. Nevýhodou je však sekvenční přístup, který neumožňuje takovou pružnost práce nad dokumentem jako metody založené na náhodném přístupu. Ze způsobu, jakým SAX pracuje, vyplývá skutečnost, že příjem dokumentu se děje po částech, bez představy o celku.

#### *2.2.1.2 DOM*

Druhá metoda zpracování XML souborů spočívá v načtení celého dokumentu do paměti a uložením v podobě stromové struktury, která je aplikaci vrácena jako objekt typu *Document*. Objektově orientovaná reprezentace takové struktury nazývána *Dokument Object Model (DOM)*. Výhodou stromové struktury je umožnění přistupovat k libovolné části dokumentu nezávisle na částech ostatních, a tím pádem také rychlá úprava menšího množství dat. Na rozdíl od *SAX* umožňuje zpracování v *DOM* měnit existující soubory nebo vytvářet nové. Při vzrůstající velikosti dokumentu se však objevují spíše nevýhody, neboť je potřeba mít neustále celý dokument načtený v paměti a parser spotřebuje značnou část času při převodu dokumentu na stromovou strukturu a zpět.

#### *2.2.1.3 JAXP*

Většina parserů v programovacím jazyce Java využívá právě *DOM* nebo *SAX*, přesto nebylo jednoduché vyměnit jednu implementaci parseru za druhou. Z toho důvodu vznikla

specifikace *Java API for XML Processing* (*JAXP*), která definovala rozhraní pro parsování a manipulaci s XML dokumenty a umožňuje také jednoduše měnit typ parseru.

# II. PRAKTICKÁ ČÁST

# 3 POUŽÍVANÝ SOFTWARE

## 3.1 NetBeans

*NetBeans* je open source projekt, který byl nejprve vytvořen v Praze, poté odkoupen firmou *Sun Microsystems*. Projekt se dělí na dva hlavní produkty: vývojové prostředí *NetBeans IDE* a vývojová platforma *NetBeans*.

### 3.1.1 NetBeans IDE

Vývojové prostředí je primárně určeno pro vytváření aplikací pomocí jazyka Java, pomocí kterého bylo celé prostředí naprogramováno. V současnosti některé verze obsahují také podporu dalších programovacích jazyků, jako například C/C++, Python, PHP, JavaScript a mnoho dalších. Jedná se o nástroj, pomocí kterého můžou programátoři psát, překládat, ladit a šířit programy. Nabízí programátorovi prakticky vše, co potřebuje pro pohodlné multiplatformní programování, včetně komponenty pro snadný návrh grafického rozhraní pomocí knihovny Swing. Prostředí NetBeans IDE je možno používat na operačních systémech Windows, Linux, Mac OS X a Solaris. [9]

### 3.1.2 Hlavní rysy

- **Formátování kódu** – pomáhá získat určitou, předem specifikovanou, úpravu kódu, umožňující programátorům lépe se v něm orientovat a rychleji tak vyhledávat možné chyby.
- **Zvýraznění syntaxe** – pomocí předdefinovaných šablon jsou příkazy, proměnné, komentáře a různé další části kódu barevně odlišovány, což zvyšuje čitelnost kódu.
- **Oprava chyb** – *NetBeans* umožňuje vyhledávat některé typy chyb a nabízí také možnosti jejich oprav.
- **Doplňování kódu** – pomocí předdefinovaných zkratek je možno doplnit kód klidně i o celý blok příkazů. Zkratky jsou plně editovatelné.

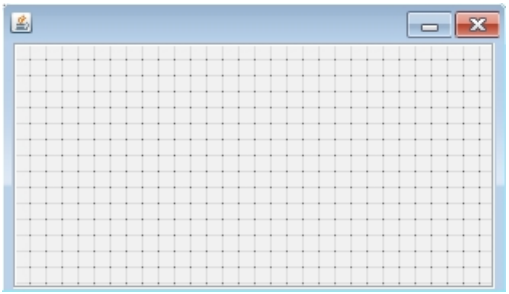
# 4 ROZŠÍŘENÍ EDITORU FORMULÁŘŮ

## 4.1 Zobrazení mřížky při návrhu formuláře

Jedním z nedostatků editoru formulářů bylo, že prvky které se umísťovaly na formulář, se neformátovaly příliš snadno. Zarovnání prvků sice již bylo v editoru obsaženo, ale uživatel neměl přehled, kam prvek vkládá a práce s ním tak byla vcelku krkolomná. Pro zjednodušení práce s formuláři bylo proto potřeba vytvořit mřížku, pomocí které by bylo přehlednější, kam daný prvek vložit.

### 4.1.1 Návrh mřížky

Při návrhu bylo důležité především zvážit, jak bude výsledná mřížka vypadat. Inspirací můžou být grafické editory formulářů profesionálních programovacích prostředí jako má například prostředí C++ Builder. Ten zobrazuje mřížku pomocí rovnoměrně a pravidelně rozmístěných jedno-pixelových bodů. Další možností je zobrazení pomocí vodorovných a svislých čar. Rozdíl mezi oběmi variantami, je pouze estetický. Po vyzkoušení obou způsobů byla nakonec vybrána kombinace světlejších čar a tmavších bodů v místě protínání vodorovných a svislých čar. Ukázka takového řešení je zobrazena na obrázku (Obr. 8).



*Obr. 8 – Mřížka formuláře*

### 4.1.2 Implementace mřížky

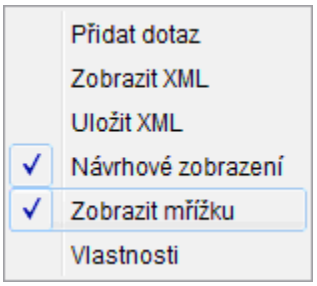
Pro vykreslení libovolného obsahu na plochu formulářového okna bylo potřeba přeprogramovat původní vykreslovací metodu *paint* ve třídě *FormBody*. Metodu *paint* obsahují všechny vizuální komponenty grafické knihovny a slouží k vykreslování zobrazovaných prvků. Jejím přepsáním lze docílit, že bude daná komponenta vykreslena jinak, než bývá zvykem, což je přesně to, co potřebujeme.

Samotné vykreslení mřížky probíhá tak, že pomocí zadané rozteče v cyklech vykreslí světlejší barvou vodorovné a svislé čáry a tmavší barvou body v místech průsečíků.

Kvůli možnosti vypínání zobrazení mřížky je potřeba uchovávat v paměti logickou proměnou, určující její aktuální stav.

### 4.1.3 Možnosti mřížky

Protože může mřížka při návrhu formulářů někdy působit rušivě, je velmi vhodné mít nějaký pohotový nástroj pro její vypínání. Řešením pak jsou položky v menu a klávesové zkratky nebo kontextové nabídky (Obr. 9). V ideálním případě program zahrnuje všechny tři případy a je jen na zvyku uživatele, které možnosti využije. Mřížka je samozřejmě zobrazena pouze v návrhovém zobrazení.




*Obr. 9 – Kontextová nabídka s možností mřížky*

### 4.1.4 Klávesové zkratky a menu „Úpravy“

Pro zjednodušení práce s mřížkou byla přidána také podpora klávesových zkratek, která uživatelům pomůže zvýšit efektivitu a produktivitu práce. Díky nově přidaným úpravám přibyla v menu editoru formulářů nová položka *Úpravy*, jež obsahuje přepínače *Zobrazit mřížku* a *Návrhové zobrazení* s příslušnými klávesovými zkratkami a položku *Vlastnosti*.

**Klávesové zkratky:**

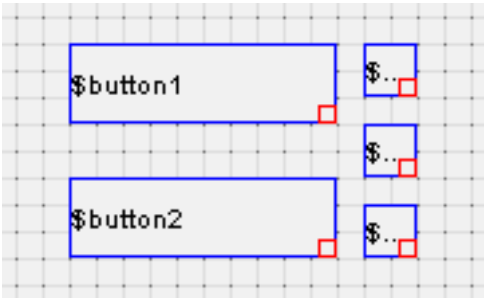
- **M** – zapínání a vypínání zobrazení mřížky
- **N** – přepínání mezi návrhovým zobrazením a náhledem
- **V** – zobrazení okna vlastností formuláře

## 4.2 Změna rozměru komponenty pomocí myši

Doposud bylo jedinou možností, jak změnit velikost komponenty, přepsat číselný údaj ve vlastnostech komponenty. Zadávání rozměru pomocí šířky a výšky je sice přesné, ale hodně zdlouhavé a nepohodlné. Bylo proto třeba vytvořit rychlejší a pohodlnější metodu, která umožní efektivněji měnit rozměr komponent.

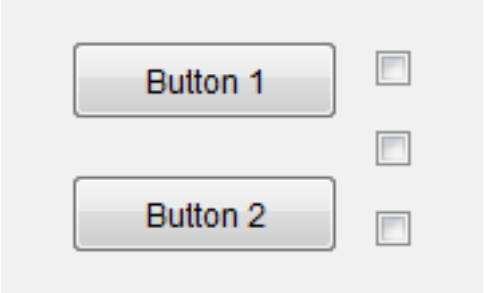
### 4.2.1 Návrh

Existuje více způsobů, jak měnit velikosti prvků formuláře. Prvotním nápadem bylo měnit rozměr táhnutím myši za okraj prvku. Představa byla inspirována změnou velikostí oken operačního systému Windows. Takový způsob má velkou výhodu v tom, že můžeme prvek protáhnout do libovolného směru. Uchycením za rožek komponenty by pak bylo možno měnit rovnou do dvou rozměrů. U oken ve Windows, či jiných operačních systémech s grafickým rozhraním, je takovýto způsob výhodný a pohodný. U prvků formuláře však nikoliv, neboť mají velmi malé rozměry a jakékoliv operace, ať už posunutí nebo změna rozměru by byla velmi nepraktická. Bylo tedy potřeba vymyslet jiný postup, vhodný zejména pro drobné prvky. Jedna z možností je taková, že bude změna velikosti aktivována až po stisku nebo přidržení nějaké klávesy či klávesové zkratky, poté by byla funkčnost prakticky stejná jako v prvním návrhu. Druhou možností je měnit šířku a výšku pouze pomocí jednoho rohu a zbytek komponenty ponechat pro posun. Z obou nových návrhů je přijatelnější druhý způsob neboť práce s ním je rychlejší a pohodlnější. Na obrázku (Obr. 10) je k vidění návrhové zobrazení, modře je ohraničena celá komponenta, červený čtvereček v rohu slouží pro změnu velikosti pomocí myši. Na obrázku jsou dvě klasická tlačítka a tři zaškrtávací políčka. Pro porovnání jsou na obrázku (Obr. 11) tytéž komponenty při náhledu.




*Obr. 10 – Změna velikosti komponent*

*(Návrhové zobrazení)*




*Obr. 11 – Změna velikosti komponent*

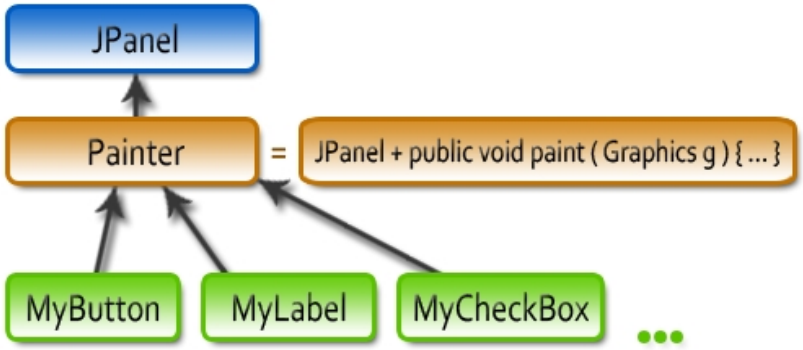
*(Náhled)*

Jak již bylo naznačeno, pro zmenšení nebo zvětšení komponenty je třeba použít metodu zvanou „*drag and drop*“, tj. stisknout levým tlačítkem myši nad červeným čtverečkem, potáhnout na požadovanou velikost a pustit. Výsledná velikost je poté, podle hustoty mřížky, přichycena k nejbližšímu většímu průsečíku mřížky, podobným způsobem jako při přesunu komponenty.

### 4.2.2 Implementace

#### *4.2.2.1 Vykreslení políčka pro změnu velikosti*

Komponenty, které se v editoru formulářů umísťují na jeho plochu, jsou definovány různými třídami, ale některé vlastnosti mají společné. Všechny jsou odvozeny od rodičovské třídy *JPanel* grafické knihovny *Swing*. Pro vykreslení červeného rožku, jako je na obrázku (Obr. 10), je nutné upravit metodu *paint* této rodičovské třídy. Úprava ve všech třídách všech druhů komponent je nevhodná, zejména kvůli opakujícímu se kódu. Při jakékoliv změně by musela být opět přeprogramována celá řada tříd, což se jistě negativně odrazí na času, který programátor při úpravách stráví, ale také se zvýší riziko možné chyby. Proto je vhodné vytvořit novou třídu, ze které budou komponenty formulářů dědit některé společné vlastnosti a metody. Z tohoto důvodu byla vytvořena třída *Painter*, jež obsahuje předefinovanou metodu *paint* a má za úkol vykreslit do komponenty v návrhovém zobrazení červený rožek, který umožní změnu rozměru daného prvku. Třídy jednotlivých komponent tedy nově dědí z třídy *Painter* a teprve ta dědí ze třídy *JPanel* viz obrázek (Obr. 12).




*Obr. 12 – Schématické umístění třídy Painter*

#### *4.2.2.2 Změna velikosti pomocí „drag and drop“*

Samotná změna velikosti probíhá v obslužných událostech myši. Konkrétně se jedná o události *mousePressed*, *mouseDragged* a *mouseReleased*. Při stlačení tlačítka myši je nejprve v metodě *mousePressedHandler* zjištěno, zda se kurzor nachází nad místem určeným pro změnu velikosti (červeně označené místo). Pokud ano, je tato informace zaznamenána v paměti současně s aktuálními souřadnicemi, které jsou později využívány při výpočtu nové velikosti. Při pohybu myši dochází ke změně souřadnic a průběžně je proto v metodě *mouseDraggedHandler* vypočítávána nová šířka a výška komponenty, která je následně ihned vykreslena.

#### *4.2.2.3 Přichycení k mřížce*

Po uvolnění tlačítka myši jsou pomocí původních a nových souřadnic spočteny rozměry, které jsou podle hustoty mřížky zaokrouhleny nahoru. Tím je zajištěno, že se komponenta přichytí k mřížce, což uživatelům velmi usnadní zarovnávání více prvků do stejného řádku nebo sloupce. Nová šířka a výška je nakonec uložena a celá komponenta je vykreslena do nové podoby. Tyto výpočty probíhají v obsluze události *mouseReleased*.

Při navrhování formulářů v editoru bylo zjištěno, že je někdy vhodné prvky k mřížce přichytit a někdy zase ne. Proto je funkce přichytávání prvků k mřížce zapnuta jen při zobrazení mřížky. A to jak při změně velikosti komponenty, tak i při jejím posunu.

