Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

DIPLOMOVÁ PRÁCE

Plzeň, 2007

František Novák

Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

2D editor pro návrh topologie čipu – projekt FlashPoM – 2

Plzeň, 2007 František Novák

Poděkování

Rád bych poděkoval Prof. Ing. Václavu Skalovi, CSc. za vedení mého studia v oboru počítačové grafiky a vedení diplomové práce, kolegům Ing. Janu Kaiserovi a Vojtěchu Hladíkovi, s nimiž jsem spolupracoval na vývoji editoru. Nemalý dík též patří mé přítelkyni, rodině a přátelům za podporu a porozumění.

Práce byla podporována projekty MATEO–FlashPoM, EU INTERREG IIIC a projektem MŠMT ČR, projekt 2C06002 VIRTUAL.

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 17.5.2007, František Novák,……..

Abstract

FlashPoM project is focused on development of techniques for low-cost low lead-time production of customized modular lab-on-a-chip prototypes devices by optimization of the PoM (Polymer-on-Multielectrode array) technology.

This work describes one part of this project, which is to create a low-complexity software tool for chip configuration by combination of a set of predefined elements and modules. These will include polymeric layers with embedded microchannels, vias, microchambers and solid layers with electrode arrays of customizable geometries. Software will also be able to estimate the production cost for a predefined batch size given the selected modules, their arrangement and customized geometrical properties.

Obsah

1 Úvod

Diplomová práce vznikla jako součást projektu FlashPoM, který je zaměřen na optimalizaci laserem vyráběných malosériových zařízení sloužích k výzkumu neuronů. Předpokládá se využití těchto zařízení především v malých a středních společnostech pracujících v oblasti analytické chemie a medicínských aplikací. Větší dostupnost zařízení by měla přinést další rozvoj výzkumu a poznání v dané oblasti.

Projekt je rozdělen na čtyři části, které jsou řešeny v různých institucích. Část, která se zabývá návrhem 2D grafického vektorového editoru, je vyvíjena na Západočeské univerzitě v Plzni a tato diplomová práce je její součástí. Vektorový editor bude sloužit k vytvoření návrhu čipu. Výstupem editoru má být soubor, který bude popisovat čip a na základě tohoto souboru bude možné navržený čip vyrobit. Součástí editoru mají být tyto nástroje:

- nástroj zobrazující rovinný průřez čipu
- nástroj na odhad ceny
- nástroj pro kontrolu vyrobitelnosti navrhnutého čipu
- nástroj zobrazující prostorový model čipu
- nástroj umožňující tisk různých uspořádání vrstev čipu

Vývojem editoru se zabývá čtyřčlenný vývojový tým. Jednotlivými členy týmu jsou Prof. Ing. Václav Skala CSc., Ing. Jan Kaiser, František Novák a Vojtěch Hladík. Václav Skala je vedoucím, Jan Kaiser je zodpovědný za návrh a vývoj editoru. Společně s kolegou Vojtěchem Hladíkem jsem se zabýval vývojem a implementací editoru, který bude napsán v programovacím jazyce C# využívajícím platformu .NET.

Ovládání editoru má být intuitivní a jednoduché. Mezi vykreslovaná primitiva má patřit obdélník, elipsa a úsečka. Nad těmito primitivy mají být definovány booleovské operace a operace seskupení. Editor musí obsahovat pomocné prvky sloužící pro návrh čipu. Mezi tyto prvky patří vodící mřížka a vodící čáry. Dále musí editor obsahovat prvky, které zpřesní a ulehčí uživateli orientaci v průběhu návrhu. Těmito prvky jsou panel vrstev, panel knihoven a panel vlastností.

V průběhu vývoje prací na projektu se moje původní zaměření na návrh datových struktur rozšířilo i o ostatní oblasti související s vývojem editoru. Mimo jiné jsem se zabýval optimalizací vykreslování elementů vrstev, odhadem ceny, validací vyrobitelnosti čipu, 3D zobrazením a jinými funkcemi.

2 Teoretická část

2.1 MATEO

Hlavním cílem projektu MATEO je rozvinout inovační procesy ve výzkumných a vývojových institucích, výměna zkušeností mezi institucemi a spolupráce mezi partnerskými regiony v této oblasti. Partnerskými regiony jsou:

- Katalánsko (Španělsko)
- Lombardie (Itálie)
- Severní Brabantsko (Holandsko)
- jižní a západní Čechy (Česká republika)

Odborné oblasti, na které je projekt zaměřen jsou: potravinářství, medicína a farmacie, obnovitelné zdroje energie, biotechnologie, vyspělé materiály, mechatronika, procesní a produkční technologie.

Cíle projektu

- rozvoj inovačních procesů v malých a středních firmách, výzkumných a vývojových institucích
- výměna zkušeností mezi institucemi a spolupráce mezi partnerskými regiony
- zmapování inovačního potenciálu regionu, diskuse relevantních částí regionálních inovačních strategií mezi všemi partnery projektu, včetně výměny zkušeností a návrhů optimalizace těchto strategií
- příprava podprojektů firem či výzkumných pracovišť, které budou podporovat inovační rozvoj sektorů

Více informací týkajících se MATEO a dílčích projektů realizovaných v rámci tohoto projektu lze nalézt na internetových stránkách Regionální rozvojové agentury jižních Čech viz. [Rera].

2.2 FlashPoM

Jedním z projektů probíhajícím v rámci projektu MATEO je projekt FlashPoM. Projekt FlashPoM je zaměřen na optimalizaci laserem vyráběných malosériových zařízení. Tato zařízení naleznou využití v malých a středních společnostech pracujících v oblasti analytické chemie a medicínských aplikací. Projekt FlashPoM

se zaměřuje na optimalizaci technik pro výrobu malosériových zařízení a to snahou o optimalizaci mikrostrukturní polymerové technologie. Na projektu se podílí tyto univerzity:

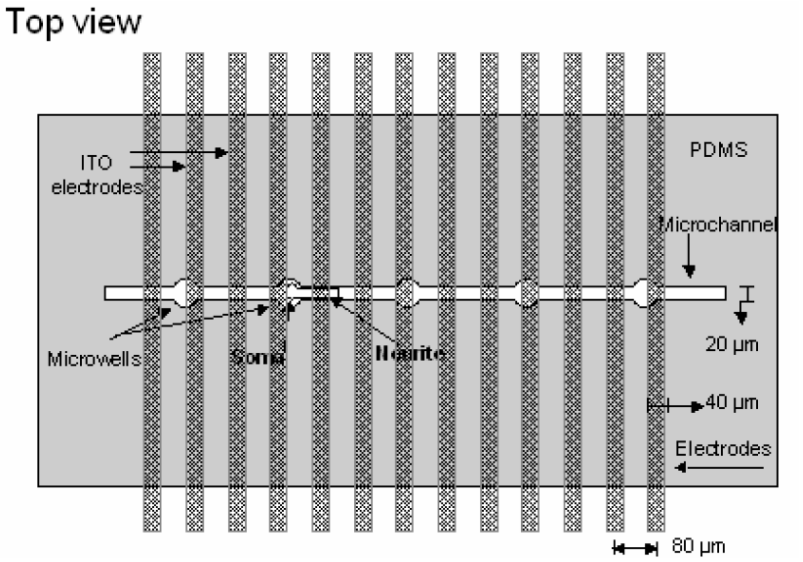
Partnerské univerzity

1	Universitat Politècnica de Catalunya, Barcelona	Španělsko	Katalánsko
2	Technische Universiteit Eindhoven, Eindhoven	Holandsko	Severní Brabantsko
3	University of Milan, Milano	Itálie	Lombardie
4	University of West Bohemia, Plzeň	Česká republika	jižní a západní Čechy

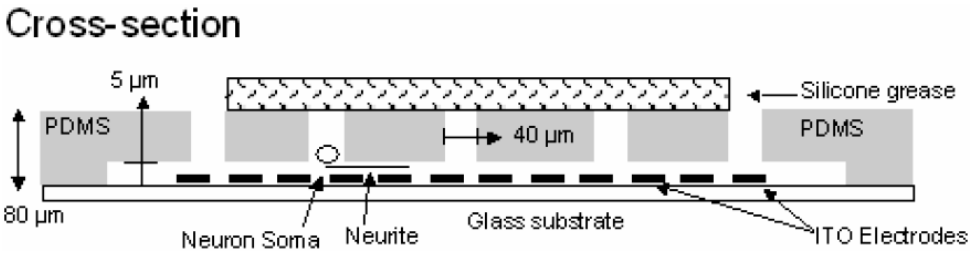
Tabulka 2.2-1: Partnerské univerzity.

Neuročip[1]

Neuročip je zařízení (Obrázek 2-1 a Obrázek 2-2), které je určené k výzkumu neuronů. Čip je složen z tenké elastomerické vrstvy (PDMS) a elektrodového pole (MEA).



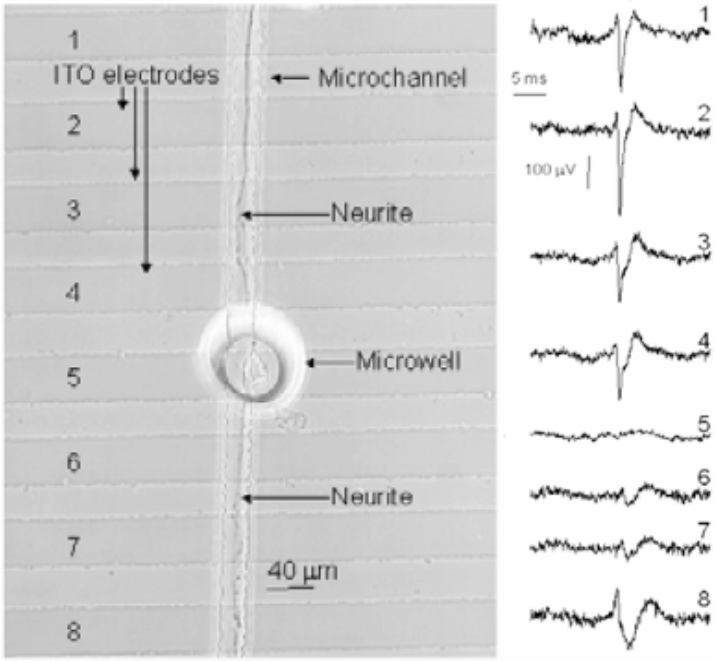
Obrázek 2-1: Nákres čipu z horního pohledu, převzato z [Clav0].



Obrázek 2-2: Nákres průřezu čipu, převzato z [Clav0].

Elektrodové pole je soustava ITO (Indium-Cín-Oxid) elektrod, které jsou propojeny s neurony nebo neuronovými spoji. Elektrody slouží k stimulaci a zaznamenávání aktivit probíhajících mezi neuronovými spoji. Na obrázku (Obrázek 2-3) je v levé části zachycena soustava elektrod propojených s neuronovými spoji a neuronem

samotným. V pravé části obrázku je pak zachycen časový průběh aktivit naměřených na těchto elektrodách.



Obrázek 2-3: Výřez čipu, převzato z [Clav0]

Elastomerická vrstva je umístěna na elektrodovém poli. Vrstva v sobě obsahuje otvory, určené pro vsazení neuronů a mikroskopické kanálky sloužící k vytvoření požadovaného propojení neuronů.

Současné neuronové čipy vyžadují pro výrobu a užití laboratorní prostředí. Levné výrobní postupy zaměřené na hromadnou produkci těchto čipů přispějí k rozvoji výzkumu v oblasti zkoumání neuronů. Kombinace vhodné výrobní technologie, rychlého návrhu a elektrodových polí povede k produkci levných neuronových čipů dostupných pro většinu laboratoří zabývajících se výzkumem neuronů.

Technologie výroby čipu[2]

V této části si přiblížíme postup výroby jednoduchého zařízení – čipu. Jednotlivé kroky procesu výroby jsou schématicky znázorněny na obrázku (Obrázek 2-4).

Proces výroby začíná nanesením vrstvy rezistu. Rezist je chemická substance, která je citlivá na určitý podnět. Podnětem může být světlo nebo elektrony, které dopadají na odkrytá místa. Požadovaná tloušťka vrstvy rezistu je 6 µm. Této tloušťky je docíleno pomocí otáčejícího se zařízení, na které se umístí substrát. Otáčením působí na rezist odstředivá síla, která způsobí rozpínání do stran. V této vrstvě je fotolitografickou metodou vytvořen požadovaný vzor. Vzor určuje místa pro osazení neurony a kanálky pro požadované propojení neuronů.

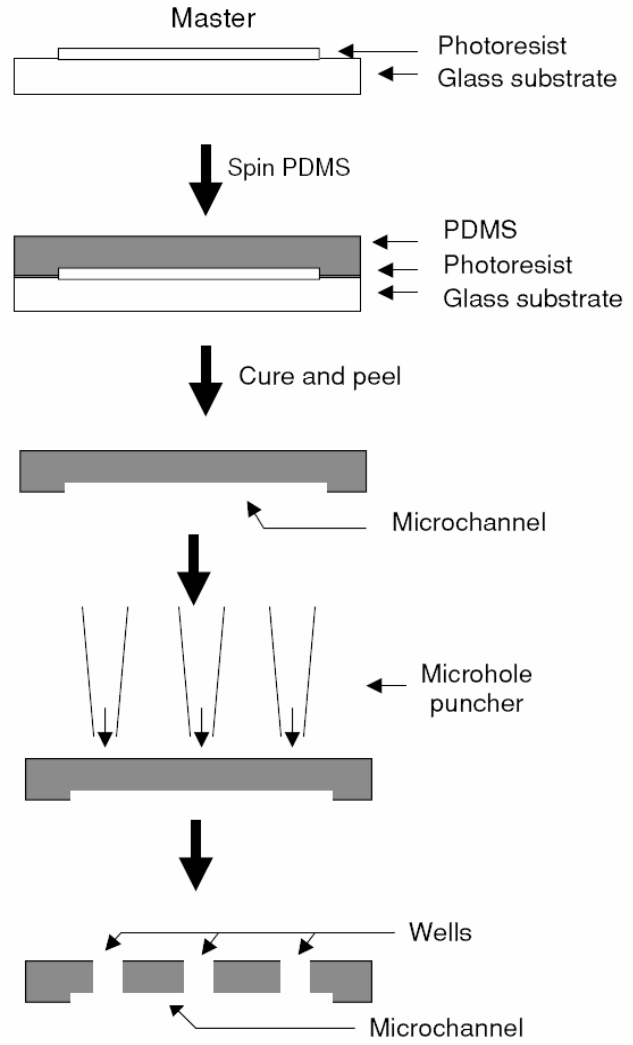
Druhý krok výroby spočívá v nanesení elastomerické vrstvy (PDMS). Způsob nanesení vrstvy je stejný jako u rezistu. Tloušťka této vrstvy je 150 µm. Po nanesení je tato vrstva vytvrzena a sloupnuta. Do sloupnuté vrstvy jsou proraženy otvory, které budou sloužit pro vsazení neuronů.

Posledním krokem výroby čipu je sesazení substrátu a elastomerické vrstvy. K sesazení se používají dva způsoby. V prvním případě jsou elektrody umístěny tak,

aby procházely přesně pod otvory určenými pro osazení neurony. Tento způsob vyžaduje přesné sesazení v rozmezí desítek mikrometrů. V druhém případě je elastomerická vrstva pouze přiložena na substrát a rozmístění elektrod je zcela náhodné.

Výše uvedeným postupem vznikne čip určený pro osazení neuronů. Před vlastním osazením čipu neurony je vnitřní prostor čipu vyplněn roztokem umožňujícím přežití a růst neuronů.

Elektrodové pole (MEA) je naneseno na vrchní části substrátu. Substrát je tvořen skleněnou destičkou s navrch naneseným elektrodovým polem.



Obrázek 2-4: Jednotlivé kroky procesu výroby čipu převzato z [Clav1]

V současné době je vyvíjen nový postup výroby, jehož cílem je urychlení výroby a návrhu čipu. K návrhu čipu slouží grafický editor, který umožňuje vytvořit soubor popisující navrhnutý čip. Na základě tohoto souboru bude vytvořen požadovaný vzoru v rezistu. Proces vytváření vzoru bude automatizován, formování rezistu bude prováděno laserem, který bude řízen počítačem.

[1] Čerpáno z článku [Clav0].
[2] Čerpáno z článku [Clav1].

2.3 Dokument specifikace požadavků

Tato část označená jako dokument specifikace vychází z dokumentu specifikace požadavků, který vypracoval Jan Kaiser v rámci projektu. Některé části jsou v originálním dokumentu podrobněji rozpracovány, než je uvedeno zde. Jedná se o části zabývající se návrhem a popisem uživatelského prostředí.

2.3.1 Úvod

Cílem této práce je jednoznačně určit požadavky kladené na vyvíjený vektorový editor. Editor je určen pro návrh zařízení popsaných v části 2.2. Práce s editorem má být co nejjednodušší a ovládání intuitivní. Při návrhu zařízení bude mít uživatel k dispozici základní primitiva, mezi které patří: obdélník, elipsa a úsečka. Uživatel bude mít k dispozici nástroje pro seskupování primitiv a nástroje pro booleovské operace. Zahrnuty budou booleovské operace: sjednocení, rozdíl a průnik. Editor bude podporovat vícevrstvý návrh čipu.

Pro navrhnutí čipu bude možné vygenerovat soubor, na jehož základě bude navržený čip vytvořen. Editor bude obsahovat funkce umožňující přibližnou kontrolu vyrobitelnosti čipu, funkce pro odhad ceny, 2D řez a 3D vizualizaci čipu.

Cíl projektu

Cílem projektu je vytvoření grafického vektorového editoru pro návrh čipu. Editor musí umožňovat:

- jednoduchý a spolehlivý návrh čipu
- vygenerování souboru, na jehož základě bude vyroben navrhnutý čip

Osoby účastnící se projektu

Vedoucí projektu
Dr. Enric Claverol-Tinturé ectmail@eel.upc.es

Enric Claverol-Tinturé je zadavatelem projektu FlashPoM.

Vedoucí části projektu zabývající se vývojem editoru
Prof. Ing. Václav Skala, CSc. skala@kiv.zcu.cz

Václav Skala je vedoucím skupiny zabývající se vývojem editoru.

Systémový analytik editoru
Jan Kaiser kaiserj@kiv.zcu.cz

Jan Kaiser je zodpovědný za návrh a vývoj FlashPoM editoru.

Vývojový pracovníci editoru
Vojtěch Hladík vojta.hladik@email.cz
František Novák novak_frantisek@centrum.cz

Vojtěch Hladík a František Novák jsou studenti ZČU a jsou programátoři editoru.

Uživatelé editoru
Michael Riss mriss@pcb.ub.es
Carlos Fernandez Escuin kescuin@gmail.com

Michael Riss navrhuje čipy. Ve fázi vývoje editoru představuje Michael typického koncového uživatele editoru.

Carlos Fernandez Escuin se zabývá technologií výrobou čipů. Čip je vyroben na základě informací získaných z výstupního souboru vyvíjeného editoru.

2.3.2 Funkční požadavky

Uživatelské prostředí

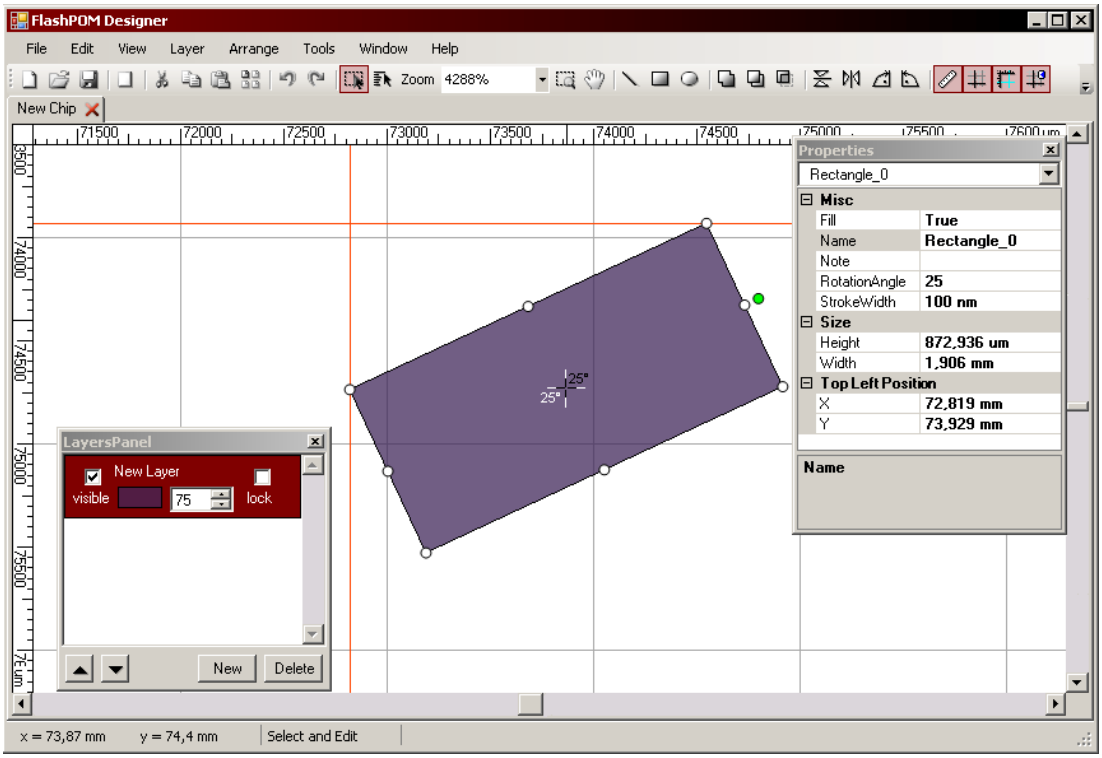
Na obrázku (Obrázek 2-5) je zachycen celkový pohled na editor. V horní části obrazovky se nachází menu, v prostoru pod ním se nachází panel nástrojů, který obsahuje při návrhu často užívané funkce. Největší část obrazovky zabírá plocha sloužící k návrhu čipu. V prostoru pracovní plochy se nachází tři plovoucí panely. Význam jednotlivých panelů bude uveden v následujícím textu.

Menu
Menu se skládá ze sedmi položek. Jednotlivé položky obsahují funkce s podobným zaměřením. V následujícím textu je uveden výčet a stručný popis funkcí obsažených v jednotlivých položkách menu.

File
Položka menu obsahuje funkce pro práci s datovými soubory, které definují popis navrhovaného čipu. Mezi tyto funkce patří: otevření (*Open*), uložení (*Save*, *Save as*), export (*Export to FPSVG*), import (*Import from FPSVG*) a tisk (*Print*) souboru. Dále tato záložka obsahuje funkci pro uzavření aktivního projektu (*Close project*) a funkci pro ukončení aplikace (*Exit*).

Edit
V této položce menu se nacházejí funkce určené pro práci s vykreslovanými elementy. Jedná se o funkce: zkopírovaní (*Copy*), odstranění (*Cut*), vložení elementu do schránky (*Paste*), vrácení elementu do předcházejícího stavu (*Undo*) a obnovení elementu do následujícího stavu (*Redo*).

View
Tato položka menu obsahuje funkce, které zapínají nebo vypínají pomocné prvky sloužící pro přesný návrh čipu. Nachází se zde: pravítka (*Rulers*), mřížka (*Grid*), vodící čáry (*Guide lines*), funkce pro přichytávání k mřížce (*Snap to grid*) a vodícím čarám (S*nap to guidelines*), přibližování (*Zoom in*) a oddalování pohledu (*Zoom out*).



Obrázek 2-5: Uživatelské prostředí editoru

Layer
V této záložce se nachází dvě funkce. První z nich (*New Layer*) přidává novou vrstvu do čipu a druhá (*Delete Layer*) odstraní aktuální vrstvu.

Arrange
V této záložce menu nalezneme funkci pro sloučení elementů do skupiny (*Group*), funkce pro rozdělení skupiny na jednotlivé elementy (*Ungroup*, *Ungroup all*), funkce pro sloučení (*Weld*), odečtení (*Trim*) a průnik (*Intersect*) elementů.

Tools
První funkce této záložky (*Settings*) slouží k nastavení vzhledu uživatelského prostředí a nastavení inicializačních hodnot ovládacích prvků a elementů. Následuje položka kontroly vyrobitelnosti navrženého čipu (*Check constraints*). Další funkce slouží k odhadu přibližné ceny čipu (*Estimate chip cost*), pak následují funkce pro aktualizaci souboru definujícího typy materiálů a ceny materiálu pro výrobu čipu (*Update materials and cost file*). Poslední je funkce pro aktualizaci editoru (*Update FlashPoM editor*).

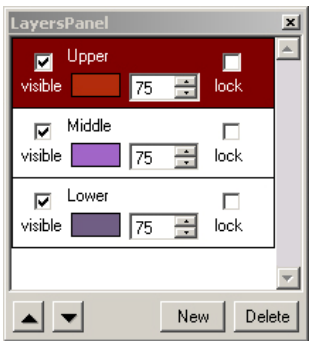
Window
Zde se nacházejí funkce pro zobrazení nebo skrytí panelů nástrojů (*Toolbar*), panelu vrstev (*Layers*), vlastností (*Properties*) a správy knihoven (*Object library*).

Help
Poslední položkou menu je nápověda, která popisuje informace o editoru a základní ovládání editoru.

Panely nástrojů
Panely nástrojů (Obrázek 2-6) obsahují často používané funkce, které jsou zastoupeny jednotlivými ikonami. Požadovaná funkce je vyvolána kliknutím na příslušnou ikonu. Panely nástrojů lze skrývat příp. zobrazovat. Panely poskytují tyto funkce: manipulaci se soubory, vkládání elementů do schránky, obnovení stavu elementů, změna měřítka pohledu, výběr elementu, vykreslení základních elementů, booleovské operace s elementy, zrcadlení a otáčení elementů, zobrazování pomocných prvků při návrhu čipu.

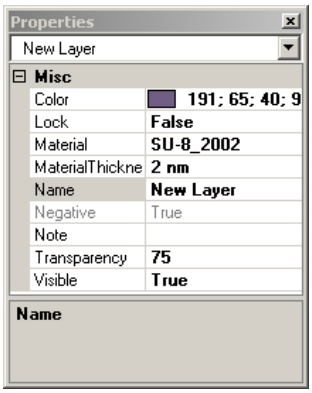
Obrázek 2-6: Panely nástrojů

Panel vrstev
Jak je již z názvu patrné, tento panel (Obrázek 2-7) souvisí s vrstvami. Každá vrstva je v panelu reprezentována jednou položkou. Pozice vrstvy v panelu udává pozici vrstvy v navrhovaném čipu. Uspořádání vrstev v panelu je stejné jako uspořádání vrstev v čipu. Uspořádání vrstev v navrhovaném čipu lze měnit přesouváním položek v panelu. U položek lze nastavit atributy: barva, průhlednost a uzamknutí vrstvy. Barva vrstvy určuje barvu, kterou jsou zobrazeny elementy vrstvy. Průhlednost určuje průhlednost vrstvy, kde 100% znamená zcela průhlednou vrstvu. Uzamknutí vrstvy způsobí, že žádný z elementů nacházející se v dané vrstvě nemůže být editován.



Obrázek 2-7: Panel vrstev

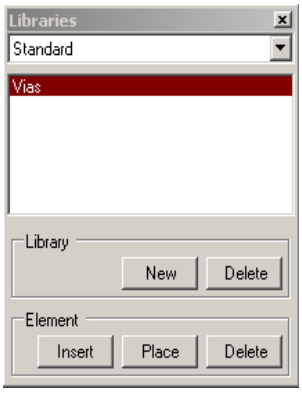
Panel vlastností
Panel vlastností (Obrázek 2-8) slouží k zobrazení a přesnému nastavení atributů objektů. V panelu je vždy zobrazen aktivní objekt. Objekty, které mohou být zobrazené v tomto panelu jsou: vykreslované elementy, vrstvy, čip, vodící čáry a vodící mřížka. Panel umožňuje výběr objektů prostřednictvím rozbalovací nabídky, která se nachází v horní části panelu. Jednotlivé prvky jsou zde hierarchicky uspořádány.



Obrázek 2-8: Panel vlastností

Panel správy knihoven

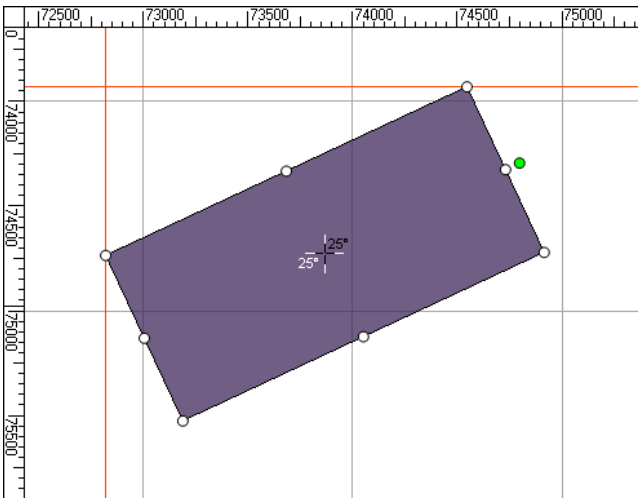
Tento panel (Obrázek 2-9) je určen pro správu knihoven. Knihovny obsahují již vytvořené elementy, které lze znovu použít při návrhu. Uživatel má možnost vytvářet nové knihovny nebo rozšiřovat knihovny stávající. Knihovny lze samostatně distribuovat mezi uživatele a usnadnit jim návrh čipu.



Obrázek 2-9: Panel správy knihoven

Návrhová plocha

Návrhová plocha (Obrázek 2-10) slouží uživateli k zobrazení průběhu návrhu čipu. Pro usnadnění návrhu čipu jsou uživateli k dispozici pomocné nástroje, jako jsou např. pravítka nebo mřížka, které zpřesňují vizuální návrh.



Obrázek 2-10: Výřez návrhové plochy editoru

2.3.3 Formáty souborů

Editor používá dva formáty souborů:

- první z formátů slouží k internímu uložení návrhu čipu
- druhý formát slouží k popisu výroby čipu

První formát je interní formát určený k uložení návrhu. Soubor je obrazem datových struktur popisujících vytvořený čip. Tento souboru je binární a komprimovaný. Soubory použité k uložení návrhu čipu používají koncovkou .fpom.

Druhý formát slouží k popisu výroby čipu. Tento soubor je vygenerován na základě navrženého čipu. Jedná se o textový XML soubor. Popis vykreslovaných primitiv vychází z SVG formátu. Popis primitiv, vrstev a čipu byl navržen tak, aby obsahoval všechny údaje nutné pro výrobu čipu. Koncovka označující tyto soubory je FPSVG.

Pro získání přesnější představy o podobě formátu .fpsvg je pod tímto odstavcem uveden ukázkový soubor. Soubor popisuje čip obsahující jednu vrstvu. Vrstva bude vytvořena z materiálu *SU-8_2002* a jedná se o negativní typ vrstvy. Plocha této vrstvy je definovaná obdélníkovou plochou o rozměrech *3x1 mm* a je natočená o *338°* vzhledem ke středu obdélníka.

```
<?xml version="1.0" encoding="utf-8"?>
<chip numberOfLayers="1" material="CPyrex" materialThickness="0">

  <layer name="New Layer" material="SU-8_2002" materialThickness="2000" negative="True"
number="0">

    <rectangle x="3.607808E+07" y="3.64745E+07" width="3000000" height="1000000" style="fill:True;
          stroke-width:100; rotate:338;" />

  </layer>
</chip>
```

2.3.4 Systémové požadavky

Odhad ceny čipu
Editor musí poskytovat funkci, která určí přibližnou cenu výroby čipu. Cena čipu je dána cenou substrátu, cenou jednotlivých vrstev a cenou odstraněného povrchu z příslušných vrstev. Výsledná cena je určena podle následujících vzorců:

$$cenaCipu = cenaSubstratu + \sum_{vrstvy}\left[cenaVrstvy \; + \sum_{elementy}\left(plochaElementu * k\right)\right]$$

k cena za jednotku práce laseru

Vzorec 2.3-1: Odhad ceny čipu složeného z negativních vrstev

$$cenaCipu = cenaSubstratu + \sum_{vrstvy}\left[cenaVrstvy + \left(cenaVrstvy \; - \sum_{elementy}\left(plochaElementu * k\right)\right)\right]$$

Vzorec 2.3-2: Odhad ceny čipu složeného z pozitivních vrstev

Validace čipu
Editor musí obsahovat funkci, která provede kontrolu vyrobitelnosti čipu. Mezi kontrolované vlastnosti patří:

- vzájemná kompatibilita sousedících vrstev
- minimální velikost elementů
- maximální velikost elementů
- překrývání elementů v rámci jedné vrstvy
- vyrobitelnost čipu, tj. všechny elementy nacházející se ve vrchní vrstvě musí být i ve spodní vrstvě

V případě zjištění některého z uvedených nedostatků, upozorní aplikace uživatele varovným hlášením.