## 4.3 Obsluha nových prvků formuláře

Editor formulářů programového systému NAHOS uměl pracovat s mnoha základními prvky, jako jsou například tlačítka, popisky, zaškrtávací políčka, tabulky a další. Nabízená paleta komponent však nestačila a bylo potřeba editor rozšířit o správu dalších prvků, které již v systému jsou, ale editor je nezná a tudíž ani neumí používat. Pokud chtěl uživatel vytvořit formulář, který by obsahoval prvky jako třeba *IconLabel* či *TabPanel*, musel je ručně zapsat v nějakém textovém nebo XML editoru, což je však velmi pracné a značně nepohodlné. Bylo tedy třeba tyto nové komponenty do editoru přidat, generovat výstup do XML souborů a naučit tyto soubory opětovně správně načítat.

### 4.3.1 IconLabel

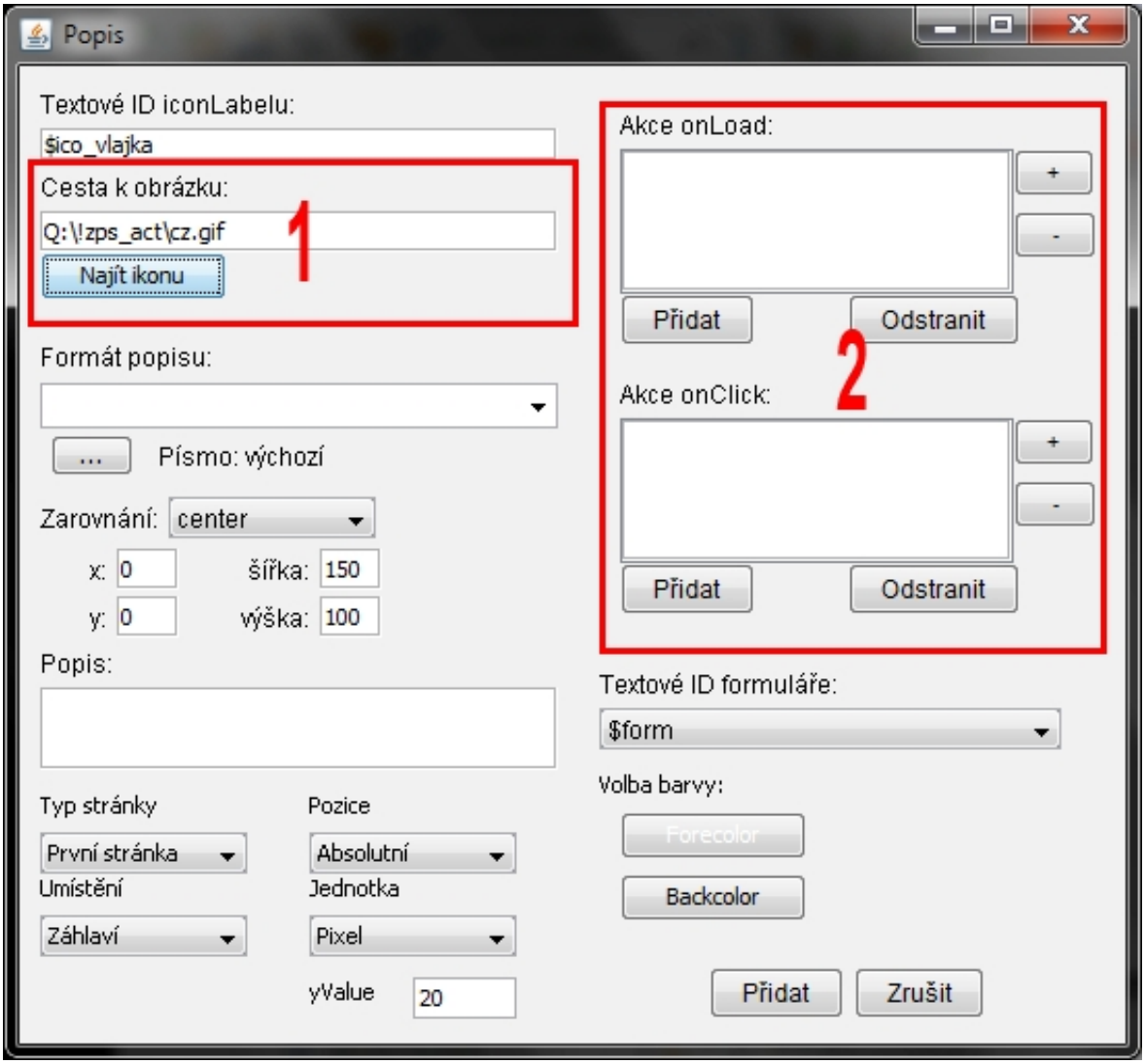
Prvním přidávaným prvkem je *IconLabel*. Jedná se o popisek, který má místo textu ikonu nebo jakýkoliv obrázek. Takový prvek se může hodit zejména při potřebě určitého grafického výstupu, ukázkového diagramu, nebo jen pro zlepšení vzhledu formulářů.

#### *4.3.1.1 Popis dialogového okna IconLabel*

Vybereme-li z menu prvků formuláře položku *IconLabel*, zobrazí se dialogové okno jako je na obrázku (Obr. 13). Okno je podobné jako při vytváření komponenty *Popis* (*Label*), obsahuje však oproti němu několik změn.

Chybí zde položka *text*, která sloužila pro zobrazení popisku. Místo ní je přidán nový údaj, *cesta k obrázku*, sloužící k popisu cesty ke zobrazovanému obrázku (označeno číslem 1). Jsou dvě možnosti jak cestu zvolit. Buďto ručním zápisem, nebo výběrem souboru pomocí tlačítka *Najít ikonu*. Obrázek lze zadat buď relativně, kdy je obrázek vybírán z kořenového adresáře programu, nebo absolutně, tj. i s případnou cestou. Výběr ikony pomocí tlačítka má oproti ručnímu zápisu několik výhod. První z nich je, že nalezená cesta bude platná a nehrozí ani překlepy jako při ručním zadávání cesty. Druhou výhodou je to, že po nalezení obrázku je automaticky změněna velikost celé komponenty přesně podle rozměrů obrázku. Například, má-li hledaný obrázek velikost 150x100 obrázkových bodů, bude po vložení do dialogového okna automaticky upravena šířka komponenty na 150 obrázkových bodů a výška na 100. Uživatel se tedy vůbec nemusí starat o velikost komponenty.

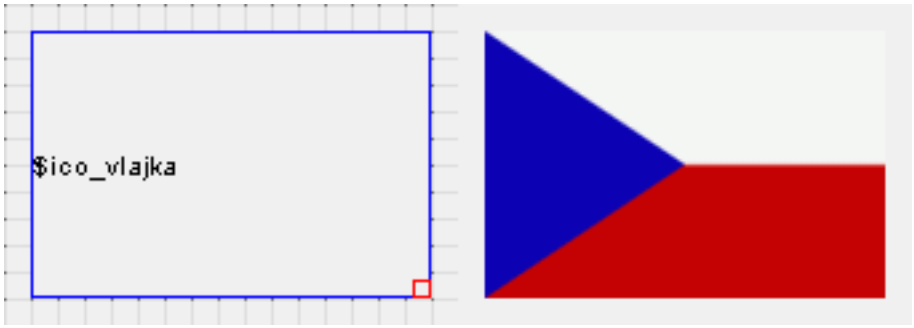
Další změnou jsou akce (na obrázku označeno číslem 2), kde přibyla možnost vkládat akce při kliknutí na obrázek. Ostatní možnosti této komponenty jsou vesměs shodné s možnostmi komponenty *Popis*. Po stisku tlačítka *Přidat* se prvek zobrazí ve formuláři a lze s ním manipulovat jako s jinými prvky.




*Obr. 13 – Dialogové okno IconLabel*

#### *4.3.1.2 Zobrazení komponenty IconLabel*

Na obrázku (Obr. 14) je zobrazen nově přidaný *IconLabel*. Obrázek je rozdělen na dvě části, vlevo je zobrazení návrhové, vpravo náhledové.




*Obr. 14 – Komponenta IconLabel zobrazená v editoru formulářů*

#### *4.3.1.3 Implementace IconLabel*

Základní třídou je *MyIconLabel*, která má obdobné vlastnosti i funkce jako třída *MyLabel* pro komponenty *Popis*. Rozdíl je v práci s obrázkem namísto textu a v přidání obsluhy akcí při kliknutí na ikonu. Třída komunikuje s dialogovým oknem definovaným ve třídě *AddIconLabelDilog*, které je zobrazeno na obrázku (Obr. 13).

**Hlavní funkce pro práci s obrázkem ve třídě MyIconLabel:**

- ***setIcon*** – Nastaví ikonu podle zadané cesty k souboru.
- ***getIconPath*** – Metoda vrací cestu k souboru (obrázku).

#### *4.3.1.4 Popis komponenty pomocí XML*

Při ukládání je vygenerován XML kód s informacemi, potřebnými pro uchování všech důležitých údajů o dané komponentě. Příklad takového kódu pro prvek *IconLabel* je následující:

```
<iconLabel>
  <itemTextID>$ico_vlajka</itemTextID>
  <bounds height="100" width="150" x="120" y="130"></bounds>
  <properties></properties>
  <iconPath>Q:\!zps_act\cz.gif</iconPath>
  <onLoad></onLoad>
  <onClick></onClick>
  <format>null</format>
  <align>center</align>
  <description>Popis tlačítka</description>
</iconLabel>
```

### 4.3.2 TabPanel

Dalším prvkem, který nebyl zakomponován do editoru formulářů je prvek *TabPanel*. Ve skutečnosti jde o dva druhy komponent, které společně dohromady tvoří jeden celek. Jde o komponenty *TabbedPane* a *TabbedItem*. *TabPanel* slouží ke zobrazení několika různých panelů překrytých za sebou, kdy je možno mít zobrazen vždy pouze jeden z nich (Obr. 16). Na těchto panelech můžou být umístěny prakticky libovolné ovládací prvky nezávisle na sobě a přepínání mezi nimi funguje pomocí záložek, umístěných přímo nad panelem. Hlavní význam spočívá v tom, že pokud má být na formuláři umístěno mnoho prvků, můžeme je řadit nejen vedle sebe, ale také nad sebou. Můžeme mít také zobrazenu jen určitou skupinu komponent, které mají v danou chvíli význam a jsou zrovna potřebné, čímž se mnohonásobně zvýší přehlednost jinak složitých formulářů.

Komponenta *TabbedPane* slouží jako základ pro *TabPanel* a kromě základních vlastností obsahuje také tzv. kontejner, do něhož se umísťují jednotlivé panely (*TabbedItem*).

#### *4.3.2.1 Návrh*

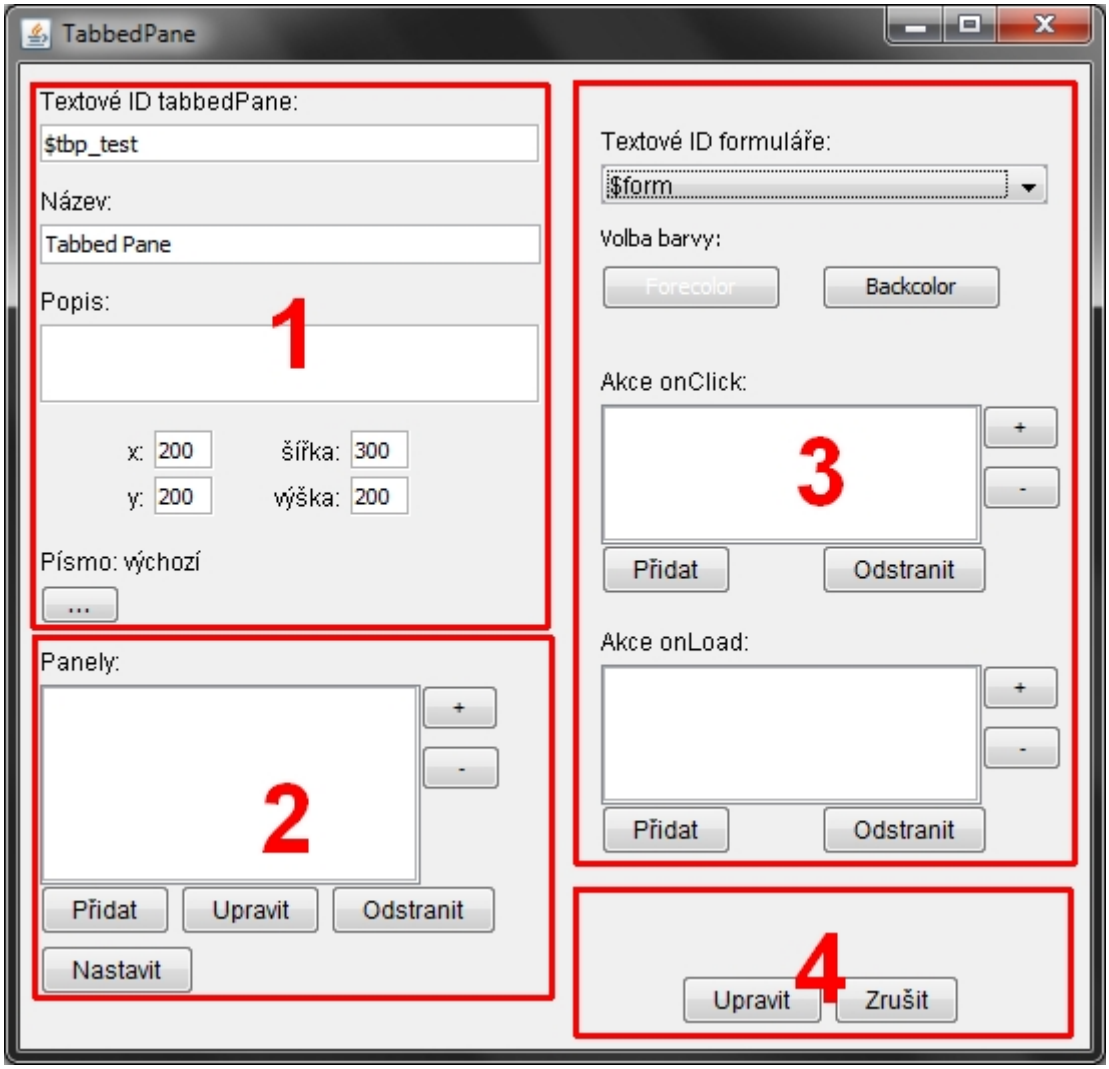
Největší úskalí při vytváření *TabPanelu* tkví v samotné podstatě komponenty, a tou je právě překrývání panelů. Doposud bylo možné ovládací prvky formuláře skládat pouze vedle sebe. Při umísťování prvků na *TabPanel* je potřeba dát aplikaci vhodným způsobem vědět, na který z panelů chceme prvek vložit.

Jednou z možností je vytvářet panely (*TabbedItem*) samostatně v nových oknech a jako hotové celky je umístit do *TabbedPane*. Takové řešení bylo zavrženo z toho důvodu, že by bylo potřeba zároveň pracovat s několika okny, což by editor formulářů činilo nepřehledným a složitým.