Tisk
Funkce obsažená v editoru, která umožní vytištění navrhnutého čipu. Tisk musí umožňovat různé varianty rozmístění vrstev vzhledem k tištěné stránce. Jednou z variant je umístění jedné vrstvy na tištěnou stránku, nebo varianta umožňující umístění několika kopií jedné vrstvy na stránku.

Zoom
Jde o funkci sloužící při návrhu čipu. Umožňuje uživateli zobrazovat čip v různých detailech. Zobrazitelné rozmezí je od jednotek centimetrů až po stovky nanometrů. Hodnota 100% přiblížení určuje stejnou délku *1 mm* na obrazovce monitoru a substrátu.

Průřez čipu
Tato funkce umožňuje zobrazit kolmý průřez vrstev. Rovina průřezu je určena prostřednictvím přímky zobrazené na návrhové ploše. Aktuální průřez je zobrazován v samostatném okně sloužícím pouze pro zobrazení řezu.

3D Vizualizace
3D vizualizace je určena k prostorovému zobrazení navrženého čipu. Zobrazována je vybraná oblast, která je určena v návrhové ploše. Zobrazeny jsou odpovídající oblasti všech vrstev. Uživatel může oblastí interaktivně otáčet, přibližovat, oddalovat. Uživatel má možnost určovat, které vrstvy budou zobrazeny a které ne.

2.3.5 Mimofunkční požadavky

Hardwarové požadavky
Editor je schopný běhu na kancelářském počítači vybaveném procesorem Intel Pentium III nebo podobným kompatibilním procesorem. Pro 3D vizualizaci je požadována grafická podporující rozhraní Direct3D.

Softwarové požadavky
Aplikace vyžaduje pro svůj běh Microsoft .NET Framework 2.0. Vizualizace vyžaduje Microsoft Direct3D 9.0c rozhraní.

2.4 Řízení softwarových projektů[1]

V této části se budu zabývat řízením softwarových projektů a činností s tím souvisejících. Hlavní fáze řízení softwarového projektu jsou:

- zahájení projektu
- konzultace se zákazníky
- ekonomická stránka projektu
- vytvoření analýzy projektu
- sestavení projektového plánu

2.4.1 Zahájení projektu

Jedna z prvních věcí, kterou je nezbytné udělat na začátku této fáze, je sestavení vývojového týmu a určení vedoucího projektu. Vedoucí projektu je člověk, který má na starosti technickou realizaci projektu a řízení vývoje. Vedoucí projektu sestaví vývojový tým. Vývojového týmu je zpravidla sestaven z pracovníků nebo zaměstnanců firmy, kteří jsou volní. V lepším případě si z nich může vybrat. Vedoucí týmu by měl znát silné a slabé stránky každého ze svých podřízených a měl by dokázat poměrně přesně odhadnout, jakou činnost by měl který člověk vykonávat a za jak dlouho ji bude schopen dokončit. Vedoucí týmu by tak měl být manažer, psycholog, ekonom a programátor v jedné osobě. Každý projekt může vyžadovat jiné složení vývojového týmu a jiné požadavky na jeho jednotlivé členy. Role členů musí být rozděleny vyváženě, a i když všichni chtějí programovat, musí se najít někdo, kdo bude psát dokumentace a testovat. Z tohoto důvodu si vedoucí týmu musí vybrat takové členy týmu, kteří budou plnit jim přiřazený úkoly a s touto rolí budou spokojeni. Dále je nutné, aby každý člen týmu byl zastupitelný. V týmu musí existovat někdo, kdo bude schopný převzít práci jiného člena týmu.

V několika následujících odstavcích si uvedeme pravidla (doporučení), kterým je vhodné věnovat pozornost v této fázi projektu. S pomocí těchto pravidel se můžeme vyhnout některým nepříjemnostem a omylům, které by mohly vést k nezdaru nebo prodloužení projektu. Pravidla se mohou lišit podle typu projektů a proto se jimi nelze vždy striktně řídit. Podle typu projektu si musíme umět vybrat ty pravidla, která lze uplatnit na daný projekt. Mezi tyto postupy patří: první rozhovor s klientem, smlouva se zákazníkem a ekonomická stránka projektu.

Typy projektů podle zadavatele

Jednou z nejdůležitějších vlastností projektu je to, pro koho je projekt vyvíjen, tedy ten, „kdo platí“. Na základě tohoto kritéria lze rozlišovat čtyři typy projektů. Jednotlivé typy jsou:

- interní projekt
- projekt pro volný trh
- projekt na zakázku
- projekt jako subdodávka komplexního řešení

Interní projekt
Jedná se o projekt, jehož zadavatelem je vlastní firma. V takovémto případě není nutné zpracovávat smlouvu a další projektové dokumenty stačí připravit neformálně. Ovšem i u takovéhoto projektu nelze podcenit zadavatele, kterým je nejčastěji manažer firmy nebo jiné oddělení, které nestanoví přesně cíl projektu. Pokud není přesně stanoveno, jak má výstup projektu vypadat, dají se těžko stanovit náklady na projekt a doba, jak dlouho bude vývoj projektu trvat. Na druhou stranu tento typ projektu má výhodu komunikace s pracovníky, kteří se nacházejí ve stejné budově nebo jsou snadno dosažitelní. Ovšem ani tato skutečnost se nesmí podcenit a přístup typu „vždyť se na to mohu kdykoliv zeptat“ nevede ke zdárnému konci. Nejlepší je k takovému projektu přistupovat jako k externí zakázce pro externího klienta.

Projekt pro volný trh
I v tomto případě je zadavatelem vlastní firma. Rozdíl oproti předchozímu typu projektu je v koncovém uživateli. Tím je v tomto případě zákazník, který si aplikaci koupí a bude jí používat. V tomto případě je nejčastěji zadavatelem projektu manažer, který aplikaci distribuuje. Manažer není v přímém kontaktu s vývojovým týmem, na projekt dohlíží „zvenku“ a zastupuje potencionální zákazníky. Požadavky na funkčnost a vzhled aplikace jsou určeny zadavatelem na základě diskuze s potencionálními zákazníky a na základě dalších marketingových průzkumů. Na vývoji aplikace se mohou podílet budoucí zákazníci a právě oni mohou přispět ke zkvalitnění produktu. Nejdůležitějším kritériem takovýchto aplikací je kvalita. Produkt má smysl uvádět na trh pouze tehdy, když je překonána konkurence a je jistota, že se produkt stane prodejným. Čas uvedení aplikace na trh se občas protahuje a to z důvodu rozšíření nových funkcí produktu nebo odstranění chyb.

Projekt na zakázku
Jedná se o jeden z nejznámějších typů projektů. Zadavatelem projektu je zákazník, který platí vývoj aplikace a zároveň určuje požadavky kladené na aplikaci. Na začátku vývoje se obě strany domluví na přesných a ověřitelných požadavcích, které bude aplikace splňovat. Na základě této dohody je vytvořen časový a finanční plán. Cílem je vytvořit aplikaci, která bude splňovat požadavky kladené zákazníkem. Snaha vytvořit aplikaci, která výrazně překonává požadované kritéria, představuje hrubou chybu vedoucího projektu.

Projekt jako subdodávka komplexního řešení
Vývoj produktu je prováděn na zakázku systémového integrátora. Systémový integrátor je zákazníkem pověřen na základě smlouvy komplexním řešením aplikace. To však neznamená, že je jeho jediným dodavatelem. Běžně pověřuje zajištěním jednotlivých odborných úkonů další dodavatele, kteří se na danou oblast specializují. Systémový integrátor odpovídá za kvalitu řešení subdodavatele, především za slučitelnost s ostatními částmi aplikace. Tento projekt je ze své podstaty velkým zdrojem problémů. Proto je důležité věnovat velké úsilí vyjasnění podmínek projektu s integrátorem.

2.4.2 Postupy při konzultacích s klienty

Tato část nastává poté, co se firma rozhodne na daný projekt přistoupit. Jedná se o situaci, kdy se poprvé setkává vedoucí projektu s klienty. Tato fáze představuje klíčový moment celého projektu. Na prvním setkání je důležité udělat dobrý dojem a

to i v případě, že projekt nakonec nezískáte. Je velice pravděpodobné, že se na vás zákazník může obrátit někdy v budoucnu.

První otázka, na kterou byste se měli snažit získat odpověď, je, z jakého důvodu klient uvažuje o nové aplikaci. Nejčastější odpověď je, že klient aplikaci potřebuje ke své činnosti. Ale můžeme narazit na případy, kdy klient očekává, že s aplikací získá „know-how“, které nemá. Občas se také stává, že klient neví, co přesně požaduje a odvolává se na již existující aplikaci. V takovýchto případech, nebo v případech kdy, se vám něco nezdá, je důležité si uvědomit, že máte právo do projektu nevstoupit.

Při rozhovorech s klienty je důležité, mít při vyjednávání člověka, který se v dané oblasti problému vyzná. Takovýto člověk může být neocenitelný jak při vývoji, tak i při prodeji vytvořené aplikace. Na tuto činnost je vhodné přijmout někoho, kdo v tomto oboru pracuje a je schopen při návrhu aplikace účinně spolupracovat.

Při vytváření specifikace požadavků se klient snaží do programu zařadit vše, na co si vzpomene. Klient nemá představu, které funkčnosti je jednoduché do aplikace přidat a které ne. Dále si zákazník musí uvědomit, že na vytvoření aplikace klade určité omezující faktory (rozpočet, peníze) a vývoj aplikace bude těmito faktory ovlivněn. Vhodné je rozdělit si požadavky klienta podle důležitosti do skupin a následně společně s klientem rozhodnout, které skupiny mají být do aplikace zařazeny a které ne.

Smlouva se zákazníkem

Smlouvu se zákazníkem uzavírá vedení firmy. Ovšem za dodržení smlouvy je zcela zodpovědný vedoucí projektu, a tudíž by měl na jejím vytváření spolupracovat.

Smlouva by měla být vyvážená, to znamená, že by měla obsahovat povinnosti obou stran. Ze strany zadavatele jde o to, aby byl kryt v případě, že vývoj nepostupuje tak jak má. Ze strany firmy by smlouva měla obsahovat povinnosti druhé strany například při zkušebním provozu. Dále by smlouva měla obsahovat co nejpřesnější specifikaci produktu a tato specifikace by měla být přílohou smlouvy. Smlouva by měla přímo obsahovat kritéria, podle kterých je určeno, zda výsledná aplikace splňuje kladené požadavky. Dále by neměla chybět část, která bude popisovat postup při změně specifikace.

2.4.3 Ekonomická stránka projektu

Úspěšný projekt je takový, který vydělá dostatek peněz, ze kterých se nejen zaplatí vývoj, ale přinese i zisk. Obvykle se zadavatel a někdy i vývojáři snaží do vytvářeného projektu nabalit co nejvíce funkcí. To však z pohledu ekonomického nemusí být správné. Je důležité si uvědomit, že náklady na projekt rostou rychleji než lineárně. A je na vedoucím projektu, aby náklady na projekt sledoval a udržoval projekt ve schváleném rozpočtu. Hlavní položku v nákladech na vyvíjený software představuje práce vývojářů. Výpočet této položky je určen z hrubé mzdy příslušných vývojářů. Dále je do nákladů potřeba započítat provozní režijní náklady. Pod těmi náklady se skrývá plat managementu, marketing, ale také například počítače, připojení na internet a poplatek za elektřinu.

Určení ceny projektu
Určení ceny je úkolem obchodníka. Vedoucí projektu k němu bývá přizván, nebo alespoň připravuje podklady pro odhad ceny. Určení ceny se dá přirovnat k smlouvání na trhu, kde se každá strana snaží získat maximum. Nejčastěji se vyskytují dva způsoby stanovení ceny aplikace:

- zaplacení veškerých nákladů plus přirážka
- určení pevné ceny

Jak je z názvu první varianty patrné, klient zaplatí celkové náklady plus nějakou přirážku. Výhodu klienta je, že může v průběhu vývoje stanovovat další požadavky. Nese však riziko, že cena projektu se může zvyšovat s počtem požadavků. U druhé varianty je cena stanovena na začátku projektu a dodavatel plní přesně specifikaci požadavků.

2.4.4 Analýza

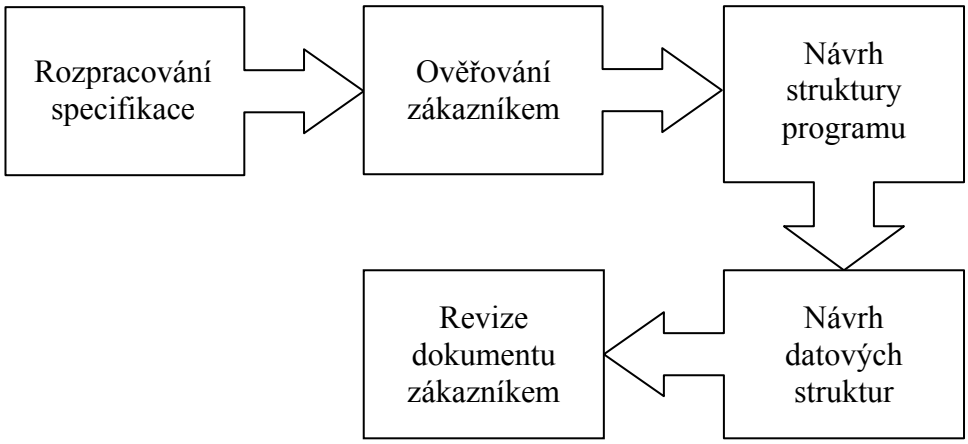
Důležitá fáze projektu, která nastává po uzavření smlouvy. V této fázi jsou ujasněny, ověřeny a rozpracovány požadavky uživatelů, které jsou na projekt kladeny.

Na analýze se zpravidla podílí menší tým než při vlastním vývoji aplikace. V tomto týmu jsou specializovaní analytici nebo zkušení vývojáři. Jejich úkolem je zajistit hladký průběh vývojových prací. Naprosto klíčová je v této fázi spolupráce zadavatele s analytickým týmem.

První věc, kterou je potřeba v této fázi projektu ujasnit, jsou požadavky uživatele. První verze specifikace projektu je součástí smlouvy a nyní je třeba všechny požadavky ověřit a podrobně rozpracovat. Analýza vyžaduje úzkou spolupráci budoucích uživatelů a designéra vývojového týmu při navrhování uživatelského rozhraní. V této fázi je uživatelské rozhraní vytvořeno ve formě obrázků a náčrtů, jak by mělo vypadat. Současně s touto činností je navrhnuta architektura aplikace a její rozdělení na jednotlivé moduly. Na základě těchto informací může vedoucí projektu poměrně přesně určit termín dokončení projektu, náklady na projekt a určit případná rizika související s vývojem aplikace. Během vývoje se může přijít na to, že některý požadavek byl opomenut, nebo že se některý požadavek musí být upraven. S tím musí vedoucí projektu počítat a tyto vlastnosti zahrnout do plánu projektu. Ovšem mělo by jít o výjimečné případy. Pokud je takovýchto případů hodně, ukazuje to na špatnou analýzu projektu a může to ohrozit úspěšné dokončení projekt.

Postup analýzy

Jednotlivé kroky analýzy a jejich posloupnost se mohou lišit podle typu projektu, zkušeností vývojového týmu, ochotě a schopnosti zadavatele spolupracovat a na mnoha dalších kritériích. V závislosti na složení analytického týmu lze na některých pracích pracovat paralelně. V této době mohou začít programátoři vytvářet prototypy aplikace, podle kterých může vývoj postupovat.



Obrázek 2-11: Postup analýzy, převzato z [Pal]

Rozpracování specifikace
Pod rozpracováním specifikace se rozumí podrobné určení toho, co by měla daná aplikace dělat. K vyjádření toho se používají tzv. uživatelské případy. Pod pojmem uživatelský případ se rozumí jedna elementární funkce programu, která provádí jeden příkaz uživatele, který s programem pracuje. Uživatel se nazývá aktér a aktérem nemusí být jenom člověk, ale také jiná aplikace. Příkladem uživatelského případu v projektu FlashPoM může být vykreslení libovolného elementu na kreslící plochu, nebo editace již vykresleného elementu.

Ověření představ zákazníkem
Název tohoto odstavce zcela vystihuje jeho obsah. Poté, co byla rozpracována specifikace, je nutné prodiskutovat se zákazníkem, zda specifikace odpovídá jeho představám. Zákazník se tak může od počátku vývoje podílet na vývoji aplikace a tím získá představu o aplikaci. Může také upozornit na některé věci, které se mu nezdají. Tyto sporné části mohou být se zákazníkem diskutovány a upraveny. Na základě rozpracované specifikace zkonzultované se zákazníkem jsou vytvořeny prototypy aplikace, na kterých je klientům demonstrován vzhled a funkce aplikace. Klient tak získá představu o aplikaci.

Návrh datových struktur a struktury programu
V této části jde o rozdělení uživatelských případů do jednotlivých datových nebo objektových modulů. Uživatelské případy jsou prostupně dekomponovány na posloupnost operací s datovými objekty. Tímto přístupem dospějeme k třívrstvé architektuře. Funkce vykonávající uživatelské případy vytvoří vrstvu služeb, operace s datovými objekty představují vrstvu služeb a datovou vrstvu reprezentují jednotlivé datové objekty. To jakým způsobem jsou rozděleny jednotlivé vrstvy do modulů, záleží na charakteru projektu a na zkušenosti vedoucího projektu.

Revize dokumentu zákazníkem
Produktem předchozích částí jsou různé dokumenty. Zákazník by měl jednotlivé dokumenty prostudovat a k případným nedostatkům a nejasným částem vyjádřit své připomínky. Zcela ideální je, pokud se zákazník přímo podílí na návrhu jednotlivých dokumentů.

Výstupem analýzy je několik dokumentů, které jednoznačně popisují požadavky klienta kladené na vyvíjenou aplikaci a jednoznačně popisují výsledný produkt, který

má vzniknout. Mezi dokumenty by také měl být dokument popisující strukturu aplikace a dokument popisující projektový plán vývoje projektu. Po dokončení fáze zabývající se analýzou nastává pracovní fáze. V této fázi je implementována aplikace podle dohodnutých specifikací ve fázi analýzy.

2.4.5 Projektový plán

Projektový plán je poslední velkou částí analýzy. Vytvoření plánu má na starosti projektový manažer. Vývoj může začít v okamžiku, kdy máme dostatek potřebných informací. Projektový plán lze rozdělit na čtyři základní úkoly:

- odhad času
- určení závislosti prací
- rozvrh úkolů pro jedince
- kontrola průběhu a postupu

Vytvořit přesný projektový plán není jednoduché a ve své podstatě to ani není možné. Plán musí zahrnovat i nepředvídané či mimořádné situace. Může se stát, že některá část projektu bude náročnější než se předpokládalo, nebo se některá část nedokončí v termínu. Tedy i tyto události by měl projektový plán předpokládat.

Prvním krokem při tvorbě plánu je rozdělení prací na dostatečně malé úkoly. Rozdělením do drobných úkolů se vyhneme nebezpečí přehlédnutí skrytých problémů, na které by se jinak narazilo až při implementaci úkolu. Dále lze u těchto úkolů poměrně přesně určit náročnost. Následujícím krokem je určení návaznosti jednotlivých kroků. Jde o určení vazeb mezi jednotlivými elementárními úkoly. Tímto se vyvarujete situacím, kdy vývojáři budou čekat na dokončení jiné části aplikace, kterou potřebují pro pokračování v svém dalším vývoji. Vhodné je stanovit milníky, které budou vyžadovat splnění určité kategorie úkolů. Nejtěžším úkolem je odhad, jak dlouho budou jednotlivé činnosti trvat. Proto je výhodné se na odhadu poradit s vývojářem, kterému je daná část přidělena. Důležité je odhady nepřecenit a ani nepodcenit a určité si rozumné rezervy. Posledním krokem v projektovém plánu je výpočet nákladů a doby trvání vývoje projektu.

U malých standardních projektů není třeba plánování příliš přehánět, ale i u takovýchto projektů platí, že jakýkoliv projektový plán je lepší než žádný.

[1] Čerpáno z [Pal]

2.5 Platforma .NET Framework

Platforma .NET je vyvíjena společností Microsoft. Skládá ze dvou hlavních částí:

1. společné jazykové prostředí pro běh aplikací (CLR)
2. knihovny (FCL)

Cílem .NETu je zlepšit složitost současných systémů Windows. Mezi hlavní rysy této platformy patří: znovu využívání kódu, specializace programového kódu, nezávislost na programovacím jazyku, bezpečnost, řízení zdrojů, nasazování a správa aplikací. Při návrhu platformy .NET se Microsoft snažil vylepšit některé z nedostatků současné platformy. Na několika následujících příkladech je uvedeno několik výhod, které tato platforma přináší:

- **Nezávislost na platformě** – nezávislosti na platformě je docíleno využitím asembleru IL (Intermediate Language). Jedná se o jazyk, do kterého je kód aplikace při překladu převeden a teprve při spuštění aplikace je kód jazyka IL převeden do asembleru dané platformy.
- **Automatická správa paměti** – zdroje alokované aplikace, které již nejsou používány, jsou automaticky uvolňovány.
- **Typová bezpečnost** – typová bezpečnost souvisí s přístupem k objektům. S každým datovým typem jsou udržovány popisné informace a na základě těchto informací je možné každý datový typ ověřit. Typová bezpečnost znamená, že nelze použít odkaz na libovolné místo v paměti.
- **Opakované použití kódu** – třídy vytvořené v libovolném programovacím jazyce založeném na platformě .NET můžeme jednoduše opakovaně použít ve svých aplikacích, nebo je můžete distribuovat ostatním uživatelům.
- **Předpřipravené typy** – součástí .NETu je velké množství balíčků (assemblies) .NET (Framework Class Library, FCL), které obsahují velké množství již předpřipravených tříd, které programátoři využijí při tvorbě vlastních aplikací.
- **Interoperabilita** – .NET podporuje spolupráci s COM komponentami i s DLL knihovnami. Tímto je zajištěna kompatibilita s předchozími technologiemi a odpadá tím nutnost přepisování již vytvořeného kódu.

2.5.1 Common Language Runtime (CLR)

Společné běhové prostředí - prostředí, které ke svému běhu využívají aplikace vytvořené v .NETu. CLR rozumí jedinému jazyku a tím je jazyk IL (někdy také označován jako mezijazyk). Prostředí představuje vrstvu, která převádí jazyk IL do strojového kódu procesoru. Zdrojový kód může být napsán v libovolném programovacím jazyce, který podporuje CLR. Podporou je myšleno to, že překladač daného programovacího jazyka převede při kompilaci soubory se zdrojovým kódem do jazyka IL. Výsledkem je tzv. řízený modul. Řízený modul je přenositelný a spustitelný (portable executable - PE) soubor, který ke svému spuštění vyžaduje

CLR. Řízený modul obsahuje tyto částí: hlavička PE, hlavička CLR, metadata a kód v jazyku IL. Přesný význam jednotlivých částí je podrobně rozepsán v [Rich, str.31]. Zjednodušeně řečeno, řízený modul musí obsahovat všechny informace potřebné ke svému spuštění. Mezi tyto informace patří: údaje o typu souboru, verze CLR, pro kterou je soubor určen, tabulky popisující členské prvky a importované typy a kód v IL jazyce.

2.5.2 Framework Class Library (FCL)

Knihovna typů - standardní součást .NET Framework. Obsahuje velké množství assembly, které vykonávají určité specifické funkce. Pro práci s konkrétními typy, které nabízí určité funkce, musíme znát celý název jmenného prostoru. Přidávání nových jmenných prostorů a jejich používání je velmi snadné. Mezi často používané jmenné prostor patří: *System*, *System.Collections*, *System.Collections.Generic*, *System.Security*, *System.IO*, *System.Net*, *System.Drawing, System.Windows.Form*, *System.Runtime.InteropServices* a mnoho dalších.

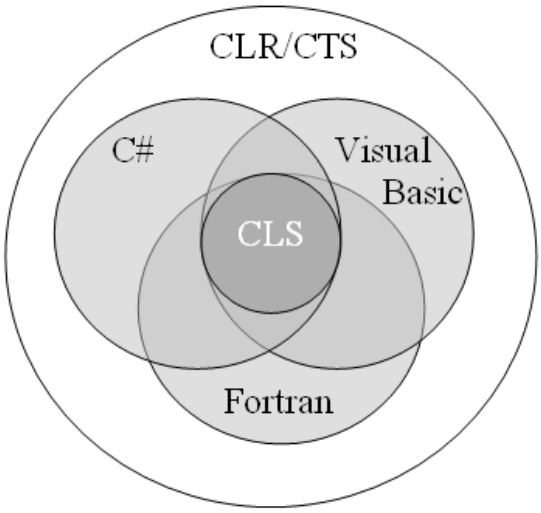
2.5.3 Common Type System (CTS)

Společný typový systém – formální specifikace vytvořená společností Microsoft. Cílem této specifikace je určit, jak se jednotlivé datové typy definují a chovají. CTS definuje pravidla pro:

- členské prvky typů, kterými jsou: datová položka (*filed*), metoda (*method*), vlastnost (*property*) a událost (*event*)
- přístupové modifikátory k členským typům a prvkům. Modifikátory jsou private, public, protected, internal. Názvy modifikátorů se mohou lišit podle typu jazyka, ale vnitřně mají jednotný význam.
- dědičnost, virtuální metody, životnost objektů - pravidla jsou navržená tak, aby vyjadřovala sémantiku vyjádřitelnou v moderních programovacích jazycích
- dědičnost od typu *System.Object*

2.5.4 Common Language Specification (CLS)

Společná jazyková specifikace - definuje funkce, které musí být v daném programovacím jazyce podporovány. Tato definice určuje autorům překladačů povinnost podporovat určitou sadu funkcí. CLS je minimální podmnožinou funkcí CLR/CTS. Při dodržení této specifikace je možné vytvářet datové typy, které lze používat v různých programovacích jazycích podporujících CLS. Různé programovací jazyky podporují různě velikou skupinu funkcí CLR/CTS. Kompletní seznam pravidel pro CLS je uveden v .NET Framework SDK v oddílu Cross-Language Interoperability.



Obrázek 2-12: Podmnožina CLR/CTS, převzato z [Rich]

2.5.5 Intermediate Language (IL)

Mezijazyk - strojový jazyk nezávislý na platformě procesoru. Byl vytvořen společností Microsoft na základě konzultací s významnými autory překladačů. IL je strojový jazyk na vyšší úrovni abstrakce než stávající strojové jazyky. Umožňuje určovat objektové typy a obsahuje instrukce pro vytváření a inicializaci objektů, volání virtuálních metod objektů a přímou manipulaci s prvky pole. Dále obsahuje instrukce na obsloužení výjimky. Jde tedy o objektově orientovaný strojový jazyk.

Programovací jazyky většinou využívají pouze část vlastností CLR. Nabízí se tedy možnost psát zdrojový kód přímo v IL jazyce. Na tuto možnost Microsoft pamatoval a společně s platformou .NET jsou distribuovány aplikace umožňující psaní kódu v jazyce IL.

IL je založen na zásobníkové struktuře, tj. všechny instrukce ukládají operandy na zásobník a výsledky operací jsou čteny z vrcholu zásobníku. Jazyk zcela abstrahuje od procesorové jednotky na daném počítači a proto neobsahuje žádné funkce pro práci s registry procesoru. Instrukce jsou beztypové, tj. nerozlišuje se mezi 32bitovým nebo 64bitovým instrukcemi. Při běhu IL je prováděna kontrola datových typů, počtu parametrů metod a čtení paměti. Cílem je zajistit bezpečné provádění operací.

2.5.6 Mono

Mono je projekt vedený firmou Novell (dříve firmou Ximian). Jeho cílem je vytvořit sadu nástrojů kompatibilních s prostředím .NET, které splňují standardy ECMA. Mezi tyto nástroje patří i překladač jazyka C# a Common Language Runtime. Mono může běžet na počítačích s operačními systémy Linux, FreeBSD, UNIX, Mac OS X, Sun Solaris a Microsoft Windows.

V současné době dospělo Mono do verze 1.2.3. Tato verze zcela nepodporuje všechny datové typy definované v .NET Frameworku 2.0. Ne zcela podporovány jsou datové typy z assembly *Windows.Forms*. Kompletní podpora .NET Frameworku 2.0 je plánována na konec roku 2007. Podpora .NET Frameworku 3.0 je ve fázi příprav.

2.6 UML (Unified Modeling Language)

Pod touto zkratkou se skrývá modelovací jazyk. Modelovací jazyk UML vznikl v průběhu 80. a 90. let pro popis objektově orientované analýzy a návrhu. V roce 1995 byly zahájeny práce na sjednocení různých metod a syntaxí sloužících pro modelování a návrh. Tyto práce byly zahájeny pod záštitou firmy Rational. Výsledkem těchto prací byl modelovací jazyk UML. Jednalo se o souhrn metod doporučení, které se postupně staly standardem.

Modelovací jazyk UML je souhrnem především grafických notací k vyjádření analytických a návrhových modelů. Dále umožňuje modelovat jednoduché i složité aplikace pomocí stejné formální syntaxe. Díky této vlastnosti lze sdílet výsledky mezi návrháři. Vybrané modely jsou pochopitelné i pro zadavatele aplikace a umožní vyjasnění požadavků zadavatele na vyvíjenou aplikaci. Diagramy zachycují určitý pohled na aplikaci, nikoliv pohled na aplikaci jako na celek.

UML je jazyk pro vizualizaci, specifikaci, stavbu a dokumentaci softwarových systémů. Existují různé metodické postupy, které vycházejí z UML. Jde o různá rozšíření postupů, o vlastní postupy, diagramy a techniky.

2.7 Objektový přístup

2.7.1 Objektově orientované programování

Modelovací jazyky zavádí pojem objekt. V objektově orientovaném programování se společně s objekty vyskytují další termíny jako např. třída, atribut, metoda, dědičnost, zapouzdření a polymorfizmus. Ale co vlastně znamenají tyto termíny? Odpověď na tuto otázku je uvedena v následujících odstavcích, ve kterých jsou vysvětleny výše uvedené termíny a tím je ve zkratce vysvětlena podstata objektového přístupu.

Objekt
Objekt představuje prvek modelované reality. Objekt je obrazem prvku, který existuje v modelovaném prostředí. Každý objekt je určen svým stavem a chováním. Stav objektu určují atributy a chování objektu určují metody, které objekt poskytuje. Při komunikaci objektu s okolím využíváme metod, které objekt poskytuje. Při využívání metod objektů nemáme žádné informace, jakým způsobem a jak daná metoda pracuje.

Třída
Třída představuje šablonu, podle které je vytvořen příslušný objekt. Třída je tvořena atributy a metodami. Atributy jsou proměnné a slouží k zachycení stavu objektu. Třída tedy neurčuje konkrétní hodnoty atributů, ale jenom určuje název a datový typ. Teprve až při vytvoření příslušné instance třídy jsou atributům přiřazeny skutečné hodnoty, které určují stav objektu. Metody slouží k přístupu a práci s atributy.