Jiným možným řešením je pracovat stále v jednom okně a mít možnost na nějakém vhodném místě zvolit aktuální panel, se kterým se pracuje. Vhodným místem může být například přepínání ve vlastnostech komponenty *TabbedPane*. Tento způsob řešení je uživatelsky přehlednější, což bylo hlavním důvodem k jeho realizaci.

#### *4.3.2.2 Popis dialogového okna TabbedPane*

Chceme-li na formulář vložit položku *TabbedPane*, vybereme ji z menu *Přidat*. Otevře se dialogové okno se základními vlastnostmi podobné, jako je na obrázku (Obr. 15). Jediné, co bude při prvním otevření tohoto okna chybět, jsou ovládací prvky pro výběr panelů (označeny číslem 2). Také místo tlačítka *Upravit* (označeno číslem 4) je zde tlačítko *Přidat*, pro vytvoření komponenty. Vládání panelů chybí z toho důvodu, že komponenta *TabbedPane* ještě nebyla vytvořena, a proto není kam panely umísťovat. Ovládací prvky označené jedničkou slouží k zadávání základních vlastností, jimiž jsou: ID, název, popis, poloha, rozměry a font. V části označené trojkou jsou pak doplňující vlastnosti a akce při načtení komponenty a při kliknutí na ni.



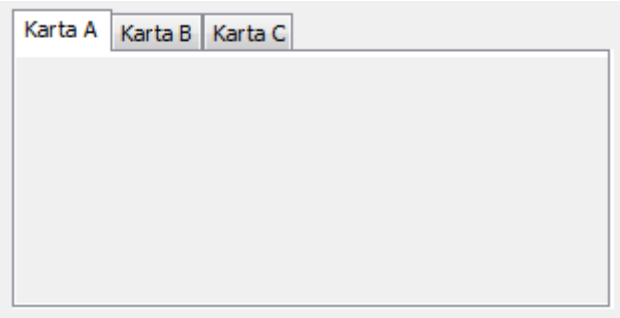

*Obr. 15 – Dialogové okno TabbedPane*

Po přidání komponenty je v návrhovém zobrazení vidět obrys, který je, narozdíl od jiných komponent, rozdělen na dvě části. Horní část ohraničuje plochu pro záložky a dolní část plochu pro vkládání komponent. V náhledu je prozatím vidět pouze bílá plocha bez jediné záložky. Pro vytváření záložek je třeba znovu otevřít vlastnosti prvku *TabbedPane*, kdy již uvidíme kompletní dialogové okno, stejné jako na obrázku (Obr. 15). Ve skupině ovládacích prvků označených dvojkou je 6 tlačítek a textová plocha pro seznam panelů.

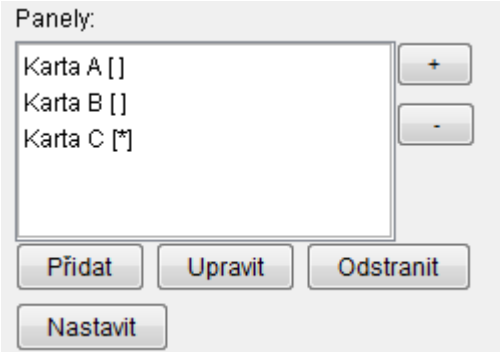
**Popis ovládacích prvků pro přidávání panelů:**

- **Seznam panelů:** V textovém seznamu na obrázku (Obr. 17) jsou přidány 3 panely. Každý panel je označen svým názvem a příznakem, zda je panel aktivní, nebo neaktivní.
  - **[*]** – Položka je aktivní
  - **[ ]** – Položka je neaktivní
- **Změna aktivní položky:** Poklepáním myší na položku v seznamu panelů lze změnit příznak, která položka bude aktivní. Vždy může být aktivní pouze jedna.
- **Tlačítko *„Přidat“*:** Vytvoření nového panelu pomocí dialogového okna *TabbedItem*. V seznamu se objeví nová položka se zadaným názvem a příznakem, že je panel aktivní.
- **Tlačítko *„Upravit“*:** Slouží ke zobrazení a úpravě vlastností položky vybrané ze seznamu panelů. Zobrazí se dialogové okno prvku *TabbedItem*. Pokud není žádný z panelů vybrán, vyskočí pouze informativní hláška.
- **Tlačítko *„Odstranit“*:** Označený panel (položka v seznamu) bude odstraněna. Pokud je odstraněn panel, který je právě aktivní, přesune se aktivita na první panel v seznamu.
- **Tlačítko *„Nastavit“*:** Po stisknutí tohoto tlačítka jsou aktivnímu panelu přiřazeny veškeré prvky, které se celou svou plochou nachází v dolní oblasti *TabbedPane*, určené pro vkládání komponent. Na obrázku (Obr. 18) leží 4 komponenty v oblasti *TabbedPane*. Při stisku tlačítka *Nastavit* budou aktivnímu panelu přiřazeny pouze prvky *A* a *B*, neboť *C* a *D* leží mimo určenou oblast, i když do ní částečně zasahují.

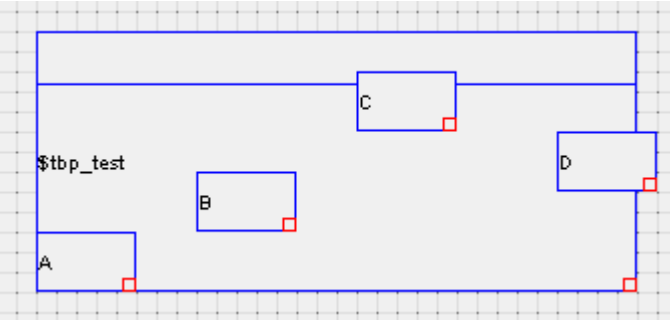
- **Tlačítka „ + “ a „ – “:** Tato dvě tlačítka slouží ke změně pořadí panelů.




*Obr. 16 – Záložky v TabPanelu*




*Obr. 17 – Správa panelů v dialogovém okně TabbedPane*

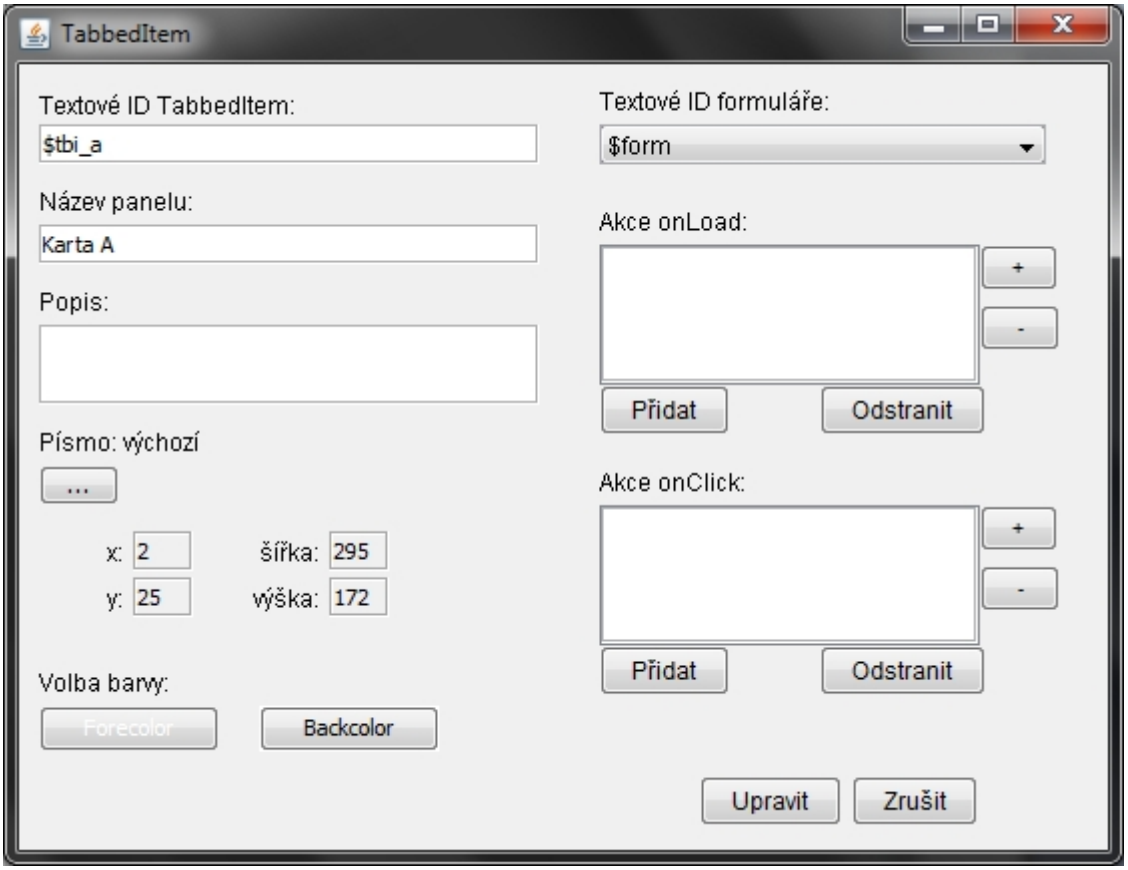



*Obr. 18 – Komponenty vkládané na TabPanel*

#### *4.3.2.3 Popis dialogového okna TabbedItem*

Přidáním panelu v dialogovém okně *TabbedPane* dojde o otevření nového dialogového okna *TabbedItem*. Na obrázku (Obr. 19) je vidět, že zde nejsou žádné nové vlastnosti oproti jiným komponentám. Jedinou změnou je, že nemůžeme ručně měnit rozměr

a polohu komponenty. To je ovšem logické, neboť každý panel (*TabbedItem*) má své umístění již pevně dané ve své nadřazené komponentě *TabbedPane*.




*Obr. 19 – Dialogové okno TabbedItem*

#### *4.3.2.4 Implementace TabbedPane*

Definice komponenty je uvedena ve třídě *MyTabbedPane*. Tato třída obsahuje kontejner typu *Vector*, který slouží pro uchovávání jednotlivých panelů, obsažených v *TabbedPane* a má implementováno několik funkcí pro práci s nimi.

**Funkce pro práci s *TabbedPane*:**

- ***addPanel*** – Přidává nový panel jako poslední položku v kontejneru a nastavuje ji jako aktivní.
- ***removePanel*** – Pokusí se odstranit panel ze seznamu panelů. Aby byl panel odstraněn, musí existovat a být v seznamu označen. Také je potřeba odsouhlasit smazání panelu v potvrzovacím dialogu, ve kterém je uživatel

upozorněn, že budou odstraněny také veškeré komponenty obsažené v příslušném panelu.

- ***getItems*** – Tato metoda vrací všechny panely jako *Vector*.
- ***panelCount*** – Vrací celkový počet panelů.
- ***movePanelToFront*** – Přesouvá panel o jedno místo dopředu, pokud je to možné.
- ***movePanelToBack*** – Přesouvá panel o jedno místo dozadu, pokud je to možné.
- ***setActiveItem*** – Metoda nastavuje, který panel bude aktivní. Vždy může být aktivní pouze jeden.
- ***getActiveItem*** – Metoda projde všechny panely, vyhledá ten, který je aktivní a vrátí jej.
- ***getItemAt*** – Metoda, která vrací panel na zadané pozici.

Tyto funkce spolupracují s třídou dialogového okna *AddTabbedPaneDialog*, která se stará o zobrazení vlastností komponenty a o zpracování obsluhy tlačítek a vstupů od uživatele. Dále je využívá třída *MyXMLDocument* při vytváření XML popisu komponenty nebo při jejím zpětném načtení do editoru formulářů.

Třída *MyTabbedPane* také přepisuje vykreslovací metodu *paint* ze zděděné třídy *Painter*, neboť navíc vykresluje linku, která v návrhovém zobrazení graficky odděluje záložky od panelů.

Další potřebnou funkcí v této třídě je implementace naslouchače události, která nastane při změně aktivního panelu. Taková událost je zachycena v metodě *ChangeListener*, která má obsluhu v konstruktoru třídy *MyTabbedPane*. Díky ní je možné se přepínat mezi aktivními panely i v náhledovém zobrazení, což usnadňuje a urychluje práci s celým *TabPanelem*.

#### *4.3.2.5 Implementace TabbedItem*

Komponenta *TabbedItem* je definována ve třídě *MyTabbedItem*. Tato třída, podobně jako třída *MyTabbedPane*, obsahuje kontejner typu *Vector*, který zde uchovává komponenty z editoru formulářů, které jsou k panelu přiřazeny.

**Funkce pro práci s MyTabbedItem:**

- ***getItems*** – Vrací seznam všech komponent přiřazených k panelu v podobě kontejneru typu *Vector*.
- ***itemsCount*** – Vrací počet komponent umístěných na panelu.
- ***removeItems*** – Odstraňuje všechny komponenty z panelu.
- ***hide*** – Skryje všechny komponenty umístěné na panelu. Používá se, když panel není aktivní. Se skrytými položkami je znemožněna jakákoliv manipulace, kromě přesunu celého *TabPanelu* se všemi panely a komponentami.
- ***show*** – Při aktivaci panelu jsou všechny komponenty, které jsou na něm umístěny znovu zobrazeny. Díky této metodě a metodě předchozí je vždy umožněna práce pouze nad jedním aktivním panelem.
- ***setActive*** – Nastavuje sám sebe jako aktivní panel. Volá metody *show* a *hide*.
- ***getActive*** – Vrací, zda je panel označen jako aktivní.

Třída pro zobrazení dialogového okna *AddTabbedItemDialog* se stará o vstupy uživatele a obsluhu tlačítek. Při vytváření panelu také spočítá polohu a rozměry, neboť ty uživatel nezadává.

Dále bylo potřeba upravit třídu *NewForm*, neboť formulář doposud neuměl pracovat se skrytými komponentami. Zejména se jednalo o úpravu událostí myši, aby nezachytávaly neviditelné komponenty.

Při kliknutí na prvek *TabbedPane* v návrhovém zobrazení jsou zároveň vybrány všechny komponenty ze všech panelů, včetně těch skrytých. Má to svůj význam při přesunu *TabPanelu*, jinak by totiž zůstávaly neviditelné komponenty na místě bez přesunu.

**Nové funkce ve třídě NewForm:**

- ***clearAllInvisible*** – Odstraní ze seznamu vybraných komponent neviditelné položky.
- ***getVisibleInRect*** – Vrací všechny viditelné komponenty, které jsou celé obsaženy v obdélníku, jenž metoda přebírá jako parametr.
- ***getAllInRect*** **–** Vrací všechny komponenty, které jsou celé obsaženy v obdélníku, jenž metoda přebírá jako parametr.

#### *4.3.2.6 Popis TabbedPane a TabbedItem pomocí XML*

Při ukládání je vygenerován XML kód s informacemi potřebnými pro uchování všech důležitých údajů o dané komponentě.

O vytvoření XML kódu se starají funkce *createTabbedPaneElement* a *createTabbedItemElement* ve třídě *MyXMLDocument* a o zpětnou konverzi z XML do editoru formulářů funkce *createTabbedPaneItem* a *createTabbedItemItem*.

Při vytváření XML dokumentu je nejprve vytvořen element <*tabbedPane*> a jeho parametry, a poté je zjištěno, kolik obsahuje panelů. Pro každý panel je vytvořen element <*tabbedItem*> s vlastními parametry včetně elementu <*items*>. Zde jsou poté v cyklu procházeny všechny komponenty příslušející panelu a jsou umístěny dovnitř uvedeného elementu <*items*>.

Příklad XML kódu pro prvky *TabbedPane* a *TabbedItem* je následující:

```
<tabbedPane> <!-- základní element TabPanelu -->
  <itemTextID>$tbp_ID</itemTextID>
  <bounds x="200" y="200" width="300" height="200"></bounds>
  <properties></properties>
  <text>Název</text>
  <onLoad></onLoad>
  <onClick></onClick>
  <description>Nějaký popis</description>
  <tabbedItem> <!-- panel -->
    <itemTextID>$tbi_ID</itemTextID>
    <bounds x="2" y="25" width="295" height="172"></bounds>
    <properties bgColor="ffffffff" fgColor="ff000000"></properties>
    <text>Karta A</text>
```

```
    <onLoad></onLoad>
    <onClick></onClick>
    <description>Popis panelu</description>
    <items>
      <!-- zde jsou umístěny komponenty obsažené v panelu -->
    </items>
  </tabbedItem>
  <tabbedItem> <!-- druhý panel -->
    <!-- zde jsou umístěny parametry a komponenty druhého panelu -->
  </tabbedItem>
</tabbedPane>
```

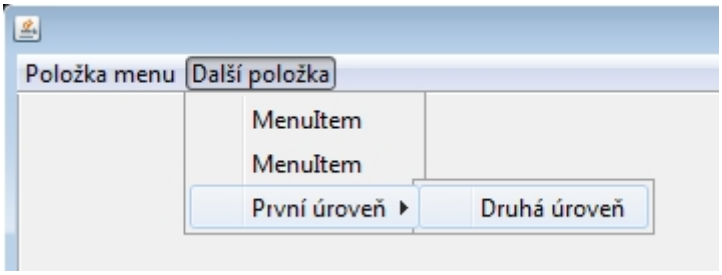
### 4.3.3 Menu

Možnost vytvářet menu byla již v editoru částečně implementována, její funkčnost však nebyla úplná a měla několik nedostatků. Menu sice vytvořit šlo, ale jednotlivé položky v menu se do sebe daly zanořovat v prakticky nekonečném cyklu. Hlavním požadavkem bylo omezit vrstvení položek na určitou úroveň a umožnit zobrazovat menu v náhledu editoru. Výsledný XML soubor také neuměl vytvářet všechny potřebné elementy.

#### *4.3.3.1 Popis menu*

Celkově je menu složeno ze tří samostatných částí (*MenuBar*, *Menu*, *MenuItem*), které si lze prohlédnout na obrázku (Obr. 20). Základní částí je komponenta *MenuBar*. Jedná se pouze o panel, umístěný v horní části formuláře, který obsahuje plochu pro umístění hlavních funkčních položek nabídky a nemá žádné textové popisky. Hlavními nabídkami, které se vkládají na *MenuBar*, jsou komponenty *Menu*. Každá z nich je označena jiným identifikátorem a názvem, který je zobrazen v *MenuBaru*. V případě ukázkového obrázku se jedná o *Položka menu* a *Další položka*. Při rozkliknutí hlavní nabídky jsou zobrazeny příslušné vedlejší nabídky (komponenty *MenuItem* nebo *Menu*), které mohou mít dvě úrovně. První úroveň je zobrazena v sloupci pod hlavní nabídkou a druhá úroveň je rozbalena do strany. Pokud se jedná o dvouúrovňovou položku, je první z úrovní vždy komponenta typu *Menu* a zobrazuje navíc kromě názvu i malou šipku, která uživatele informuje, že nabídka obsahuje další podnabídky. Na obrázku je to prvek *První*

*úroveň*. Prvky na druhé úrovni a jednoúrovňové položky jsou zase vždy komponenty typu *MenuItem*.



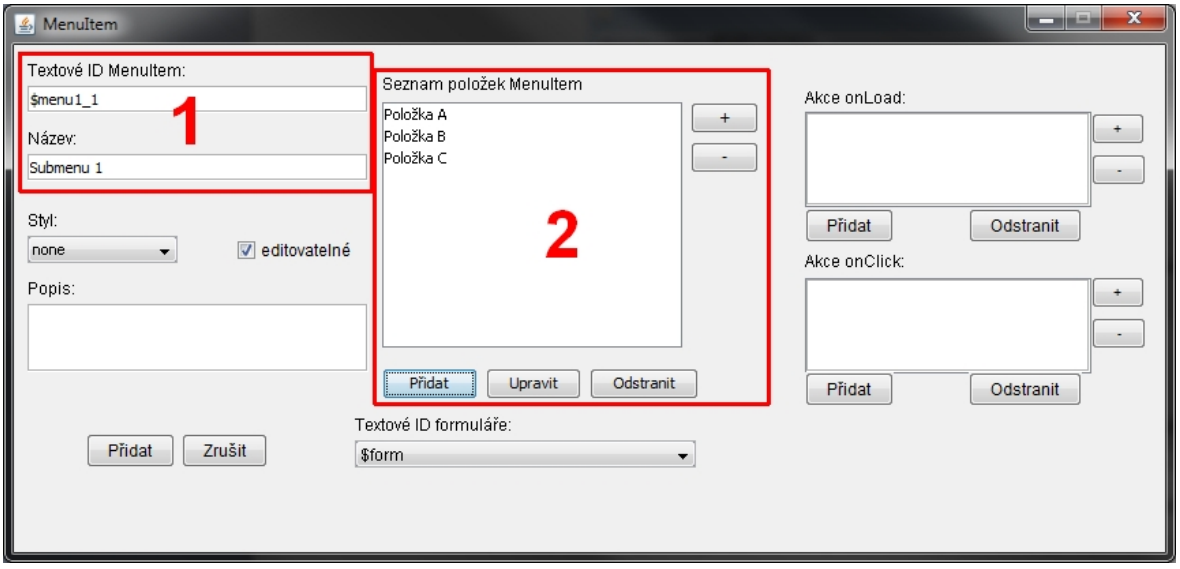

*Obr. 20 – Ukázka menu*

Každé položce v menu lze přiřadit určité akce, a tím vytvořit její obsluhu. Jedná se o akce *onClick* a *onLoad*.

#### *4.3.3.2 Implementace menu*

Přesto, že se menu skládá ze tří samostatných částí, v editoru formulářů je složena pouze ze dvou. Prvky *Menu* a *MenuItem* jsou součástí jediné třídy *MyMenuItem*, a o který z prvků se jedná je rozhodnuto až při běhu aplikace. Samotné rozhodování funguje na základě toho, zda se jedná prvek koncový (neobsahuje již další prvky), nebo prvek, obsahující ještě další prvky.

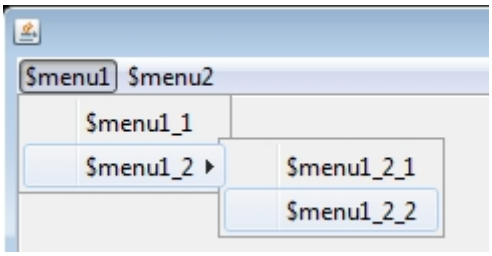
Kvůli omezení úrovní byla třídě *MyMenuItem* přidána proměnná *level*, která pro každou položku v menu určuje, na jaké úrovni se nachází. Při vkládání nových prvků menu je zobrazeno dialogové okno jako na obrázku (Obr. 21), ale pokud je přidávána položka na poslední možné úrovni, seznam položek (označeno číslem 2) v dialogovém okně chybí.




*Obr. 21 – Dialogové okno MenuItem*

Identifikátory pro položky v menu (na obrázku označeno číslem 1) jsou generovány automaticky v hierarchickém pořadí, uživatel je však může změnit. Číslování identifikátorů na různých úrovních je oddělováno podtržítkem. Následující hierarchie úrovní je shodná s menu na obrázku (Obr. 22):

- $menu1
  - $menu1_1
  - $menu1_2
    - $menu1_2_1
    - $menu1_2_2
- $menu2




*Obr. 22 – Hierarchie menu*

Samotný panel pro menu je vytvořen ve třídě *MyMenuBar*. Ta se mimo jiné stará i o vykreslení menu v editoru formulářů při náhledovém zobrazení. Původně v náhledovém zobrazení u menu chybělo, což bylo pro editaci nevýhodné. Uživatel neměl možnost v editoru zkontrolovat, jak menu vypadá a zda jsou všechny položky správně umístěny. Z toho důvodu bylo do editoru přidáno zobrazení vytvořeného menu. Vytvoření a vykreslení položek probíhá ve funkci *showItem* ve třídě *MyMenuBar*. Pomocí tří vnořených cyklů jsou procházeny veškeré položky (*Menu* a *Menuetem*), které jsou následně přidány na *MenuBar* a zobrazeny.

#### *4.3.3.3 Popis menu pomocí XML*

O vytvoření XML kódu se stará funkce *createMenuBarElement* ve třídě *MyXMLDocument*, o znovunačtení menu z XML souboru do editoru formulářů funkce *createMenuBarItem*.

Celá struktura menu, zapsaná pomocí XML, je obsažena v elementu <*menuBar*>, který má ve svém těle mimo jiné obsaženy elementy <*menu*>. Ty v sobě zase mohou mít vnořovány elementy <*menuItem*>. Tímto způsobem vzniká stromová struktura menu.

Příklad XML kódu pro menu:

```
<menubar>
  <itemTextID> MENUBAR </itemTextID>
  <bounds x="0" y="0" width="1004" height="20"/>
  <description/>
  <menu>
    <itemTextID> $menu1 </itemTextID>
    <text> Soubor </text>
    <onLoad/>
    <onClick/>
    <menuitem type="none"> <!-- první úroveň -->
      <description/>
      <itemTextID> $menu1_1 </itemTextID>
      <text> Otevřít </text>
      <onLoad/>
      <onClick/>
      <menuitem type="none"> <!-- druhá úroveň -->
        <description/>
        <itemTextID> $menu1_1_1 </itemTextID>
```

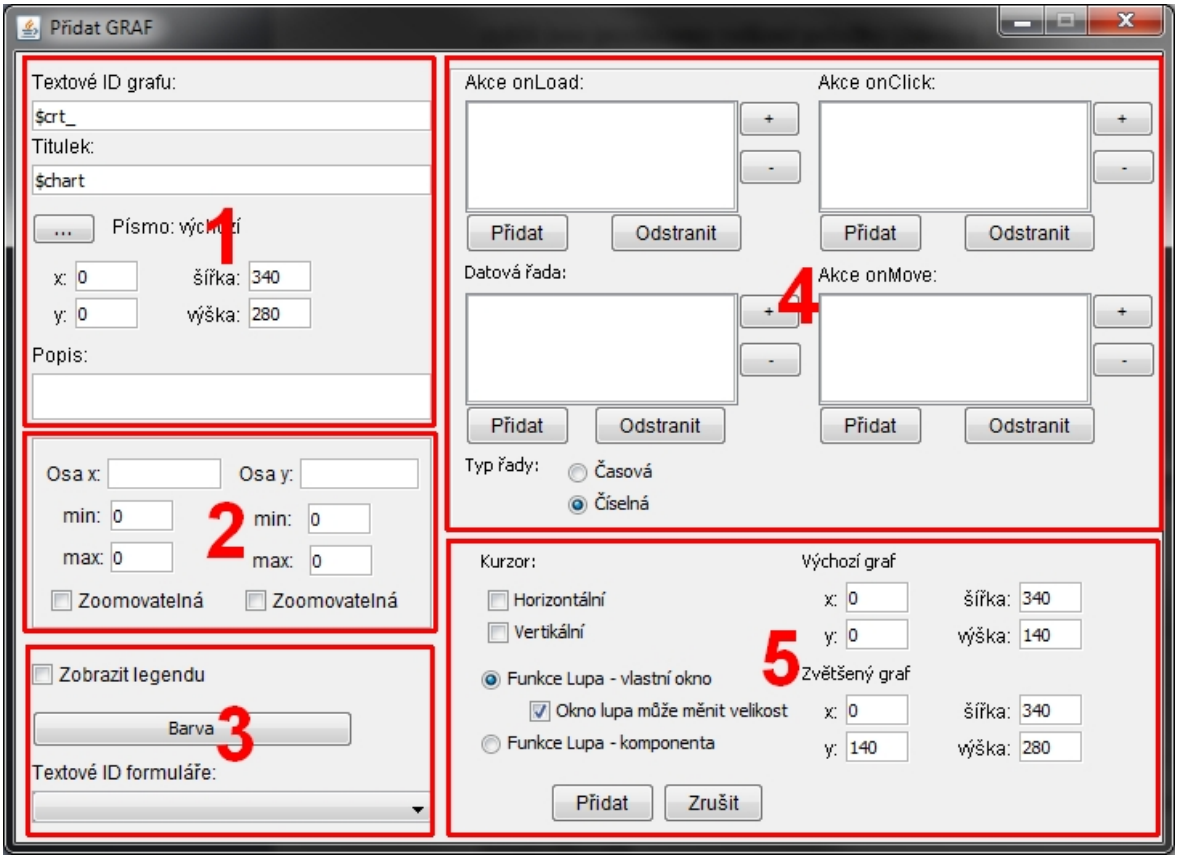
```
        <text> Naposled otevřený </text>
        <onLoad/>
        <onClick/>
      </menuitem>
    </menuitem>
  </menu>
</menubar>
```

### 4.3.4 Chart

Prvek *Chart* slouží ke zobrazování grafů. Jedná se o poměrně rozsáhlou komponentu, která již v editoru zahrnutá byla, jednalo se ale o starší verzi, která neumožňovala veškeré potřebné editování. Popis pomocí XML byl pro novou verzi razantně změněn, přibyly nové funkce a stávající verze, obsažená v editoru, již nebyla kompatibilní s verzí novou.

#### *4.3.4.1 Popis dialogového okna*

Komponenta *Chart* je přidávána pomocí dialogového okna ve třídě *AddChartDialog*, které je zobrazeno na obrázku (Obr. 23). Ovládacích prvků je zde hodně, proto je popis rozdělen na několik samostatných částí.