V UML jazyku je třída zobrazována jako obdélník, ve kterém je v horní části uveden název třídy a pod ním čarou oddělené skupiny atributů a metod.

Dědičnost
Základní prvek objektového programování je dědičnost. Umožňuje objektům sdílet jejich charakteristiky a zároveň určuje rozdíly mezi navzájem děděnými třídami. Třída využívající dědičnosti přebírá všechny atributy a metody děděné třídy a přidává svoje vlastní rozšíření.

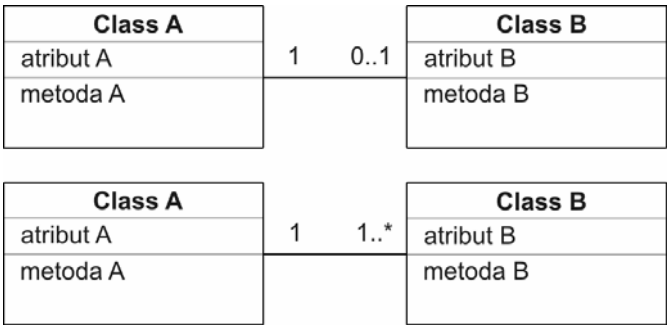
Polymorfizmus
Polymorfizmus určuje, že se objekt chová podle toho, jakého je typu. Máme například dvě třídy, obě dědí od stejné třídy. Obě třídy přetíží metodu definovanou v bázové třídě. Platí, že rodičovská třída může zastoupit své potomky. Polymorfizmus zaručí zavolání metody definované v třídě příslušného potomka.

Zapouzdření
Třetí z vlastností definujících objektové programování je zapouzdření. Zapouzdření určuje přístup k atributům třídy a manipulaci s nimi. K jednotlivým atributům objektu lze přistupovat pouze prostřednictvím metod, které objekt poskytuje. Tímto přístupem lze zabránit nekonzistenci dat.

2.7.2 Vztahy mezi třídami

V realizační části jsou zobrazeny jednotlivé navržené datové struktury pomocí UML diagramů. V těchto diagramech se používá grafická konvence, která udává vztahy mezi třídami. V následujícím textu si stručně popíšeme nejčastěji se vyskytující vztahy mezi třídami a jejich grafické znázornění.

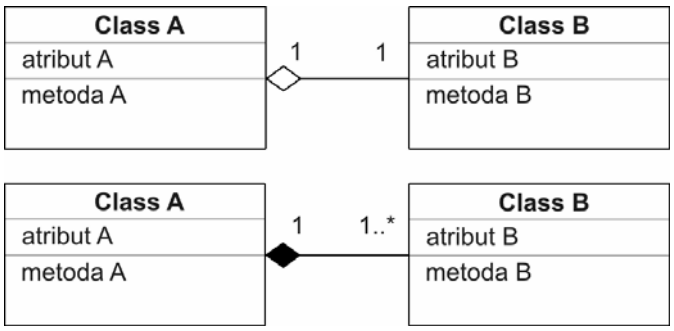
Asociace
Vztah typu asociace reprezentuje vztah mezi dvěma nebo více třídami. Asociace mezi dvěma objekty mohou být jednosměrné i obousměrné. S asociací může být sdruženo určité pravidlo, které může mít omezující nebo doplňující charakter. Násobnost asociace (udává se číslicemi nad vazbu) vyjadřuje poměr vztahu mezi objekty jednotlivých tříd. Asociace představuje obecnější vazbou nežli agregace. Na obrázku (Obrázek 2-13) je zobrazeno značení asociace v UML diagramech.



Obrázek 2-13: Asociace, varianty s různými násobnostmi

Agregace
Agregace je speciálním typem asociace. Agregace udává vlastnost, kdy jedna třída je součástí druhé. Jako příklad agregace si můžeme představit dvojici objektů kniha a stránka. Speciálním typem agregace je kompozice. Rozdíl mezi kompozicí a agregací

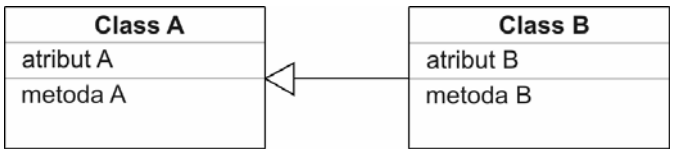
je v tom, že ve vztahu kompozice je jeden objekt závislý na druhém a sám o sobě nemůže existovat, kdežto v agregaci ano. Značení agregace a kompozice v UML je zobrazeno na obrázku (Obrázek 2-14).



Obrázek 2-14: V horní části je znázorněna agregace, ve spodní kompozice

Generalizace / specializace

Pod tímto označením se skrývá dědičnost. Pojem dědičnost byl vysvětlen v odstavci předcházející kapitoly. Obrázek (Obrázek 2-15) znázorňuje grafické značení této vazby. Směr prázdné šipky určuje nadřazenou třídu, tj. třídu, od které je děděno.

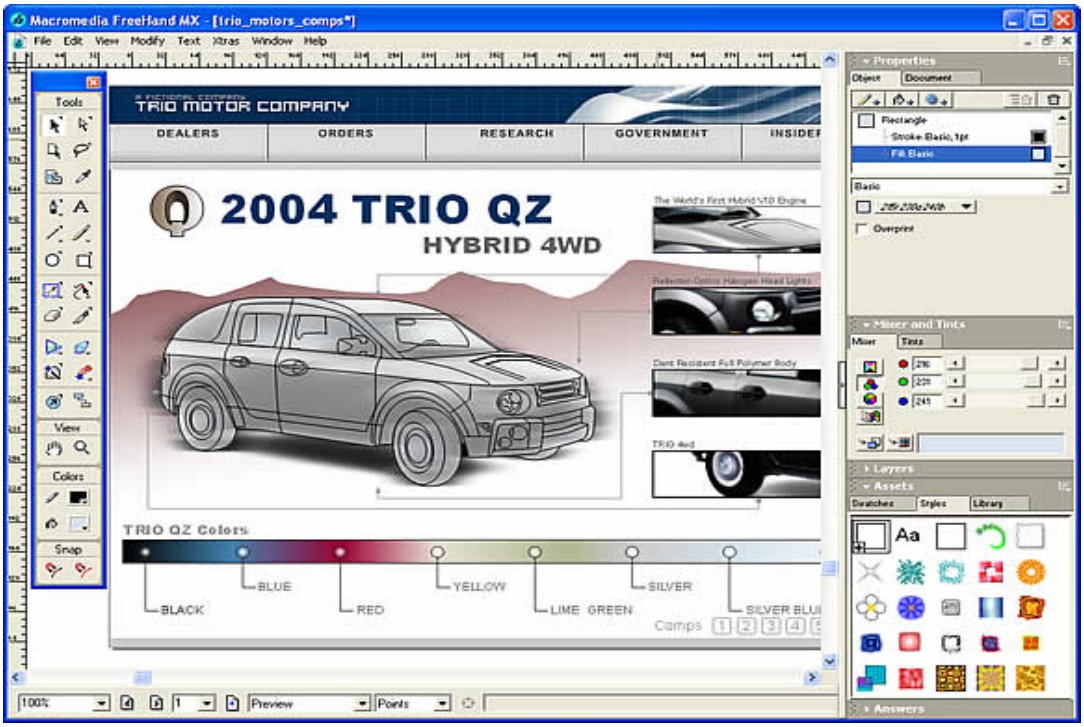


Obrázek 2-15: Generalizace

2.8 Přehled existujících řešení

2.8.1 Macromedia FreeHand

FreeHand je 2D grafický, vektorový, kreslící program, vyvíjený společností Macromedia, kterou v roce 2005 získala společnost Adobe Systems. V současné době je na trhu již jedenáctá verze tohoto kreslícího programu, která nese označení MX. FreeHand MX patří mezi jedny z nejlepších vektorových kreslících programů na trhu používaných nejen pro vektorovou grafiku. Umožňuje vytvářet složité grafické návrhy pro tisk, webové stránky, bitmapové obrázky a vícestránkovou montáž. Jde o robustní kreslící program určený pro profesionální využití. Uživatelské prostředí je podobné jako v ostatních aplikacích firmy Macromedia, základní ovládání je jednoduché a intuitivní.



Obrázek 2-16: Macromedia FreeHand, převzato z [Amos]

Při tvorbě ilustrace má uživatel k dispozici geometrické objekty, jako jsou body, úsečky, křivky, polygony, text atd. Na tyto objekty může uživatel aplikovat různé efekty a přitom může původní objekt nadále plně editovat. Parametry objektů lze přesně nastavovat v panelu vlastností. Jednotlivé panely lze různě kombinovat a umísťovat v libovolné rozložení v okně podle potřeb uživatele. K dispozici při návrhu jsou pravítka a vodící mřížka pro snadné umísťování objektů. Program podporuje barevné systémy CMYK, RGB, HLS a množství dalších.

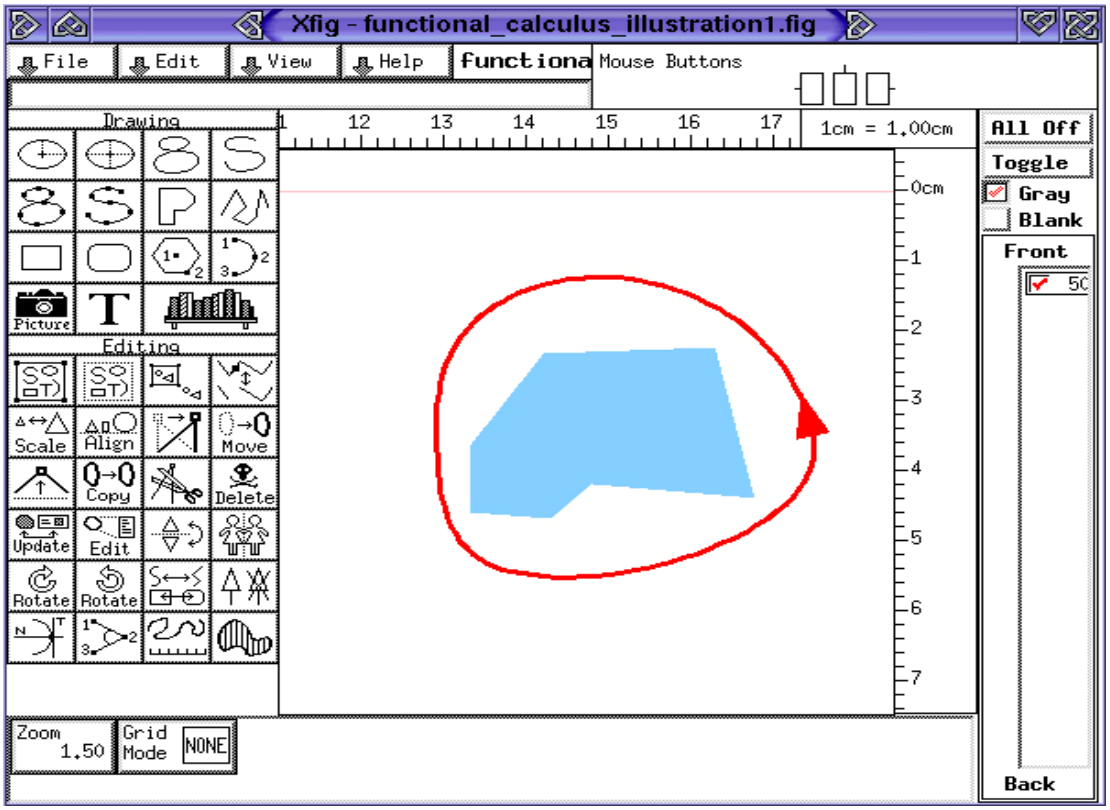
FreeHand nabízí velké množství nástrojů a funkcí. Jsou to funkce určené pro práci s textem, vektorovou gumu pro mazání částí křivek, nástroj Freeform na tvarování křivek, nástroj na kreslení vazebních čar mezi objekty pro tvorbu schématických

diagramů, nedestruktivní vektorové efekty (ohýbání, zkosení, stíny apod.), nástroj SnapToObject pro přichytávání k jiným objektům, vektorová průhlednost, nedestruktivní bitmapové efekty (stíny, orámování apod.), 3D efekty (vytažení do prostoru, 3D rotace, ...), nástroj pro práci s gradienty, kaligrafické pero a mnoho dalších. Více informací o tomto programu lze nalézt na internetových stránkách společnosti [Amos].

2.8.2 Xfig

Xfig je freeware 2D grafický program určený primárně pro X Windows systém a operační systémy založené na Unixu. Při tvorbě ilustrace má uživatel k dispozici standardní objekty, jako jsou kružnice, obdélník, úsečka, spline křivka a text, který podporuje i českou diakritiku. Program poskytuje podporu načítání bitmapových obrázků ve formátech .gif, .jpeg, .eps. Na rastrové obrázky lze aplikovat operace posunutí a změna měřítka, operace rotace není podporována.

Uživatelské prostředí je postaveno na knihovně Athena. Na některé uživatele může prostředí editoru působit zastarale a možná i ošklivě, ale to je věc názoru a vkusu. Program se ovládá myší.



Obrázek 2-17: Xfig, převzato z [Wiki]

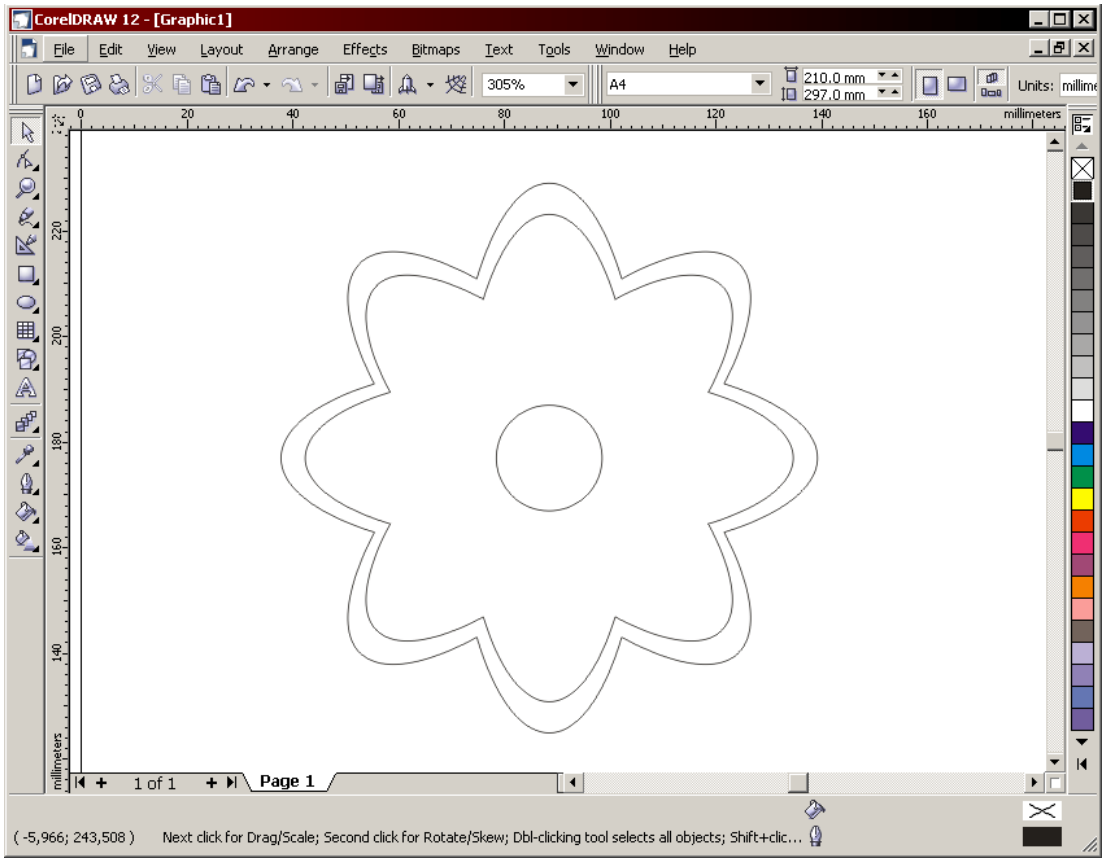
Xfig poskytuje funkci sloučení objektů do skupiny, změnu měřítka, rotaci, zarovnání objektů, vytvoření kopie, nástroj pro změnu parametrů objektů, vertikální a horizontální zrcadlení, převod elementu na spline křivku, přidání a odebrání bodu na spline křivku, určení obsahu polygonů a elips, určení délky a obvodu.