*Obr. 23 – Dialogové okno Chart*

1. V první části obrázku (označena jedničkou) jsou základní parametry komponenty. ID grafu, název, font, poloha, rozměry a popis.
2. V části druhé je popis os X a Y, který slouží k zadání názvu os, například (Cena, čas…). Pod popiskem každé osy jsou rozsahy, kterých může daná osa nabývat. Zatržením políčka *Zoomovatelná* umožníme měnit rozsah osy v grafu.
3. V části označené trojkou je políčko *Zobrazit legendu*, kterým můžeme vypnout, či zapnout zobrazení legendy a tlačítkem *Barva* zvolíme pozadí grafu. Textové ID formuláře udává, ke kterému formuláři je graf přiřazen.
4. Čtvrtá část obsahuje seznamy akcí (*OnLoad*, *OnClick* a *OnMove*). Také je zde umístěn seznam pro datové řady, jejichž popis je uveden níže a výběr typu řady, který udává, zda se jedná o data číselná nebo časová.
5. V poslední, páté části, je volba kurzoru. Při pohybu myší nad grafem bude na pozici kurzoru zobrazena vertikální nebo horizontální osa. Můžou být zobrazeny také obě

osy či žádná. Při zvětšování máme možnost zobrazení ve stejném okně, nebo v okně novém pomocí lupy. Pro zobrazení ve stejném okně je možné nastavit polohu a rozměry lupy.

#### *4.3.4.2 Popis prvku Chart pomocí XML*

Ve třídě *MyXMLDocument* je funkce *createChartElement*, která se stará o vytvoření XML kódu, a také funkce *createChartItem*, sloužící pro zpětnou konverzi z XML do editoru formulářů. Základním elementem pro grafy je element <*chart*>.

Ukázka XML kódu pro prvek *Chart*:

```
<chart>
  <itemTextID> $crt_1 </itemTextID>
  <bounds x="50" y="30" width="500" height="400"/>
  <text> GRAF </text>
  <data type="number">  <!-- datové (číselné, časové) řady -->
    <dataserie showpoints="false" penwidth="1" bgColor="FFFF0000"
showlegend="true"> Řada 1 </dataserie>
  </data>
  <properties showlegend="true" bgColor="FFC0C0C0"><!-- vlastnosti -->
    <cursorType horizontaltrace="true" verticaltrace="false"/>
  </properties>
  <zoom type="COMPONENT" sizeable="false"> <!-- zoomování -->
    <bounds1 x="0" y="0" width="600" height="200"/>
    <bounds2 x="0" y="140" width="600" height="200"/>
  </zoom>
  <axis> <!-- osy (X a Y) -->
    <xAxis text="XXX" x="0" y="2000" zoomable="true"/>
    <yAxis text="YYY" x="10" y="50" zoomable="false"/>
  </axis>
  <font face="Arial" size="12" style="plain"/>
  <onLoad/>
  <onClick/>
  <onMove/>
  <align> left </align>
  <description> Popis </description>
</chart>
```

### 4.3.5 GraphicArea

Komponenta *GraphicArea* je na rozdíl od předchozích komponent (*ImageIcon*, *TabPanel*, *Menu*, *Chart*) zcela nová a v editoru, ani v celé aplikaci ještě nebyla zavedena. Hlavní motivací k vytvoření nového prvku bylo umožnit v editoru formulářů vytvářet různá schémata, která by se zobrazovala jako samostatný grafický prvek.

#### *4.3.5.1 Návrh komponenty GraphicArea*

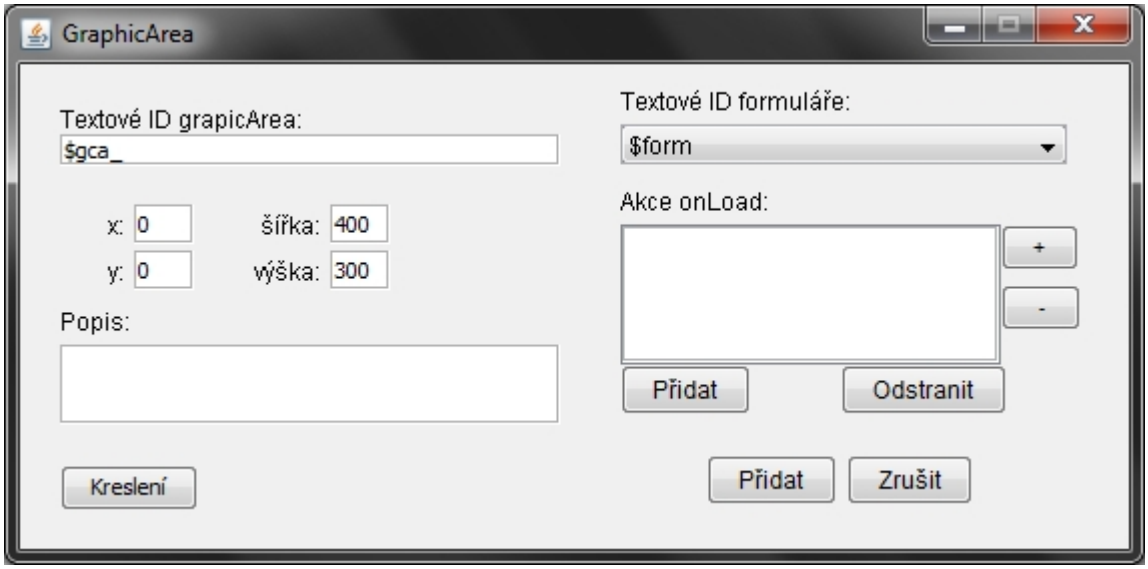
Protože struktura nového prvku nebyla jasně dána, bylo potřeba ji nejprve navrhnout. Pro vytváření schémat je vhodná práce s grafickými objekty, jako je čára, obdélník, elipsa a další. S takovými objekty by mělo být umožněno libovolně manipulovat, vytvářet nové a mazat nepotřebné. Výsledkem by pak byl obraz vytvořený vykreslením všech objektů na nějakém plátně, přičemž by právě ono plátno bylo komponentou, přidávanou na formulář. Umožnění práce s objekty může být realizována pomocí grafického editoru, který by se stal součástí editoru formulářů.

#### *4.3.5.2 Popis dialogového okna GraphicArea a grafického editoru*

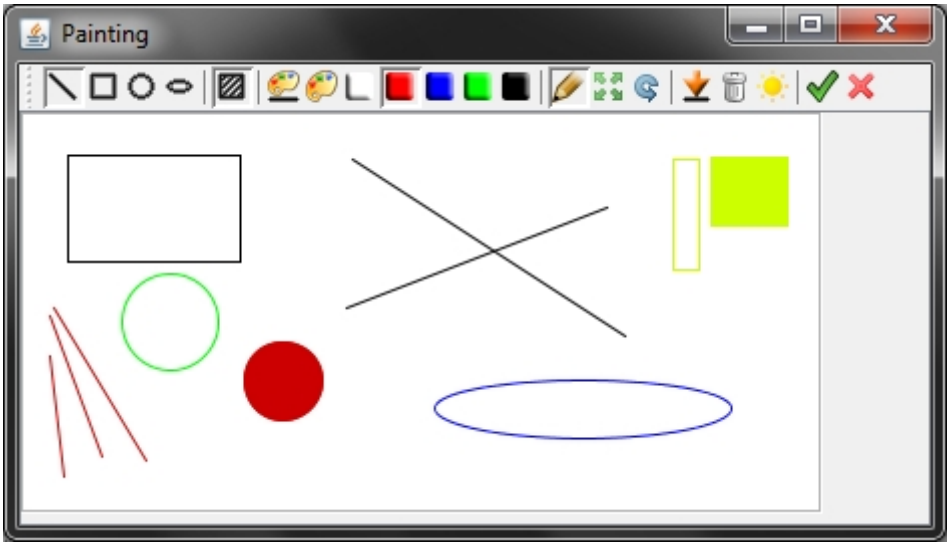
Vytvořením nového prvku *GraphicArea*, je otevřeno dialogové okno jako na obrázku (Obr. 24). Zde můžeme zvolit ID komponenty, polohu a rozměry, popis, případně i akce při načtení. Důležitým ovládacím prvkem je tlačítko „*Kreslení*“, kterým se spouští grafický editor (Obr. 25).

V současné době obsahuje editor pouze základní grafické objekty pro vykreslování a základní operace pro manipulaci s nimi. Rozšíření editoru bude vhodné až po specifikaci potřeb uživatelů a může být náplní dalších diplomových prací.

V horní části editoru jsou ovládací tlačítka, ve spodní části je plátno, na kterém jsou objekty vykreslovány. Plátno má rozměry pevně dané podle zadání v dialogovém okně komponenty *GraphicArea* a popis tlačítek a funkcí grafického editoru je uveden v příloze diplomové práce.




*Obr. 24 – Dialogové okno GraphicArea*




*Obr. 25 – Grafický editor*

#### *4.3.5.3 Implementace komponenty GraphicArea*

Základní třídou, popisující komponentu, je třída *MyGraphicArea*. Obsahuje především plátno pro zobrazení komponenty, barvu na pozadí a kontejner typu *Vector*, který slouží pro uchovávání vykreslovaných objektů. Se základní třídou pracuje třída dialogového okna *AddGraphicAreaDialog*, která se stará o vstupy uživatele, spouštění grafického editoru a zprostředkovává přenos vytvořených objektů mezi základní třídou a grafickým editorem.

Grafické objekty a funkce, jsou definovány ve třídě *GraphicElement*. Jedná se o abstraktní třídu, od níž jsou odvozeny vnitřní třídy *Line*, *Rect*, *Circle* a *Ellipse*. Nad každým vytvořeným objektem je definována sada funkcí pro manipulaci s nimi.

- ***move*** – přesun objektu na určenou pozici
- ***rotate*** – otočení objektu podle počátečního bodu o určený úhel (nemá význam pro objekt typu *Circle*)
- ***fill*** – vykreslovaný objekt bude vyplněn barvou (nemá význam pro objekt typu *Line*)
- ***draw*** – vykreslení objektu
- ***setHighlighted*** – nastaví zvýraznění objektu při manipulaci s ním

Další funkce slouží především pro vnitřní účely třídy, nebo pro získávání a nastavování informací o objektu.

Grafický editor je vytvořen pomocí dvou tříd (*PaintingFrame* a *Screen*). První z nich, *PaintingFrame*, se stará o vytvoření a vykreslení okna grafického editoru, plátna a tlačítek, ale také o obsluhu ovládacích prvků. Třída *Screen* je samotné plátno, ve kterém jsou objekty vykreslovány. Zpracovává události myši nad plátnem, díky nimž jsou objekty vytvářeny i mazány a je s nimi možná manipulace pomocí myši.

#### *4.3.5.4 Popis komponenty GraphicArea pomocí XML*

Při vytváření XML souboru je potřeba uložit nejen samotnou komponentu, ale také všechny grafické objekty v komponentě obsažené. Při procházení objektů je vždy vytvořen příslušný element podle typu objektu, a ten je vkládán do elementu <*items*>. Elementy jsou podle typu objektu děleny na <*line*> pro čáru, <*rect*> pro obdélník, <*circle*> pro kruh a <*ellipse*> pro elipsu. Každý z těchto elementů má definovány atributy, pomocí nichž je přesně definován celý objekt. Atributy *x* a *y* udávají polohu počátečního bodu (u kruhu střed), *x2* a *y2* polohu koncového bodu (u kruhu libovolný bod na obvodu). Barva objektu je dána pomocí barevných složek modelu *RGB* (červená = *r*, zelená = *g*, modrá = *b*). Atribut *fill* udává, zda je objekt vyplněn barvou, nebo ne (u čáry nemá význam) a atribut *angle* udává o kolik stupňů je objekt otočen (u kruhu nemá význam).

Ukázka XML kódu pro prvek *GraphicArea*:

```
<graphicArea>
  <itemTextID> $gca_1 </itemTextID>
  <bounds x="0" y="0" width="400" height="300"/>
  <onLoad/>
  <description/>
  <background r="200" g="230" b="180"/> <!-- barva na pozadí -->
  <items>
    <line x="22" y="30" x2="88" y2="105" r="0" g="0" b="0" fill="false"
angle="0.0"/>
    <rect x="53" y="209" x2="103" y2="254" r="0" g="0" b="0" fill="false"
angle="38.75"/>
    <circle x="144" y="171" x2="164" y2="181" r="255" g="0" b="0"
fill="true" angle="0.0"/>
    <ellipse x="229" y="142" x2="270" y2="252" r="204" g="204" b="0"
fill="false" angle="42.5"/>
  </items>
</graphicArea>
```

## 4.4 Správa komentářů v generovaném XML dokumentu

Editor formulářů měl velkou nevýhodu v tom, že neuměl číst XML dokumenty, obsahující komentáře. Takové dokumenty editor zkrátka přeskočil a nenačetl. Bylo tedy potřeba naučit editor načítat formuláře z XML souborů i s jeho případnými komentáři a ty následně znovu uložit do XML souboru.

### 4.4.1 Návrh správy komentářů

Vytvořit návrh pro uchovávání komentářů v XML souboru bylo oproti ostatním bodům zadání znatelně složitější. Bylo potřeba brát v potaz nejen chování počítačů a programového jazyka samotného, ale také chování uživatele při editaci XML souborů.

Editor formulářů byl navržen tak, že z existujícího souboru načítá veškeré komponenty i s jejich vlastnostmi a zobrazuje je. Komentáře ale nejsou vizuálními prvky ani jejich vlastnostmi, proto je nelze uchovávat stejným způsobem jako ostatní prvky. Zpětné uložení probíhá vygenerováním nového souboru z prvků a vlastností otevřeného formuláře. Komentáře přitom byly zcela ignorovány a bylo třeba nalézt způsob, jak poznámky uživatelů zachovávat.

#### *4.4.1.1 Problém „lidský faktor vs. počítač“*

Pravděpodobně největším problémem pro návrh i realizaci bylo, že počítačový program nemusí být vždy schopen správně interpretovat požadavky svých uživatelů. Jako příklad uvedu situaci, ke které může dojít:

```
<element1> Uvnitř prvního elementu </element1>
<!-- komentář -->
<element2> Uvnitř druhého elementu </element2>
```

V uvedeném XML kódů jsou dva elementy, které jsou na sobě zcela nezávislé. Mezi nimi je uveden krátký jednořádkový komentář, který uživatel vložil do kódu pomocí externího XML editoru. Problém nastane tehdy, chceme-li mezi uvedené dva elementy vložit třetí, nebo chceme-li jeden z elementů odstranit. Nabízí se několik otázek, na které se nedá jednoznačně a definitivně odpovědět. Co s komentářem? Patří k prvnímu elementu, nebo k druhému? Také nemusí patřit k žádnému z nich a může nést pouze nějakou všeobecnou informaci. Jak to tedy uživatel myslel, už bohužel nijak nezjistíme.

Přesné řešení výše uvedeného problému není, kvůli neznalosti uživatelova pohledu na věc, vůbec možné. Proto je třeba problém vodným způsobem obejít a pokusit se minimalizovat ztrátu komentářů či jejich špatnému umístění.

#### *4.4.1.2 Návrh pomocí identifikátorů*

Jednou z možností jak se nejednoznačnostem v umístění komentářů částečně vyhnout je připisovat k nim nějakou značku nebo identifikátor, díky nimž by bylo definitivně určené umístění daného komentáře. Každý komentář by tak dokázal zjistit svou pozici v dokumentu. Nevýhoda tohoto řešení je zřejmá. Uživatelé by museli znát postupy pro vkládání identifikátorů, a ty by musely být psány přesně, což není žádoucí.

#### *4.4.1.3 Návrh pomocí porovnání souborů a hledání nejvhodnějšího umístění*

Další možností, která však po uživatelích nevyžaduje žádné aktivity navíc, je minimalizace problému a vyhledávání nejlepšího možného řešení porovnáním původního a nově vytvořeného souboru. I tento způsob má své nevýhody. Jsou-li úpravy ve formuláři příliš drastické, přestává se původní soubor novému podobat a nalezení vhodného umístění

komentářů je pak obtížnější. Tato metoda je sice složitější, ale byla vybrána z toho důvodu, že neklade prakticky žádné nároky na uživatele.

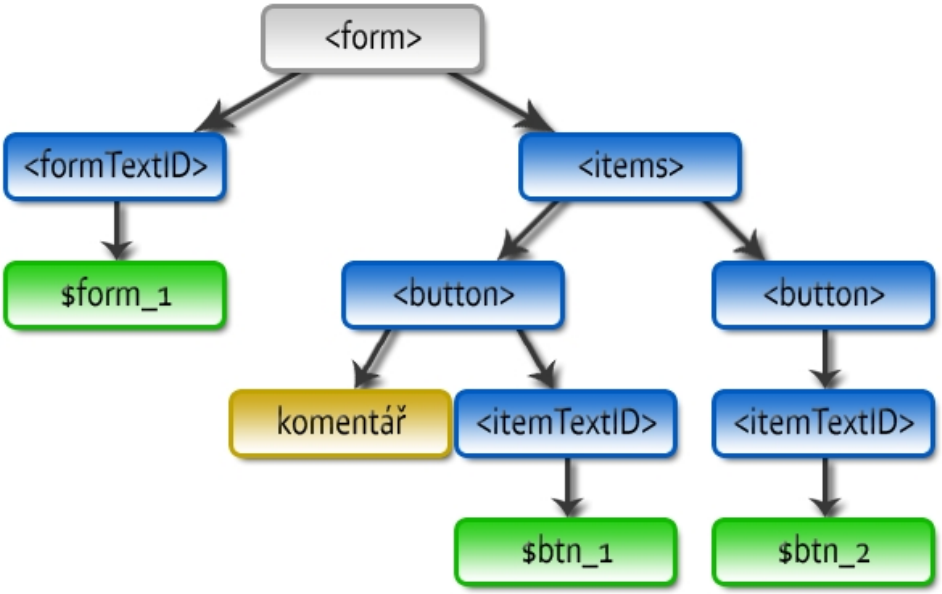
### 4.4.2 Implementace správy komentářů

#### *4.4.2.1 Stromová struktura dokumentů*

Struktura jakýchkoliv XML souborů se dá popisovat pomocí tzv. stromů, ve kterých jsou jednotlivé elementy, texty, ale i komentáře ukládány jako uzly. Prvním uzlem, nazývaným kořen je první prvek XML souboru. Další uzly jsou hierarchicky skládány jako větve stromu. Každý uzel, má svého rodiče (vždy pouze jednoho), pouze kořenový uzel rodiče nemá. Uzly mohou mít potomky, kterých může být libovolný počet. Pokud uzel nemá žádného potomka, říkáme, že se jedná o list stromu.

Z následujícího XML kódu lze sestavit strom jako je na obrázku (Obr. 26). Každý prvek stromu (uzel) má své pevné místo a lze o něm zjistit například kdo je jeho rodičem nebo kolik má potomků a jaké. Ve stromu jsou také uchovávány informace o typu uzlu, zda se jedná o element, text, komentář či něco jiného (na obrázku jsou barevně odlišeny). Stejně tak je barevně odlišen i kořenový uzel <*form*>.

```
<form>
  <formTextID>$form_1</formTextID>
  <items>
    <button>
      <!-- komentář -->
      <itemTextID>$btn_1</itemTextID>
    </button>
    <button>
      <itemTextID>$btn_2</itemTextID>
    </button>
  </items>
</form>
```




*Obr. 26 – Stromové schéma XML souboru*

#### *4.4.2.2 Uchování původního dokumentu*

Při otevírání dříve uloženého formuláře je vytvořena stromová struktura popisující umístění všech prvků v dokumentu. Díky struktuře je možné uchovat původní soubor v paměti pro potřebu porovnání s nově vygenerovaným souborem a zjistit tak umístění jednotlivých komentářů. Strom je uchováván ve třídě *NewForm*, která popisuje celý načtený formulář a také realizuje změny v něm, jako jsou například přidávání a odebírání prvků. Původní strom zůstává beze změny až do chvíle ukládání formuláře, kdy je používán pro výběr komentářů a jejich správnému umístění do nově vytvořeného souboru.

#### *4.4.2.3 Ukládání nového dokumentu*

Při ukládání je nejprve z prvků ve formuláři sestavena nová stromová struktura a poté jsou z původního stromu vybrány všechny uzly, které jsou komentářem. Pro každý nalezený komentář je pomocí vytvořené funkce *putIntoBestPosition* nalezeno umístění v novém stromu a komentář je následně uložen jako nový uzel. Mohou nastat situace, kdy komentář již nemá existovat, například pokud byl smazán prvek s tímto komentářem. V takových případech funkce nemá kam komentář umístit a vrací pouze informaci o tom, že nebyl uložen.

#### *4.4.2.4 Zjištění nejvhodnějšího umístění komentáře*

Každý komentář ve stromu má svého nejbližšího předchůdce. Vždy se jedná buď o rodiče, anebo sourozence, což poznáme podle toho, zda je předchozí prvek na stejné úrovni nebo vyšší. Například:

```
<button>
  <!-- Tento komentář má přímého předchůdce rodiče „<button>“ -->
  <itemTextID>
    $btn_1
    <!-- Tento komentář má přímého předchůdce sourozence „$btn_1“,
         „<itemTextID>“ je rodičem, ale není přímým předchůdcem -->
  </itemTextID>
</button>
```

Pro každý komentář se pokusíme nalézt prvek s odpovídajícím předchůdcem. Problém je, že takových umístění může být více. Pokud například zjistíme, že má být komentář hned za elementem <*button*> a takových elementů je v dokumentu více, musíme pokračovat v hledání.

Každý prvek formuláře je označen jednoznačným identifikátorem (element udávající ID, např. <*itemTextID*>). Díky této informaci dokážeme určit která z větví stromu je ta správná. Pomocí kombinace hledání identifikátoru a přímého předchůdce zjistíme počet možných umístění komentáře. Můžou nastat tři případy:

1. nalezneme právě jedno řešení
2. nenalezneme žádné řešení
3. nalezneme více než jedno řešení

První případ je ideální. Je to stav, kdy se podařilo přesně identifikovat místo, kde byl původně komentář umístěn, a proto jej můžeme vložit na své místo.

Druhý případ může nastat ze dvou důvodů. Buď byl prvek s komentářem zrušen, nebo se komentář nachází uvnitř nějakého textu. Je proto potřeba projít všechny textové elementy v identifikované větvi, pokusit se v nich nalézt shodnou část textu, za kterou komentář patří a vložit jej tam.

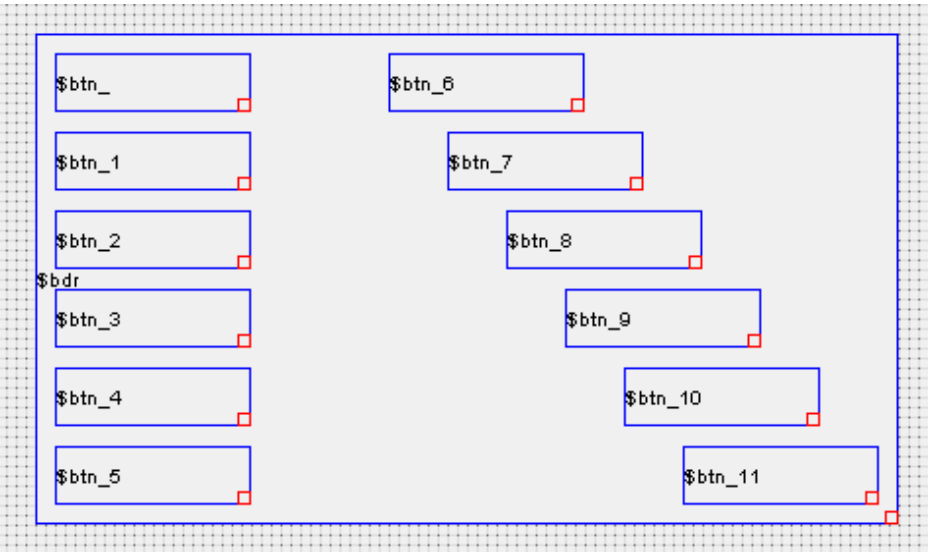
Pokud je však nalezeno více možných umístění i v rámci jednoho prvku formuláře, je potřeba použít další metodu pro nalezení správného místa. Metoda spočívá v postupném procházení uzlů pozpátku a srovnáváním těchto uzlů s původním dokumentem. Počet shodných předchůdců je ukládán pro každé z možných umístění, a je tak zjišťována vhodnost jednotlivých řešení. Jako finální řešení je vybrán uzel s největší vhodností a za něj je komentář umístěn.

## 4.5 Hromadné označení komponent myší

Jednou z věcí, která by urychlila a zpříjemnila práci s editorem formulářů je mít možnost myší označit více prvků najednou. Jedinou možností jak označit více prvků bylo doposud klikání myší na komponenty se současným stiskem klávesy *Ctrl*.

### 4.5.1 Návrh

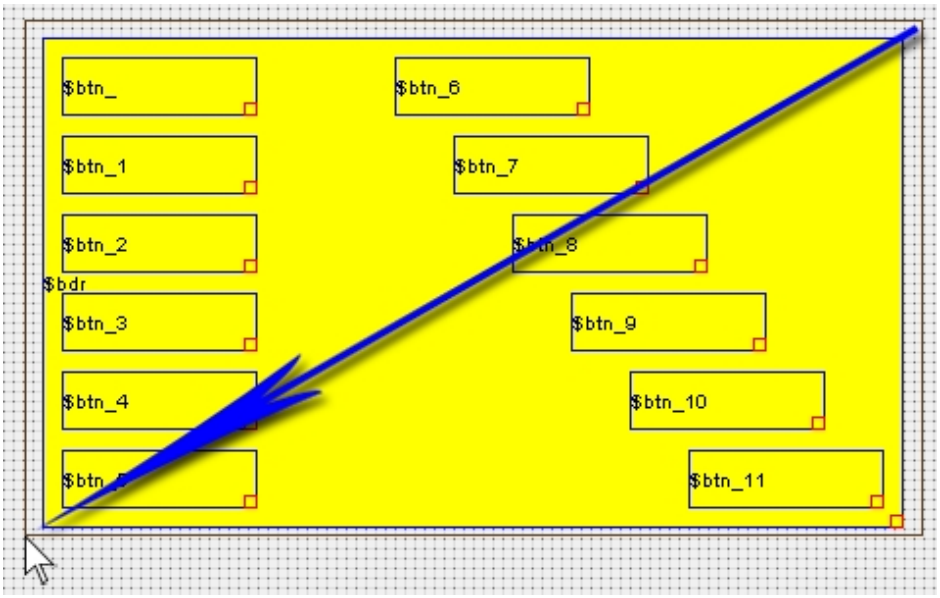
Na obrázku (Obr. 27) je na formuláři vloženo dvanáct tlačítek a okolo nich jeden rámeček. Pokud bychom chtěli všechny tyto prvky vybrat, museli bychom na každý prvek kliknout zvlášť a přitom držet klávesu *Ctrl*.




*Obr. 27 – Velké množství komponent*

Takový způsob je pomalý a pracný. Mnohem lepším řešením by bylo kliknout někam do rohu mimo prvky a s přidrženým tlačítkem posunut myší do protějšího rohu od prvků.

Tím by vznikl výběrový obdélník a všechny prvky, které by se nacházely uvnitř tohoto výběru, by byly označeny najednou, tak jak ukazuje šipka na obrázku (Obr. 28).



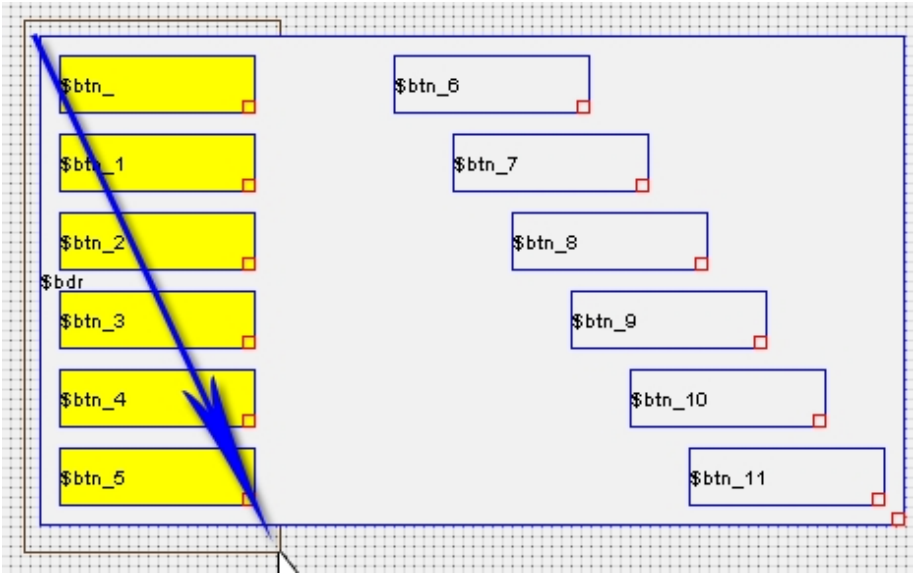

*Obr. 28 – Hromadné označení komponent myší*

Současně by mělo být možné prvky k tomuto výběru přidávat a ubírat pomocí již zavedeného klikání s klávesou *Ctrl*, nebo dalším obdélníkovým výběrem.

### 4.5.2 Implementace

Pro vytvoření výběrového obdélníku je potřeba vytvořit obsluhu pro události myši ve třídě *FormBody*. Při stisknutí tlačítka myši (*MousePressed*) je uchována pozice, na které ke stisku došlo, při uvolnění tlačítka (*MouseReleased*) je za pomocí uložené pozice a nové pozice kurzoru vytvořen obdélník pro výběr komponent. Mezi těmito událostmi může dojít ještě k jedné události – tažení myší (*MouseDragged*). V této události je také vypočítán obdélník, který slouží pouze k vykreslení výběrového obdélníku. Ten je vykreslen v metodě *paint*.

Samotný výběr komponent probíhá až po uvolnění tlačítka, kdy je zavolána metoda *selectItemsInRect* ze třídy *NewForm*. Ta projde všechny nevybrané prvky formuláře a podle jejich pozice rozhodne, zda bude prvek přidán, nebo ne. Aby byl přidán, musí celý ležet uvnitř výběrového obdélníku. Leží-li prvky ve výběru pouze částečně, nejsou vybrány. Tím je umožněno vybírat pouze určité prvky (Obr. 29).