2.8.3 Corel DRAW

CorelDRAW je vektorový grafický editor vyvíjený společností Corel. CorelDRAW je součásti balíku grafických editorů označovaných jako Graphics Suite. V současné době je na trhu verze označená jako X3, pod tímto označením se skrývá již třináctá verze programu. CorelDRAW je primárně vyvíjen pro Microsoft Windows. V jisté omezené míře jsou uvolňovány verze pro operační systémy Linux, Mac OS a Mac OS X. Jistým omezení je míněno, že pro Linux je k dispozici CorelDraw verze 9 a pro Mac OS je k dispozici verze 11.

CorelDraw je určen jak pro profesionály, tak i pro úplné začátečníky. Jednotlivé verze CorelDRAW přináší vylepšené způsoby ovládání a editaci objektů. Corel DRAW podporuje široké množství funkcí usnadňující tvorbu. Poslední verze vylepšuje velké množství funkcí. Mezi vylepšené funkce například patří: editace uzlů, jejíž chování je nyní závislé na typech objektů, interaktivní stín, nástroj stín, perspektivní projekce, gradientní výplně a další.

Rozdílů mezi tímto editorem a konkurenčními editory je několik: tím prvním je balík editorů, které jsou dodávány v rámci aplikace CorelDRAW. Druhým rozdílem je, že se společně s editorem získá obrovské množství klipartů a kolekci fontů. Další výhodou je to, že CorelDRAW umožňuje editovat bitmapy. Při editaci bitmapy má uživatel k dispozici nástroje umožňující změnu jasu, kontrastu, vyvážení barev, změnu barevného prostoru a další množství efektů aplikovatelných na bitmapu.

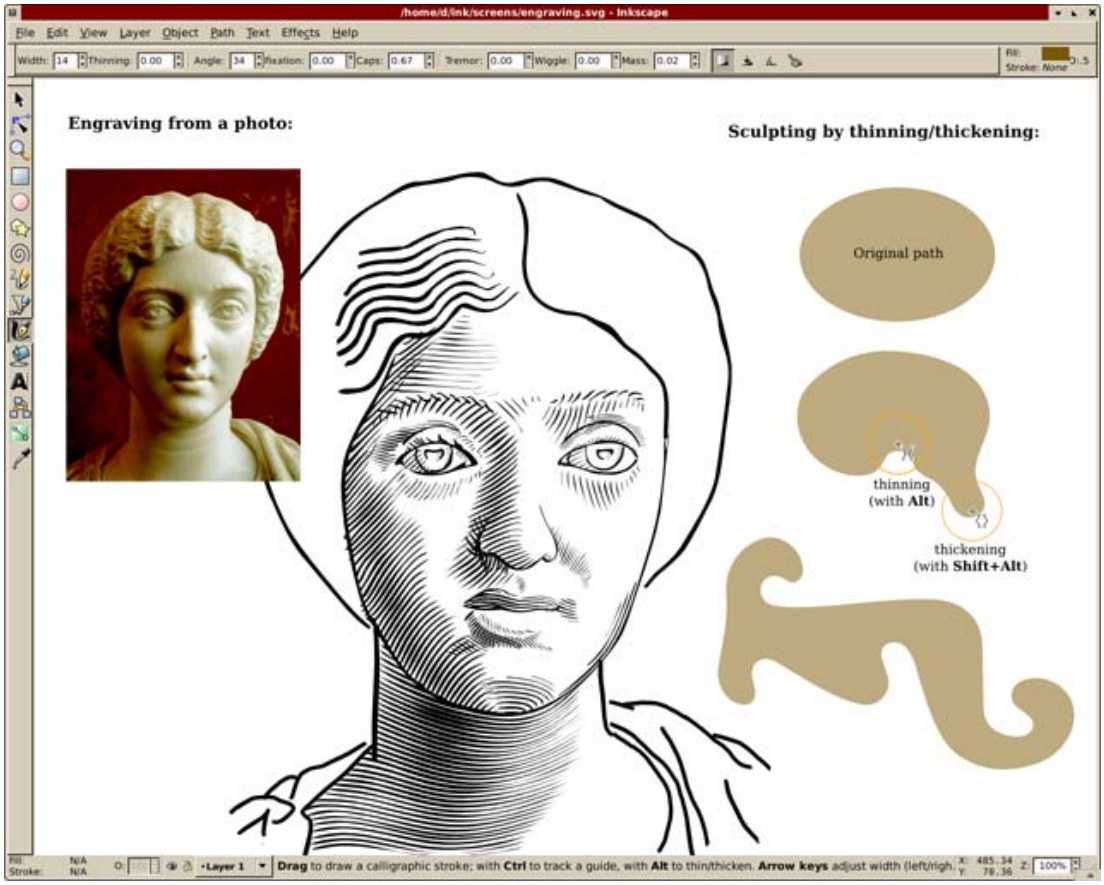


Obrázek 2-18: CorelDRAW

2.8.4 InkScape

Inkscape je vektorový grafický editor s otevřeným zdrojovým kódem. Cílem vývoje je vytvořit mocný a praktický grafický nástroj, který bude plně odpovídat standardům XML, SVG a CSS. Vývoj editoru začal v roce 2003 a to jako vedlejší projekt vektorového editoru Sodipodi, který je určen pro operační systém Linux. InkScape je primárně vyvíjen pro operační systém Linux, ovšem distribuce jsou uvolňovány i pro operační systémy Microsoft Windows, Mac OS X a různé operační systémy založené na Unixu. V současné době vývoj editoru stále pokračuje a editor již dospěl do verze 4.51. Stávající verze ještě zcela nepodporuje všechny standardy SVG a CSS, např. zatím nejsou podporovány filtry pro SVG, animace a SVG fonty.

Do výčtu podporovaných objektů patří: obdélník, elipsa, polygon, spirála, text, bézierova křivka a bitmapové obrázky. Mezi podporované nástroje sloužící k manipulaci s objekty patří: afinní transformace, klonování, vzory, seskupování, zarovnávání k vodícím čárám a mřížce, práce ve vrstvách. Podporovány jsou tyto barevné prostory: RGB, HSL, CMYK. Editor podporuje import grafiky z Postscriptu, EPS, JPEG, PNG a TIFF. Export je umožněn do rastrového formátu PNG a do některých vektorových formátů.



Obrázek 2-19: InkScape, převzato z [Inks]

2.8.5 Porovnání editorů

Na tomto místě by bylo vhodné provést porovnání editorů představených v předchozích kapitolách s námi vytvořeným editorem. Ovšem provést objektivní porovnání není možné a to z důvodu odlišného využití editorů. Námi vytvořený editor slouží k návrhu čipů, výše představené editory slouží k vytváření plakátů, reklamních letáků, log, kreseb atd. Těžko asi budeme hledat funkci 2D řezu, odhadu ceny v Corelu či FreeHandu. Důvod absence těchto funkcí je jasný, tyto funkce nemají v těchto editorech žádné uplatnění.

Společným rysem porovnávaných vektorových grafických editorů je způsob vytváření výsledného návrhu – kresby. Návrh začíná nakreslením základních primitiv, na která jsou aplikovány různé transformace a operace. Prostřednictvím těchto transformací a operací dochází k cíleným přeměnám původního objektu.

Funkce a nástroje poskytované vektorovými editory InkScape, CorelDRAW a FreeHand jsou do jisté míry stejné. Nedá se říct, který z těchto editorů je lepší, a který horší. Hodnocení editorů se bude lišit podle návyků uživatelů. Z mého hlediska je velmi zajímavý a výhodný editor InkScape a to z důvodu volné distribuce a množství poskytovaných funkcí.

3 Realizační část

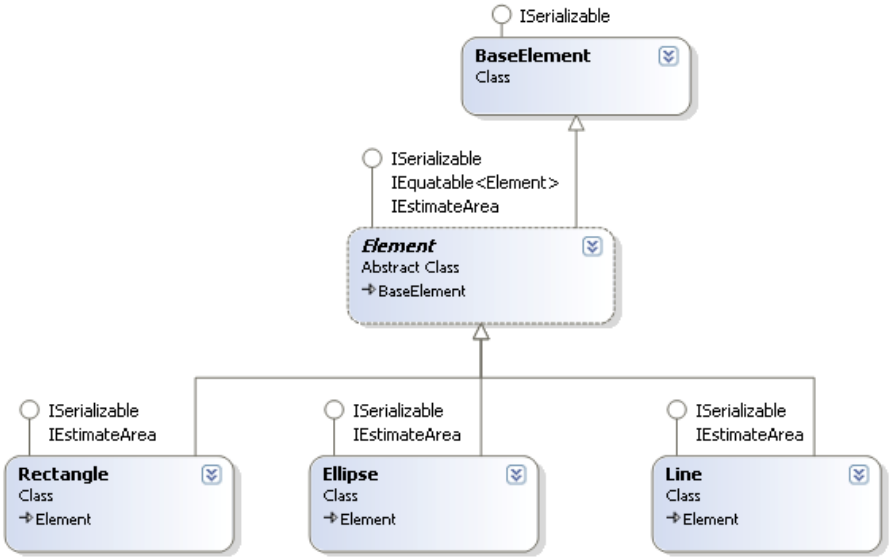
Tato část je věnována vlastnímu řešení zadání diplomové práce. Diplomová práce je součástí projektu FlashPoM. Cílem tohoto projektu bylo vytvoření 2D vektorového kreslícího programu určeného pro návrh čipů.

Významnou částí mé práce v rámci vývoje editoru byl návrh datových struktur, jejich reprezentace a manipulace s nimi. Kromě návrhu datových struktur jsem se věnoval i ostatním částem vývoje. Zabýval jsem se např. optimalizací vykreslování, odhadem ceny, validací vyrobitelnosti čipu, 3D zobrazením a jinými.

3.1 Základní datové třídy

3.1.1 Základní primitiva

Mezi základní vykreslovaná primitiva patří: obdélník (*Rectangle*), elipsa (*Ellipse*) a úsečka (*Line*). Na obrázku (Obrázek 3-1) je zobrazen UML diagram těchto tříd spolu s jejich rodičovskými třídami. Třída *BaseElement* zajišťuje svým potomkům atributy, které určují jednoznačnou identifikaci, časovou značku a index udávající pořadí při vykreslování. V následujícím textu jsou podrobněji popsány jednotlivé datové třídy. Vazby mezi třídami jsou znázorněny prostřednictvím UML diagramů.

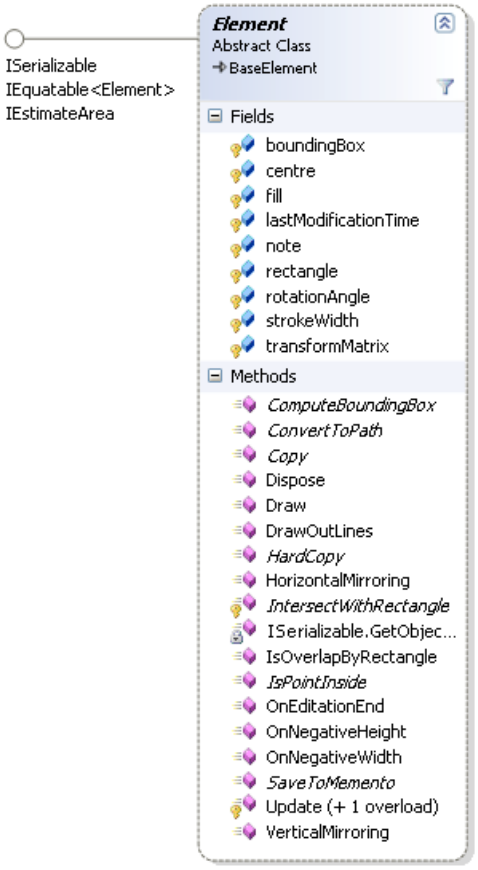


Obrázek 3-1: UML diagram základních elementů

Element

Třída *Element* je abstraktní třídou, která slouží jako bázová třída pro vykreslované objekty. Mezi potomky této třídy patří: *Rectangle*, *Ellipse*, *Line, Path, Group a TmpGroup*. Obrázek (Obrázek 3-2) zobrazuje UML diagram této třídy. Třída *Element* definuje abstraktní metody, které jsou potomci povinni implementovat. Abstraktní metody představují minimální množinu metod, které jsou potřebné pro manipulaci a vykreslení potomků třídy.

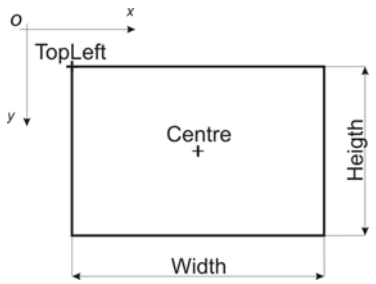
Obrázek (Obrázek 3-2) zobrazuje UML diagram třídy *Element*. Názvy jednotlivých metod a atributů vystihují význam a funkcionalitu, kterou metody a atributy poskytují.



Obrázek 3-2: UML diagram třídy Element

Třída *Element* definuje čtyři základní atributy, které jsou určeny na základě způsobu vykreslování primitiv v GDI+ (Obrázek 3-3). Těmito atributy jsou:

- souřadnice levého horního rohu
- rozměry elementu
- souřadnice středu, podle kterého je objekt otočen
- úhel rotace, který určuje natočení objektu



Obrázek 3-3: Základní atributy vykreslovaného objektu

Rectangle

Jak již název třídy napovídá, tato třída definuje objekty obdélníkového tvaru. Třída *Rectangle* je potomek třídy *Element* (Obrázek 3-1). Z implementačního pohledu se jedná o nejjednodušší třídu, která pouze přetěžuje jednotlivé abstraktní metody definované v rodičovské třídě. Ke svému popisu nepotřebuje přidat žádný atribut.

Ellipse

Tato třída definuje objekt elipsa resp. kružnice. Třída *Ellipse* je opět potomek třídy *Element* (Obrázek 3-1). Třída nerozšiřuje rodičovskou třídu o žádný atribut ani metodu. Rozšiřuje pouze vlastnosti (*Properties*) třídy a to přidáním vlastnosti popisující hlavní a vedlejší poloosu. Objekt elipsa je interně popsán obdélníkem.

Line

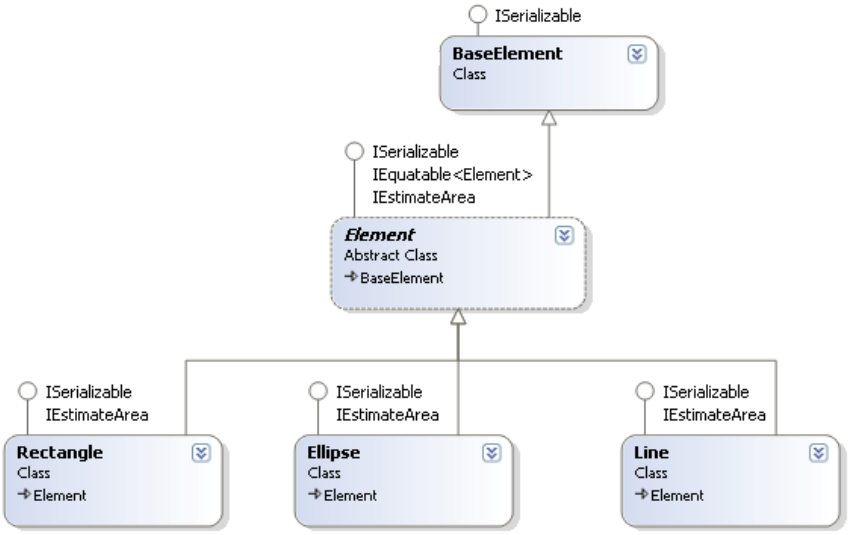
Třída *Line* představuje objekt úsečka. Třída *Line* je potomek třídy *Element* (Obrázek 3-1). Na rozdíl od předchozích tříd rozšiřuje tato třída své atributy o jeden. Tento atribut udává pozici počátečního bodu úsečky v obdélníku, prostřednictvím kterého je úsečka definována. Koncový bod úsečky je umístěn vždy v protilehlém rohu.