*Obr. 29 – Hromadné označení komponent myší*

## 4.6 Změna pořadí komponent formuláře

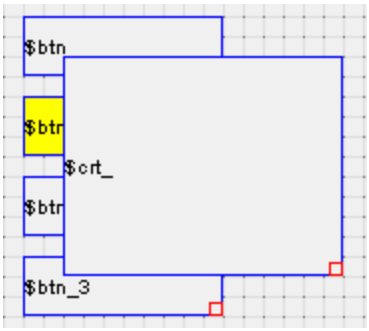
Při vytváření formulářů může občas docházet k situaci, že jeden prvek překryje druhý – menší prvek, a ten zůstane naspodu, kde se k němu není možné dostat. Jedinou možností je pak horní prvek posunout. Pokud je takových překrytých prvků víc, pak je již manipulace s nimi značně obtížná a zdlouhavá. Proto je vhodné mít k dispozici nějaký nástroj k posouvání prvků do popředí a do pozadí.

### 4.6.1 Návrh

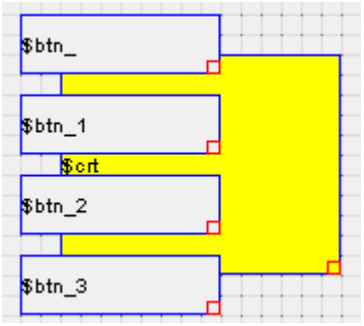
Na obrázku (Obr. 30) je vidět, jak větší komponenta částečně překrývá menší komponenty. Je to z toho důvodu, že byla do formuláře vložena jako poslední. Vhodný způsob jak měnit pořadí komponent je posouvat je pomocí klávesových zkratek. Kdy se označené prvky budou moci posouvat oběma směry, do popředí i do pozadí. Mělo by být možné posouvat je s každým stisknutím pouze o jednu pozici dopředu i dozadu, ale také na úplně první či poslední místo. Navíc by bylo dobré, kdyby se dalo manipulovat s více označenými prvky najednou. Na obrázku (Obr. 31) byl horní prvek posunut z horního místa na místo úplně poslední.

Klávesy vhodné pro obsluhu posunu jsou například plus a mínus na numerické klávesnici, neboť logicky označují kladný a záporný směr pohybu. Pro případ notebooků

bez numerické klávesnice by mělo být umožněno také jiné ovládání. Položky v menu jsou dobrou volbou.




*Obr. 30 – Překrývání prvků*




*Obr. 31 – Překrývání prvků*

### 4.6.2 Implementace

Pro posun prvků byly vytvořeny celkem 4 metody ve třídě *NewForm* (*dopredu*, *dopredu1*, *dozadu*, *dozadu1*). Metody nejdříve zjišťují, zda jsou vybrány nějaké prvky. Pokud ano, je aktualizována jejich pozice mezi ostatními prvky (někdy je potřeba aktualizovat pozici více prvků), a také informace pro vykreslování prvků. Pořadí prvků musí být pevně stanovené a bez mezer, aby bylo možné z formuláře vygenerovat XML soubor, neboť se i zde mění pořadí prvků.

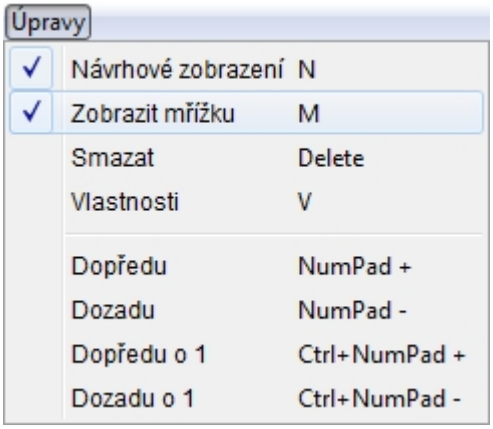
Tyto čtyři metody jsou volány ze třídy *Aplication*, kde jsou přidány do menu *Úpravy* a je tady také přidána podpora pro klávesové zkratky.

### 4.6.3 Klávesové zkratky

- **(plus)** – posun označených prvků úplně dopředu
- **(mínus)** – posun označených prvků úplně dozadu

- **Ctrl + (plus)** – posun označených prvků o jedno místo dopředu
- **Ctrl + (mínus)** – posun označených prvků o jedno místo dozadu

Na obrázku (Obr. 32) je vidět nabídka z menu *Úpravy*, která navíc kromě již uvedených položek obsahuje ještě položku *Smazat*. Ta umožňuje mazání označených prvků, ale o tom bude psáno dále.




*Obr. 32 – Menu „Úpravy“*

## 4.7 Další úpravy a rozšíření

Kromě výše uvedených úprav bylo v editoru formulářů přidáno ještě několik menších rozšíření. Některé opravují chyby, objevené při práci s editorem, jiné zjednodušují a urychlují vytváření formulářů a úpravy v nich.

### 4.7.1 Zachování fontů při ukládání formuláře

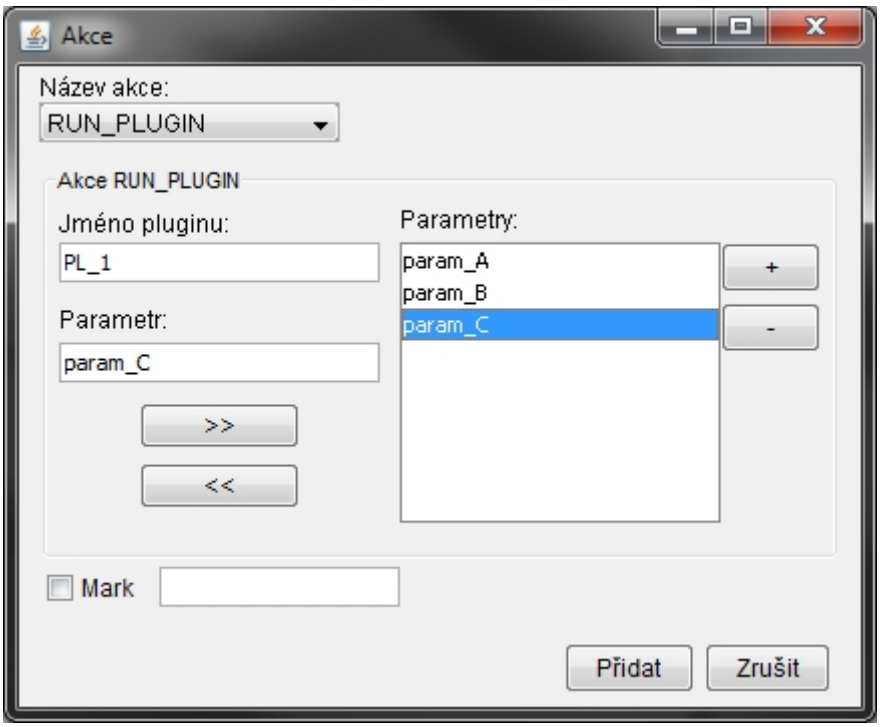
Některé komponenty mají možnost změnit svůj font nezávisle na ostatních, nebo dokonce na fontu celého formuláře. Ve vlastnostech je umožněno měnit písmo, styl i velikost. Chybou editoru docházelo k tomu, že pokud nebyl zvolen font ve vlastnostech samotného formuláře, ale některá z komponent jej nastaven měla (např. tlačítko), tak se po uložení a znovuotevření souboru načetl font, který aplikace nalezla jako první, a ten uložila jako vlastnost celého formuláře. Tato chyba byla nalezena a odstraněna ve třídě *MyXMLDocument*, která se stará o ukládání a otvírání XML dokumentů.

### 4.7.2 Mazání prvků klávesou Delete

Mazání prvků formuláře je docela častá operace, a proto se pro ni vyplatí mít funkci ovládanou klávesnicí. Proto byla vytvořena funkce *smaz* ve třídě *NewForm*, která je volána ze třídy *Aplication* pomocí stisknutí klávesy *Delete*. Funkce se nejdříve pomocí hlášky ujistí, zda uživatel skutečně chce smazat označené prvky. Po potvrzení jsou prvky odstraněny a formulář je znovu vykreslen, tentokrát již bez smazaných prvků. Funkce byla také přidána do menu *Úpravy*, jak je vidět na obrázku (Obr. 32).

### 4.7.3 Přidání akce „RunPlugin“

Pro spouštění externích doplňků, tzv. pluginů, byla formuláři přidána akce *RunPlugin*. Jedná se o vcelku jednoduchou akci zadávanou názvem pluginu a parametry. Dialogové okno pro vložení akce *RunPlugin* je na obrázku (Obr. 33). Dialogové okno panelu bylo vytvořeno ve třídě RunPluginPanel a samotná akce byla přidána mezi již hotové akce.




*Obr. 33 – Dialogové okno RunPlugin*

Akce je pomocí XML identifikována názvem *RUN_PLUGIN*, prvním parametrem je název pluginu, další jsou již samotné parametry pluginu. Název i parametry jsou obsaženy v elementu <*param*>.

Ukázka XML kódu akce:

```
<action name="RUN_PLUGIN">
  <param> PL_1 </param>
  <param> param_A </param>
  <param> param_B </param>
  <param> param_C </param>
</action>
```

### 4.7.4 Formátování výstupního XML

Při ukládání formuláře měl výsledný *XML* soubor nepříliš pěkný formát. Veškeré tagy a jejich obsah byly vypisovány pod sebou vždy na nový řádek, a to včetně prázdných elementů. Strukturované odsazení uvnitř elementů nebylo žádné. Takový způsob prezentace dat byl velmi nevhodný, neboť výsledný dokument činil velmi nepřehledným a dlouhým. Z následující ukázky výsledného *XML* kódu je nepřehlednost a nečitelnost více než patrná.

```
<!-- Předchozí formátování: -->
<items>
<button>
<itemTextID>
$btn_1
</itemTextID>
<bounds x="20" y="20" width="100" height="30">
</bounds>
<text>
Klik
</text>
<onClick>
</onClick>
<description>

</description>
</button>
</items>
```

Ve třídě *MyXMLDocument*, která se o vytváření *XML* stará, byla proto vytvořena funkce *formatXML*, která ze vstupního neformátovaného řetězce vytvoří řetězec formátovaný s odstupňovanou strukturou elementů. Počáteční i koncové tagy velmi krátkých elementů jsou umístěny na jeden řádek a prázdné elementy jsou uvedeny ve zkráceném tvaru. Příliš dlouhé texty jsou rozděleny na řádky po maximálně 80 znacích. Z takto upraveného naformátovaného textu je na první pohled patrná struktura dokumentu a snáze se v ní orientuje, což dokazuje následující ukázka kódu. Jedná se o stejná data jako v neformátovaném případě.

```
<!-- Současné formátování: -->
<items>
  <button>
    <itemTextID> $btn_1 </itemTextID>
    <bounds x="20" y="20" width="100" height="30"/>
    <text> Klik </text>
    <onClick/>
    <description/>
  </button>
</items>
```

### 4.7.5 Změna identifikátorů komponent

Identifikátory komponent byly značeny podle názvu a čísla, což bylo příliš dlouhé. Po domluvě s vedoucím mé práce bylo rozhodnuto udělat v tomto značení drobnou změnu. Názvy byly proto zkráceny na 3 – 4 písmenný kód a bylo také upraveno číslování. Původně byly komponenty číslovány od čísla 1, které postupným přidáváním dalších komponent narůstalo. Například tlačítka byla značena jako *$button1*, *$button2*… Nevýhodou bylo, že ve většině situací jsou identifikátory přepisovány, neboť číslice nevyjadřují o dané komponentě vůbec nic, a také názvy jsou příliš dlouhé např. *$radioButton1*. Nově bylo značení předěláno tak, že první vkládaný prvek je nazván *$btn_* a uživatel ihned za podtržením napíše název, například *$btn_sklad*.

Aby nedocházelo ke kolizím se stejnými identifikátory, je v případě již existujícího prvku identifikátorem *$btn_* vložen jako následující *$btn_1* a další jsou číslovány, podobně jako tomu bylo doposud.

#### *4.7.5.1 Seznam nového označení komponent*

Nově přidané komponenty nemají původní identifikátor, proto je u některých tato položka v prostředním sloupci v tabulce (Tabulka 1) vynechána.

| Název komponenty: | Původní identifikátor: | Nový identifikátor: |
|---|---|---|
| Tlačítko | $button1 | $btn_ |
| Popis | $label1 | $lbl_ |
| Textové pole | $textField1 | $txf_ |
| Textová plocha | $textArea1 | $txa_ |
| Zaškrtávací pole | $checkBox1 | $ckbx_ |
| Radio group | $RadioGroup1 | $rgrp_ |
| Radio button | $RadioButton1 | $rbtn_ |
| Graf | $chart | $crt_ |
| Výběrové pole | $comboBox1 | $cmbx_ |
| Tabulka | $table1 | $tbl_ |
| Listovací lišta | $lister1 | $lsr_ |
| Rámeček | $border1 | $bdr_ |
| Vložený formulář | $eForm1 | $efm_ |
| IconLabel | | $ico_ |
| TabbedPane | | $tbp_ |
| TabbedItem | | $tbi_ |
| GraphicArea | | $gca_ |

*Tabulka 1 – Identifikátory komponent*

### 4.7.6 Podpora klávesových zkratek v editoru formulářů

Kromě již uvedených klávesových zkratek bylo přidáno ještě několik dalších, například pro otevírání a ukládání souborů nebo vytváření nových formulářů.

#### *4.7.6.1 Seznam klávesových zkratek editoru*

- **E**: otevření editoru formulářů z okna aplikace Nahos
- **Ctrl + N**: vytvoření nového formuláře
- **Ctrl + O**: otevření formuláře z XML souboru
- **Ctrl + Shift + O**: otevření formuláře z databáze

- **Ctrl + S**: uložení formuláře do XML souboru
- **Ctrl + Shift + S**: uložení formuláře do databáze
- **M**: zapínání a vypínání zobrazení mřížky
- **N**: přepínání mezi návrhovým zobrazením a náhledem
- **V**: zobrazení okna vlastností formuláře nebo vybraného prvku
- **Delete**: smazání označených prvků
- **(plus)**: posun označených prvků úplně dopředu
- **(mínus)**: posun označených prvků úplně dozadu
- **Ctrl + (plus)**: posun označených prvků o jedno místo dopředu
- **Ctrl + (mínus)**: posun označených prvků o jedno místo dozadu

# ZÁVĚR

Ve své diplomové práci jsem se zabýval rozšířením editoru formulářů informačního systému nástrojového hospodářství ve strojírenském podniku Tajmac-ZPS Zlín, a.s. Nejprve bylo potřeba celý systém kompletně nastudovat, a také zhodnotit původní stav systému a editoru formulářů. Poté bylo nutné připravit návrh pro každý z bodů zadání a takto navržená rozšíření a úpravy implementovat do stávajícího systému.

V průběhu plnění prvního úkolu jsem doplnil editor o možnost zobrazování mřížky, která usnadňuje zarovnávání komponent. Systém jsem rovněž obohatil klávesovými zkratkami, kterými lze ovládat základní a často používané funkce editoru. Nejenže nám usnadňují práci s programem, ale současně ji značně urychlují.