3.1.2 Rozšířená primitiva

Mezi rozšířené elementy patří primitiva: cesta (*Path*), dočasná skupina (*TmpGroup*) a skupina (*Group*). Obrázek (Obrázek 3-4) zobrazuje UML diagram těchto tříd. Rozšířené elementy nelze vytvářet přímo. Tyto elementy vznikají převedením dočasným nebo trvalým sloučením základních elementů. V následujícím textu si objasníme jednotlivé třídy a funkce, které poskytují.

Path

Třída *Path* je určena k provádění booleovských operací nad dvojicí základních elementů. Třída *Path* je potomkem třídy *Element* (Obrázek 3-4). Podporované booleovské operace jsou: sloučení, rozdíl a průnik. Operace jsou vždy prováděny nad dvojicí objektů typu *Path*. Základní vykreslovaná primitiva obsahují metodu, která převádí daný objekt na objekt typu *Path*. Implementací této třídy se ujal kolega Hladík. Bližší informace týkající se této třídy lze nalézt v jeho diplomové práci.



Obrázek 3-4: UML diagram rozšířených elementů

TmpGroup

Třída *TmpGroup* představuje dočasnou skupinu. Dočasná skupina je seskupení objektů, které je platné pouze po dobu editace tohoto elementu. V okamžiku ukončení editace dočasná skupina zaniká a jednotlivé objekty se stávají na sobě nezávislými. *TmpGroup* je opět potomkem třídy *Element* (Obrázek 3-4). Této vlastnosti je využito při manipulaci s objektem tohoto typu.

Jednotlivé parametry aplikované na dočasnou skupinu se přímo promítají na jednotlivé objekty zahrnuté v této skupině. Třída *TmpGroup* rozšiřuje atributy a metody své rodičovské třídy o atributy a metody sloužící k správě elementů zahrnutých v této skupině. Stav objektu *TmpGroup* je určen na základě stavu jednotlivých zahrnutých objektů.

TmpGroup, objekt jako takový, se z podstaty svého účelu neukládá do datových struktur. Účelem tohoto objektu je editace více elementů současně a zpříjemnění manipulace s více elementy.

Group

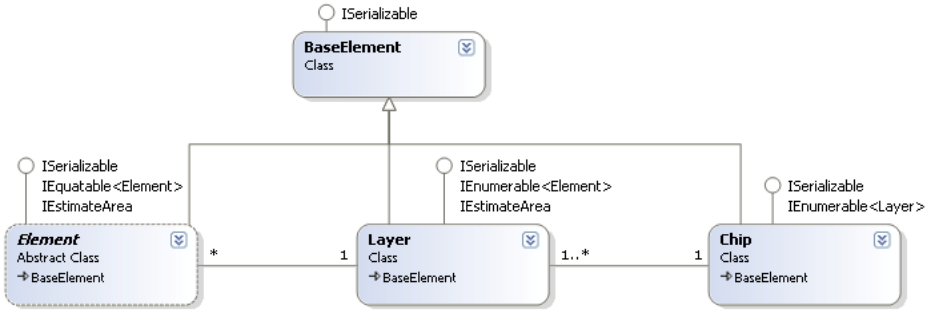
Třída *Group* slouží k trvalému seskupení objektů. Objekty zahrnuté do skupiny v této skupině přetrvávají po dobu její existence. Touto vlastností se třída *Group* liší od třídy *TmpGroup*. Z obrázku (Obrázek 3-4) patrné, že i tato třída je potomkem třídy *Element*.

Třída slouží jako kontejner pro jednotlivé objekty, které jsou do skupiny začleněny. Mezi objekty zahrnuté do této třídy mohou také patřit objekty typu *Group*. Dochází tak k vytvoření hierarchické stromové struktury.

Chování objektu *Group* je obdobný jako chování základních objektů. Odlišnost oproti základním objektům spočívá v tom, že daný atribut je aplikován na všechny objekty zahrnuté ve skupině. Další rozdíl je ve způsobu editace objektu tohoto typu. Objekty typu *Group* a *TmpGroup* umožňují pouze proporcionální změnu měřítka poměru stran. Důvodem je zachování poměrů velikostí elementů. Nedochází tak k deformaci objektů. Způsoby editace elementů jsou blíže popsány v části 3.2.

3.1.3 Struktura čipu

Datové struktury popisující navrhovaný čip jsou založeny na reálných postupech výroby čipu. Na základě uspořádání čipu a znalosti postupu výroby byly navrženy datové třídy *Layer* a *Chip*, které zachycují topologii čipu. Třída *Layer* představuje jednu vrstvu čipu, třída *Chip* představuje čip samotný. Na obrázku (Obrázek 3-5) je zobrazen UML diagram popisující navržené třídy, vzájemné vazby mezi těmito třídami a vykreslovanými elementy.



Obrázek 3-5: UML diagram popisující strukturu čipu

Layer

Tato třída definuje jednu vrstvu čipu. Vrstva je z hlediska výrobního procesu charakterizována materiálem. Z uživatelského pohledu je vrstva charakterizována jménem, barvou, parametrem udávajícím uzamknutí vrstvy a elementy, které jsou na vrstvě vykresleny. Elementy vrstvy jsou uloženy ve zvláštní struktuře, která umožňuje rychlou manipulaci s elementy. Uložením elementů se zabývá kapitola 3.1.4. Elementy jsou vykresleny barvou vrstvy. Tvar vrstvy je dán použitým substrátem. Pokud je vrstva označena jako negativní, znamená to, že plocha vymezená jednotlivými elementy zůstane součástí výsledné plochy vrstvy a okolní plocha bude odstraněna. V případě pozitivního typu vrstvy je tomu naopak, tj. plocha vymezená elementy bude odstraněna.

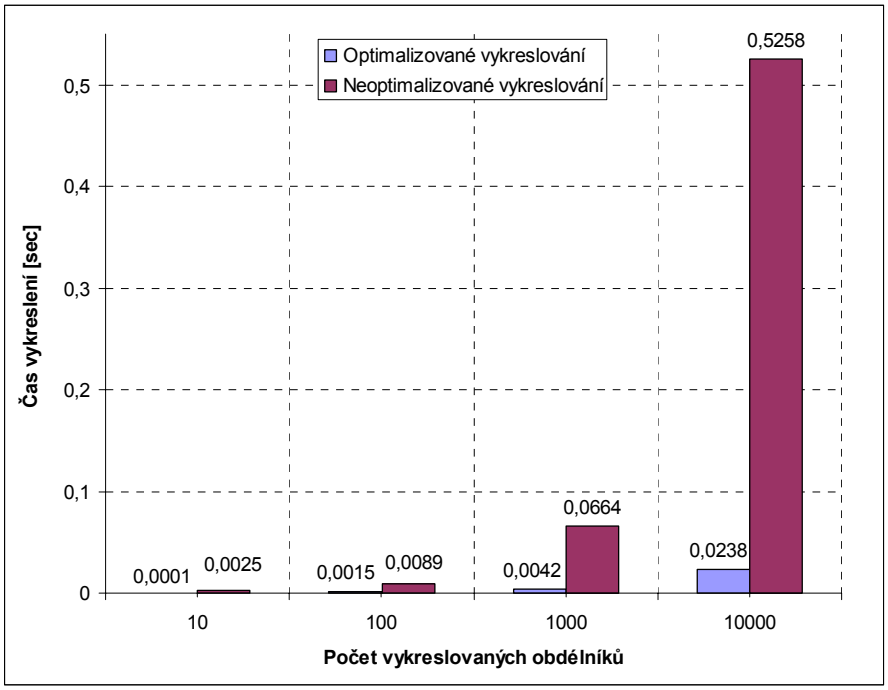
Jednotlivé vrstvy obsahují vlastní bitmapu (pod pojmem bitmapa rozumíme místo v paměti), do které jsou vykreslovány elementy náležící dané vrstvě. Výsledný vzhled čipu vznikne sloučením těchto bitmap do jedné výsledné bitmapy, která je zobrazena na návrhové ploše. Bitmapy jsou slučovány podle uspořádání vrstev na čipu. Tento způsob vykreslování přináší následující výhody:

- při práci s více vrstvami dochází k překreslování pouze té vrstvy, se kterou uživatel v daném okamžiku pracuje
- k překreslení vrstvy dochází pouze při změně atributu nějakého elementu, nebo při změně pohledu
- při označení některé vrstvy jako skryté dochází pouze k sloučení ostatních bitmap, elementy jednotlivých vrstev nemusí být vykreslovány

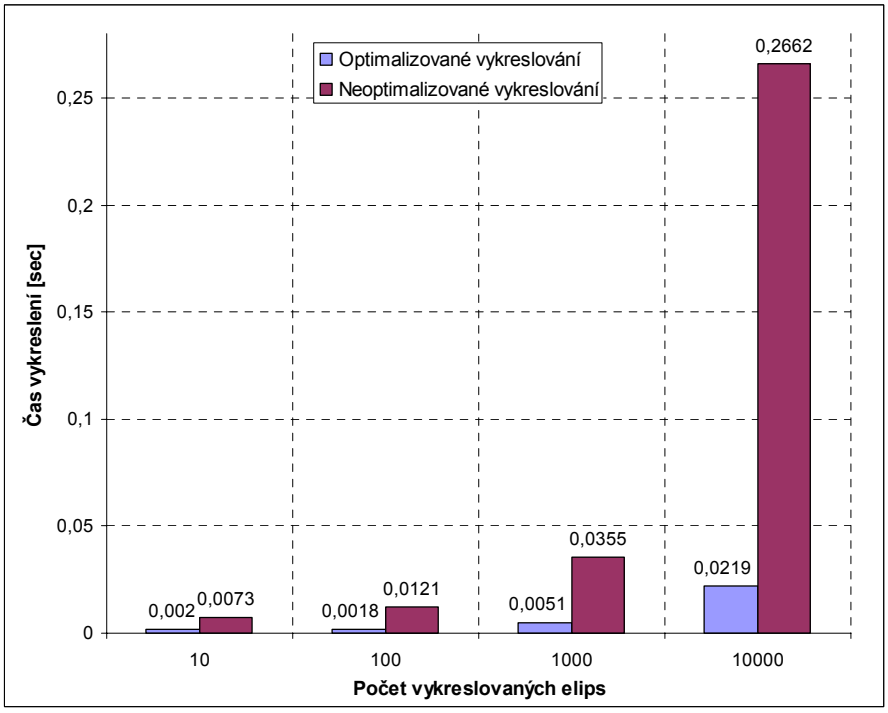
Vykreslování elementů v rámci jedné vrstvy je také optimalizováno. Optimalizace spočívá v překreslování pouze těch oblastí, ve kterých došlo k nějaké změně. Oblasti překreslení jsou určeny podle ohraničujících obdélníkových oblastí elementů. V těchto oblastech jsou znovu vykresleny pouze ty elementy, které částečně zasahují do těchto oblastí. V případech, kdy dochází ke změně měřítka vykreslování nebo

k posunu pracovní plochy, jsou vykreslovány pouze ty elementy, které zasahují do oblasti návrhové plochy.

Obrázky (Obrázek 3-6 a Obrázek 3-7) zobrazují dosažené urychlení. Naměřené výsledky byly získány tak, že na návrhové ploše byly zobrazeny uvedené počty elementů a jeden z elementů byl přesunut na jinou pozici. Optimalizované vykreslování odpovídá způsobu uvedenému v předchozím odstavci. Neoptimalizované vykreslování odpovídá vykreslení všech elementů.



Obrázek 3-6: Srovnání časů vykreslování obdélníků



Obrázek 3-7: Srovnání časů vykreslování elips

Chip

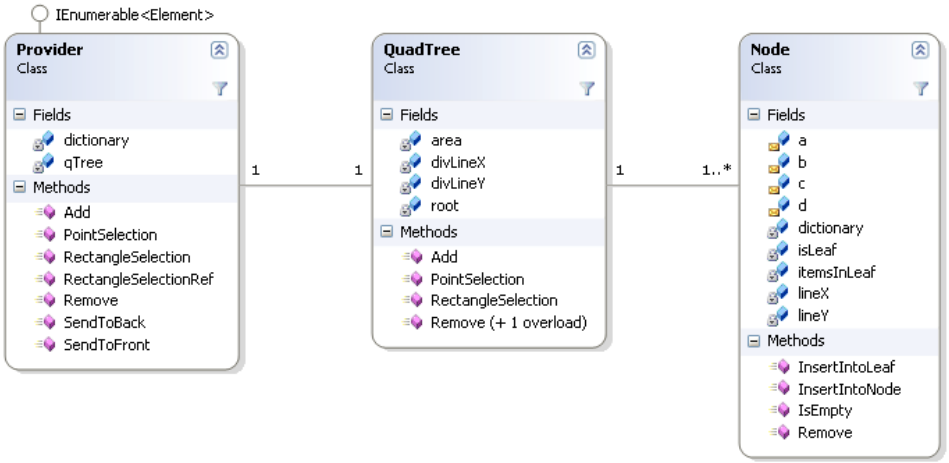
Třída *Chip* představuje navrhovaný čip. Hlavním úkolem této třídy je správa vrstev. Jednotlivé vrstvy jsou zde udržovány v pořadí, které odpovídá uspořádání vrstev na substrát. Tato třída slouží jako kontejner pro objekty popisující výsledný čip.

3.1.4 Vyhledávací struktura

Elementy náležící vrstvě jsou uloženy ve struktuře, která umožňuje vyhledávání elementů třemi různými způsoby:

1. vyhledávání elementu podle identifikačního čísla
2. vyhledávání element, který obsahuje bod o daných souřadnicích
3. vyhledávání elementů zcela obsažených nebo částečně protnutých obdélníkovou oblastí

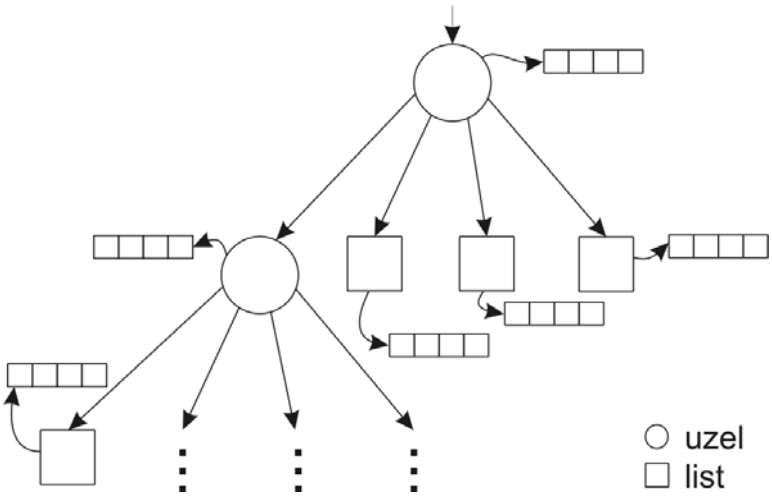
Vyhledávání elementů těmito způsoby zprostředkovává třída *Provider*. Na obrázku (Obrázek 3-8) je znázorněn UML diagram vyhledávací struktury. Struktury použité k vyhledávání jsou *hash* tabulka (*Dictionary*) a upravený kvadrantový strom (*QuadTree*).



Obrázek 3-8: UML diagram vyhledávací struktury

Hash tabulka slouží k vyhledávání elementů podle identifikačního čísla. Klíčem v *hash* tabulce je identifikační číslo elementu, které je jedinečné a přiřazené každému elementu při jeho vzniku.

Struktura použitá k vyhledávání elementů podle souřadnic bodu nebo podle obdélníkové oblasti vychází z kvadrantového stromu. Rozdíl oproti klasickému kvadrantovému stromu je, že pokud ohraničující oblast vkládaného elementu protíná dělící přímku daného uzlu, je tento element zařazen do seznamu tohoto uzlu. Schématicky je tato struktura znázorněna na obrázku (Obrázek 3-9).