V následující části jsem vytvořil funkci, která umožňuje měnit velikost libovolného prvku bez nutnosti ručního zápisu rozměrů, jako tomu bylo doposud. Velikost každé komponenty lze nyní upravovat tažením myši na požadovanou velikost, a to za pomoci úchytky vytvořené v rohu komponenty.

Jelikož v editoru formulářů chyběla podpora některých stávajících prvků, nebo byly implementovány pouze z části, naprogramoval jsem rozšíření o prvky *IconLabel* a *TabPanel*. Částečně implementované prvky *Menu* a *Chart* jsem rozšířil o nové možnosti a usnadnil jejich editaci. Dalším krokem bylo vytvoření základu pro zcela nový prvek *GraphicArea*, který má sloužit pro vytváření grafických objektů, jako jsou například schémata. Součástí nového prvku je i jednoduchý grafický editační program.

Přidáním podpory komentářů v XML souborech dostal editor formulářů nový rozměr. V minulosti byla editace souborů s komentáři zcela nemožná, nyní lze načítat formuláře z XML souborů, které obsahující komentáře a zpětně detekovat jejich umístění při ukládání.

Během své práce jsem editor formulářů rozšířil o několik dalších funkcí a opravil některé z nedostatků, jež byly objeveny, nebo zadány v průběhu práce. Nové funkce a opravy, které urychlují a usnadňují tvorbu formulářů, byly konzultovány s vedoucím mé diplomové práce, popřípadě s konzultantem. Hromadné označování komponent myší, přeskupování jejich pořadí, klávesové zkratky, formátování výstupu XML a další funkce jsou pro editor zcela jistě přínosem.

Diplomová práce byla zpracována v celém svém rozsahu a všechny body zadání byly úspěšně splněny. Navržená a zpracovaná rozšíření poskytují uživatelům editoru formulářů nejen pohodlnější uživatelské prostředí, ale také mnohonásobně zvyšují produktivitu a efektivitu při vytváření či úpravách formulářů.

# CONCLUSION

In my work I dealt with the extension of the Form Editor of tools management information system in the engineering company Tajmac-ZPS Zlín, a.s. It was need to study the entire system first, and also to assess the initial state of the system and the Form Editor. Then it was necessary to prepare a proposal for each of the points of assignment and implement the proposed extensions and the modifications in the current system.

During the implementation of the first task I have added the editor about the possibility of showing the grid to facilitate alignment of components. I have also enriched the system of keyboard shortcuts, which can control the basic functions and frequently used features. These features make it easier to work with the program, and make it significantly faster.

In the following section, I created a feature that allows you to resize any element without the need for manual entry of dimensions, as has been the case. The size of each component can be adjusted by dragging the mouse to the desired size, with the clips created in a corner of component.

As the Form Editor lacked support for some existing elements, or were implemented only partially, I programmed the extension of the elements *IconLabel* and *TabPanel*. Partially implemented elements *Menu* and *Chart* I extended with new opportunities and facilitated their editing. The next step was to establish the basis for an entirely new element *GraphicArea*, which is used for creating graphics such as diagrams. Part of the new element is a simple graphic editing program.

Adding support for comments in XML files gives a new dimension to the Form Editor. Editing the files with comments was entirely impossible, in the past. Now you can read an XML files that contains comments and re-detect the location of the storage.

During my work I was expanded the Form Editor to several functions and fix some of the deficiencies that were discovered or entered during the work. New features and fixes that accelerate and facilitate the creation of forms, was consulted with the supervisor of my thesis, or a consultant. Mass marking of components with the mouse, rearrangement of their order, keyboard shortcuts, formatting XML output and other features are certainly beneficial.

Master's thesis was prepared in its entirety and all points of entry have been met successfully. Proposed and processed extensions provide users not only more convenient user interface, but also greatly enhance productivity and efficiency on creating or editing forms.

# SEZNAM POUŽITÉ LITERATURY

[1] HORTON I., *Java 5*, Neocortex, Praha 2005, ISBN 80-86330-12-5

[2] BRŮHA L., *Java Hotová řešení*, Computer Press, Brno 2006,

ISBN 80-251-0072-3

[3] VAŘECHA M., *Programové vybavení pro evidenci nářadí ve strojírenské výrobě - vzdálená správa databáze nářadí*, Zlín, 2003. 68 s. Diplomová práce na IIT FT UTB

[4] VERBOVSKÝ J., *Programové vybavení pro evidenci nářadí ve strojírenské výrobě – komunikace uživatele se systémem*, Zlín, 2003. 58 s. Diplomová práce na IIT FT UTB

[5] SVOBODA M., *Rozšíření programového systému pro správu nástrojového hospodářství ve strojírenské výrobě*, Zlín, 2004. 93 s. Diplomová práce na IIT FT UTB

[6] SVOBODA P., *Rozšíření editoru formulářů programového systému NAHOS pro řízení nástrojového hospodářství*, Zlín, 2006. 82 s. Diplomová práce na IIT FT UTB

[7] HLAVÁČEK M., *Rozšíření programového systému NAHOS pro řízení nástrojového hospodářství*, Zlín, 2007. 96 s. Diplomová práce na IIT FT UTB

[8] Architektura informačních systémů [online]. [cit. 2010-03-21] Dostupný z WWW: <http://www.dagblog.cz/2004_11_07_archive.html>

[9] Wikipedia – internetová encyklopedie [online]. [cit. 2010-03-21] Dostupný z WWW: <http://wikipedia.org/>

[10] ZENDULKA J., *Databázové systémy* [online]. [cit. 2010-03-21] Dostupný z WWW:
<http://www.fit.vutbr.cz/study/courses/DSI/public/pdf/nove/10_clsrv.pdf>

[11] POPELKA O., *Aplikační programové vybavení* [online]. [cit. 2010-03-21] Dostupný z WWW:

<https://akela.mendelu.cz/~xpopelka/apv08z/pred/09-aplikace.ppt>

# SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

| | |
|---|---|
| API | Application Programming Interface, rozhraní pro programování aplikací |
| DOM | Document Object Model, objektový model dokumentu |
| IDE | Integrated Development Environment, Vývojové prostředí |
| IS | Informační Systém |
| IT | Informační Technologie |
| JAXP | Java API for XML Processing, Javy rozhraní pro zpracování XML |
| JVM | Java Virtual Machine, virtuální stroj jazyka Java |
| NAHOS | Nástrojové Hospodářství |
| OS | Operační Systém |
| RGB | Red Green Blue, barevný model základních barevných složek |
| SAX | Simple API for XML, jednoduché rozhraní pro analýzu XML |
| SQL | Structured Query Language, strukturovaný dotazovací jazyk |
| XML | eXtensible Markup Language, rozšiřitelný značkovací jazyk |

# SEZNAM OBRÁZKŮ

# SEZNAM TABULEK

# SEZNAM PŘÍLOH

# PŘÍLOHA P I: POPIS GRAFICKÉHO EDITORU KOMPONENTY GRAPHICAREA

## Popis tlačítek a funkcí grafického editoru

Všechny ovládací prvky jsou v editoru umístěny v horní liště a mají podobu tlačítek. Na obrázku 1 je tato lišta zobrazena. Je rozdělena do několika částí, které spolu logicky souvisí a ty jsou odděleny oddělovačem.

*Obrázek 1 – Ovládací lišta grafického editoru*

**Tvary:**

Prvními čtyřmi tlačítky je možno zvolit tvar, který budeme vykreslovat. V současné době editor podporuje pouze základní tvary – čára, obdélník, kruh a elipsa. Který ze 4 tvarů je vybrán, můžeme poznat podle stisknutého tlačítka. Na obrázku 2 je to tvar čáry.

*Obrázek 2 – Tvary*

**Výplň:**

Tvary obdélník, kruh a elipsa mohou být buď prázdné, kdy je vykreslen jen obvod tvaru, nebo mohou být vyplněny barvou. K tomu slouží tlačítko na obrázku 3. Pokud je stisknuto, bude se vyplňovat.

*Obrázek 3 – Výplň*

**Barvy:**

Tlačítka na obrázku 4 slouží k výběru barev. První z nich slouží k obarvení pozadí, které má jako výchozí barvu bílou. Druhým tlačítkem je otevřen dialog pro výběr libovolné barvy, kterou lze vybírat z palety barev. Následují tlačítka pro přímý výběr barvy (bílá, červená, zelená, modrá, černá). Opět je pomocí zobrazeného stisknutí znázorněno, která barva je vybrána (na obrázku je to červená).

*Obrázek 4 – Barvy*

**Režimy:**

Grafický editor se může nacházet ve třech režimech, mezi kterými se lze přepínat pomocí tlačítek na obrázku 5. Prvním a základním je kreslení. Můžeme v něm vykreslovat tvary se zvolenou barvou a případnou výplní. Dalším režimem je přesouvání, v němž lze přesouvat označené tvary na libovolné místo a posledním je otáčení, kdy můžeme označené tvary otáčet podle počátečního bodu.

*Obrázek 5 – Režim*

**Doplňkové funkce:**

V grafickém editoru jsou přidány tři doplňkové funkce pro manipulaci s vykreslenými tvary. První funkcí zobrazenou na obrázku 6 šipkou, je posouvání tvarů na pozadí. Díky tomu lze měnit překrývání objektů. Obě další funkce slouží k mazání objektů. Tlačítko s ikonou popelnice maže jeden označený objekt a tlačítko se sluníčkem smaže úplně všechny vykreslené objekty a vyčistí tak celou kreslící plochu. Mazání samozřejmě předchází dialogová hláška, která upozorňuje na celou operaci.

*Obrázek 6 – Doplňkové funkce*

**Ukládání:**

Poslední dvě tlačítka, uvedená na obrázku 7, grafický editor ukončují a rozdíl mezi nimi je pouze v tom, že tlačítko se zelenou značkou „*v*“ všechny změny uloží a tlačítko s červenou značkou „*x*“ změny ignoruje.

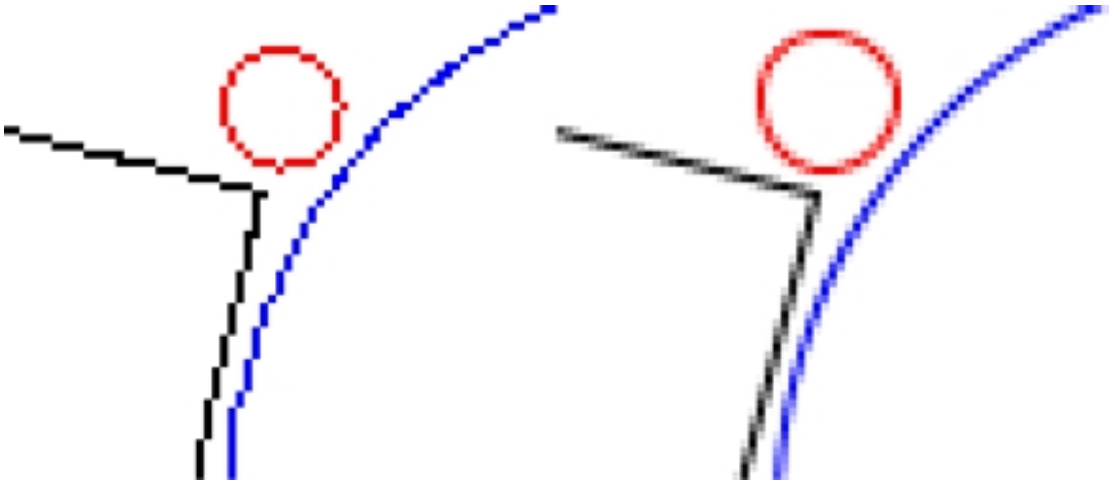
*Obrázek 7 – Ukládání*

## Kreslení objektů a označování pro manipulace

Kreslení objektů je umožněno levým tlačítkem myši. Pravé tlačítko slouží k označování tvarů, se kterými lze poté manipulovat. Po samotném označení objektu se stane hned několik věcí. Označený objekt je obarven speciální barvou, aby jej bylo možno rozlišit od ostatních objektů, a zároveň dojde automaticky ke změně režimu z kreslení na posunutí. Pokud pravým tlačítkem klepneme mimo oblast tvaru a je přitom nějaký objekt označen, dojde k jeho odznačení.

## Antialiasing

Při vykreslování tvarů na plátno grafické komponenty vzniká nežádoucí jev, tzv. aliasing. Je to jev, který je způsoben převodem spojité veličiny na diskrétní. V tomto případě jde o to, že nejmenší možnou zobrazenou jednotkou na monitoru je 1 obrázkový bod a ten má nenulovou velikost.



*Obrázek 8 – Antialiasing*

Metodou, jak tomuto jevu částečně zabránit je vyhlazování (antialiasing). Působení antialiasingu je zobrazeno na obrázku 8. Funkci pro vyhlazování programovací jazyk Java podporuje a je použita v metodě *paintComponent* ve třídě *Screen*.

Ukázka kódu pro vyhlazování ve třídě *Screen*:

```
RenderingHints rh = g2D.getRenderingHints();
rh.put(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);
g2D.setRenderingHints(rh);
```

# PŘÍLOHA P II: POPIS INSTALACE SYSTÉMU NAHOS

Při instalaci systému je nutno udělat několik nových kroků, neboť byly přidány nové funkce a struktury, které vyžadují jisté změny a aplikace musí mít přístup k určitým „*.*class*“ souborům. Ty byly přidány do balíčku *client.jar*, který je dostupný na přiloženém CD, včetně dalších potřebných souborů aplikace a zdrojových kódů. Je také potřeba upravit konfigurační soubor *Client.properties.*

**Popis možné instalace (pro OS Windows):**

1. Namapovat nový disk *Q:* (např. pomocí souboru *makeQ.bat*, přiloženém na CD)
2. Na disk *Q:* nakopírovat celý adresáři *!zps_act*, z přiloženého CD.
3. Nastavit konfigurační soubor *Client.properties*, důležité jsou zejména položky:
   - *JRE version = 1.6.0_18* – správná verze nainstalované Javy
   - *Personal Classpath = Q:\!zps_act\client.jar* – cesta k souboru *client.jar*

# PŘÍLOHA P III: OBSAH PŘILOŽENÉHO CD

- **Aplikace** – adresář s instalačními soubory
  - **makeQ.bat** – vytváří disk Q:
  - **!zps_act** – adresář s aplikací
    - **data** – adresář pro databáze
    - **images** – adresář pro obrázky aplikace
    - **xerus** – adresář pro soubory aplikace
    - **Client.exe** – spustitelná aplikace
    - **client.jar** – balíčky potřebné pro aplikaci
    - **Client.properties** – uživatelský konfigurační soubor
    - **config.properties** – konfigurační soubor aplikace
    - **menuini.xml** – uživatelské menu
- **Dokumentace** – diplomová práce v elektronické podobě
- **Obrázky** – adresář s obrázky pro grafický editor
- **Zdrojové kódy** – adresář se zdrojovými kódy aplikace v Javě