Obrázek 3-9: Upravený kvadrantový strom

Struktura použitá pro uložení elementů v uzlech a listech stromu je *hash* tabulka. *Hash* tabulka je použita pro rychlost vkládání a odebírání elementů.

Elementy jsou do stromu vkládány na základě osově zarovnané ohraničující obdélníkové oblasti. Ve stromu jsou elementy uloženy v uzlu nebo v listu. V uzlu jsou uloženy elementy, jejichž ohraničující oblast je protnuta některou z dělících přímek daného uzlu. V listu jsou uloženy elementy náležící do oblasti vymezené daným listem. Každý list může obsahovat nejvýše *n* elementů (zpočátku 4). Po překročení maximálního počtu elementů obsažených v listu dojde k přeměně listu na uzel a elementy obsažené v listu jsou znovu zatříděny do této části stromu, ale s tím rozdílem, že počet elementů v listech je nyní navýšen na *n + 1*.

Každý uzel obsahuje čtyři listy. V každém uzlu jsou definovány dvě dělící přímky, které rozdělují danou oblast uzlu na čtyři stejně velké oblasti. Každá oblast odpovídá jednomu listu daného uzlu. Do listů jsou ukládány elementy spadající do této oblasti.

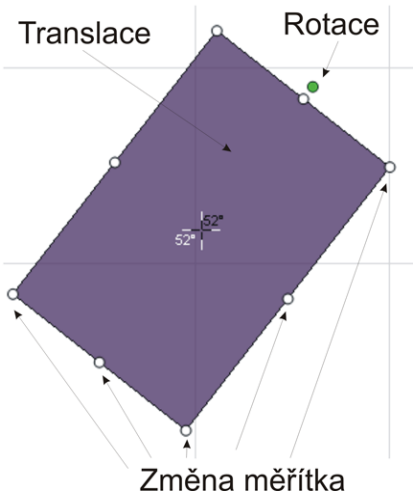
Při vyhledávání podle souřadnic bodu je vybrán element, který obsahuje daný bod a současně má nejvyšší index pořadí vykreslování. Při vyhledávání obdélníkovým regionem jsou vybrány všechny elementy, které jsou částečně protnuté nebo zcela obsažené v tomto regionu.

3.2 Editace

Editace slouží k vizuálním úpravám elementů. Editační mód (pojem editační mód označuje dobu, po kterou lze element editovat) elementu je indikován zobrazením editačních symbolů a zobrazením atributů elementu v panelu vlastností. Tento mód umožňuje uživateli měnit jednotlivé parametry elementu a aplikovat na element transformace. Výběr transformace je prováděn stlačením levého tlačítka myši v oblasti zastupující danou transformaci. K ukončení transformace dojde v okamžiku uvolnění tlačítka.

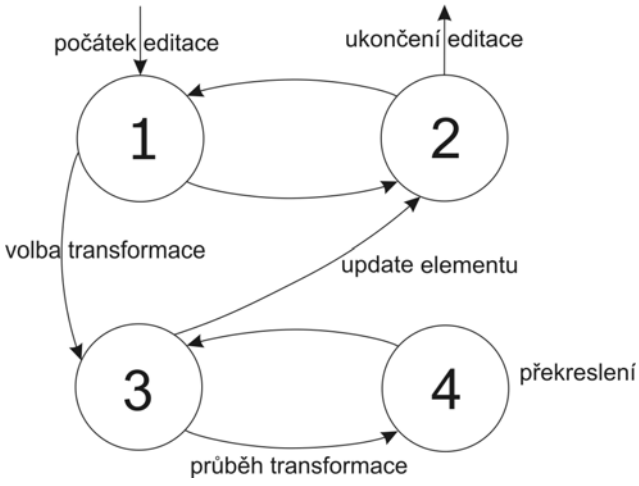
V editačním módu lze na editovaný element aplikovat tyto transformace: translace, rotace a změna měřítka. K výběru translace dochází ve vnitřní oblasti elementu, která určuje její výběr. Tato oblast je na obrázku (Obrázek 3-10) tmavě vyplněná. Rotace elementu je vybrána v oblasti definované vyplněným kruhovým symbolem, který je

zobrazen na obrázku (Obrázek 3-10). Změna měřítka je zastoupena osmi kruhovými nevyplněnými symboly umístěnými kolem editovaného elementu. Každý z osmi symbolů určuje jiný způsob změny měřítka.



Obrázek 3-10: Výběr transformací

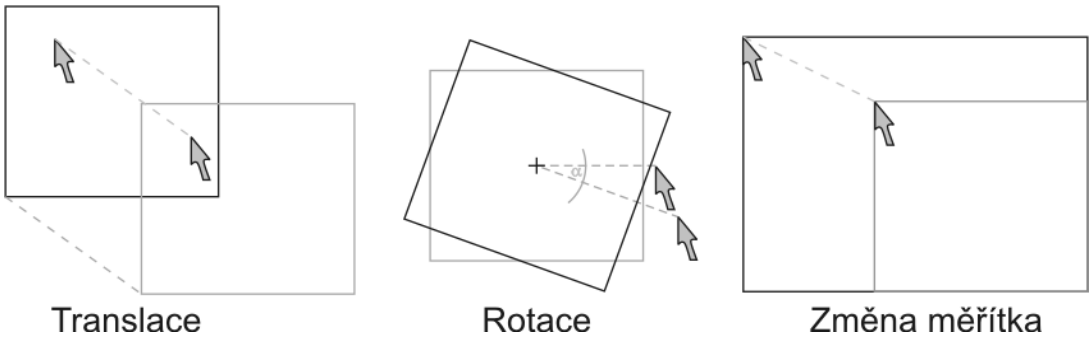
Editaci elementu lze aktivovat prostřednictvím ikony umístěné v panelu nástrojů nebo prostřednictvím položky kontextového menu. Na obrázku (Obrázek 3-11) je zobrazen stavový diagram popisující průběh editace. Stav *(1)* označuje výběr elementu určeného k editaci a tento stav je indikován zobrazením zástupných symbolů okolo editovaného elementu (Obrázek 3-10). Z tohoto stavu lze přejít do stavu *(2)* nebo *(3)*. K přechodu do stavu *(3)* dochází v případě výběru jedné z transformací, v ostatních případech následuje přechod do stavu *(2)*, který označuje ukončení editace. Ve stavu *(3)* je na objekt aplikována vybraná transformace. Následuje překreslení elementu *(4)* a přechod zpět do stavu *(3)*. Přechody mezi těmito stavy se opakují po celou dobu transformace, která je ukončena v okamžiku uvolnění stisknutého tlačítka. V tento okamžik je také provedena aktualizace elementu uloženého v datových strukturách a editace se vrátí do stavu *(1)*.



Obrázek 3-11: Stavový diagram průběhu editace

Transformace jsou vyhodnocovány na základě vzdálenosti, kterou urazil kurzor myši od okamžiku stisknutí tlačítka (Obrázek 3-12). Tato vzdálenost udává u translace relativní posunutí elementu. Na základě vzdálenosti je určen úhel rotace. V případě

změny měřítka udává vzdálenost relativní posun příslušné hrany nebo vrcholu editovaného elementu oproti původní hraně nebo vrcholu elementu.



Obrázek 3-12: Vyhodnocení transformací

3.3 Historie

Jedním z požadavků kladených na editor je uchovávání historie změn aplikovaných na objekty. Historie zachycuje změny objektů. V historii jsou uchovávány tyto objekty: potomci třídy *Element*, třída *Layer* a třída *Chip*. Do historie jsou ukládány kopie jednotlivých objektů. Události, které vedou k vložení objektu do historie jsou:

- přidání nového objektu
- odstranění objektu
- úprava objektu

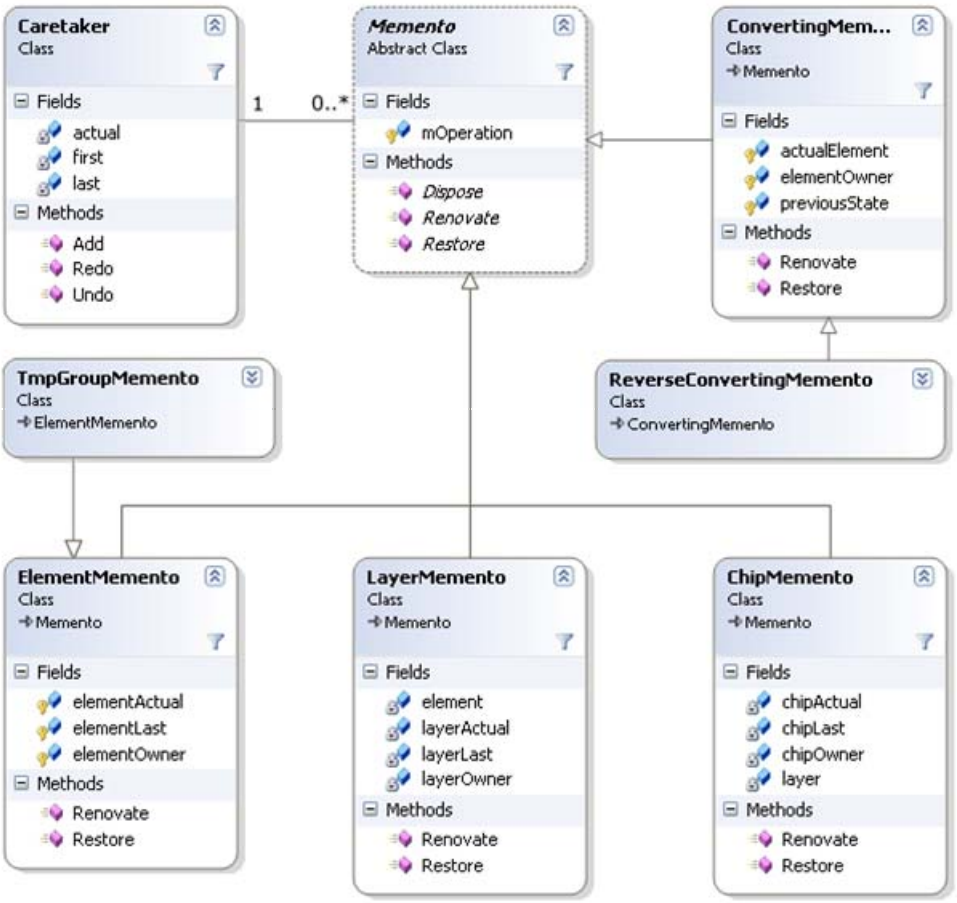
Hlavní funkcí historie je obnovení předcházejících stavů objektů (*Undo*), případně nastavení následujících stavů objektů (*Redo*). Obnovení je vždy vztaženo k aktuálnímu stavu. Počet kroků zpět je standardně nastaven na 24 kroků, tuto hodnotu lze změnit v nastavení projektu.

Kopie objektů vkládané do historie se liší podle typu objektu. V případě potomků třídy *Element* se jedná o přesné kopie objektů. U objektů typu *Layer* a *Chip* je tomu jinak. U těchto objektů jsou v historii uchované jen některé atributy, a to takové, které může uživatel měnit v panelu vlastností. Nejsou tedy uchovávány celé datové struktury obsahující elementy náležící vrstvě, popř. vrstvy náležící čipu. Důvodem je úspora místa zabraného objekty v paměti.

Obrázek (Obrázek 3-13) zobrazuje kompletní UML diagram popisující hierarchii tříd historie. Správu uložených objektů a komunikaci s okolím zajišťuje třída *Caretaker*. Manipulace s jednotlivými prvky historie je prováděna prostřednictvím bázové třídy *Memento*. V této třídě jsou definovány abstraktní metody *Renovate()* a *Restore()*, které mění aktuální stav objektu na předchozí nebo následující stav.

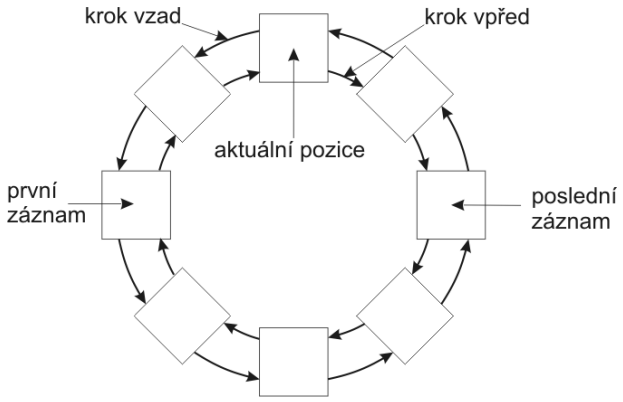
Hierarchické uspořádání tříd historie vychází z uspořádání tříd používaných pro čip. Na nejnižší úrovni se nachází třída *ElementMemento*. Tato třída zachycuje změnu stavu objektu typu *Element*. Přidání, nebo odebrání elementu souvisí s vrstvou, protože element je přidáván, nebo odebírán z vrstvy. Přidáním nebo odebráním elementu dojde tedy ke změně stavu vrstvy. Změna stavu vrstvy souvisí s třídou *LayerMemento*, ovšem přidání vrstvy souvisí s třídou *ChipMemento*. Třída

ConvertingMemento je určena k obnovení konverze objektů na jiný objekt, např. při booleovských operacích.



Obrázek 3-13: UML diagram historie

Struktura použitá pro uložení jednotlivých kroků historie je obousměrný kruhový seznam (Obrázek 3-11), reprezentovaný jednorozměrným polem. Pokud je vkládán objekt do historie a historie již obsahuje maximální počet objektů, je tento objekt vložen na místo prvního objektu. Při provedení několika kroků zpět a následném vložení nového objektu do historie dojde ke ztrátě kroků, které se nacházejí před aktuální pozicí.



Obrázek 3-14: Datová struktura pro uchovávání historie objektů

3.4 Komunikace s GUI

Komunikaci mezi GUI (grafické uživatelské rozhraní) a datovými strukturami zajišťuje třída *Mediator*. Hlavním úkolem této třídy je zprostředkovat funkcionalitu interních tříd, které jsou okolním třídám skryté. Okolní třídy nemají možnost přistupovat k datovým třídám a objektům uloženým v interních datových strukturách. Pokud tyto třídy manipulují s objekty používanými interně, pak manipulují s kopiemi těchto objektů. Na základě těchto kopií jsou aktualizovány originální objekty uchovávané uvnitř. Tento přístup přináší tyto výhody:

- zamezuje nekonzistenci datových objektů
- umožňuje provádět kontroly objektů
- umožňuje vytváření historie objektů
- poskytuje funkcionalitu bez odhalení způsobu interního řešení

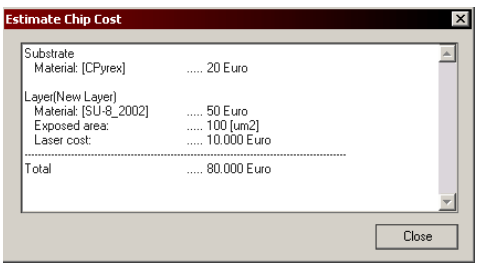
Jednou z nevýhod tohoto přístupu je právě ono vytváření kopií. Vytváření kopií pro manipulaci s elementy zabírá čas procesoru a kopie zabírají místo v paměti. I přes výše zmiňované nedostatky je použitý způsob komunikace s GUI výhodný, zvláště při uchovávání historie.

3.5 Odhad ceny

Tato kapitola se zabývá odhadem výrobní ceny navrženého čipu. Výsledná cena je určena na základě vzorců uvedených v části zabývající se dokumentem specifikace požadavků (2.3.4 Systémové požadavky).

Cena čipu je dána součtem ceny jednotlivých částí čipu a ceny práce laseru. Laser je použit při procesu vytváření schématu. Cena práce laseru je dána plochou elementů, tj. plochu, kterou musí laser osvítit. U vyplněných elementů je obsah určen na základě příslušného typu objektu. Obsah nevyplněných objektů je odhadnut na základě délky střední čáry elementu a tloušťky čáry tak, že dojde k vynásobení obvodu tloušťkou.

Na obrázku (Obrázek 3-15) je zobrazeno dialogové okno, ve kterém je uvedena výsledná cena čipu společně s cenami jednotlivých součástí čipu.



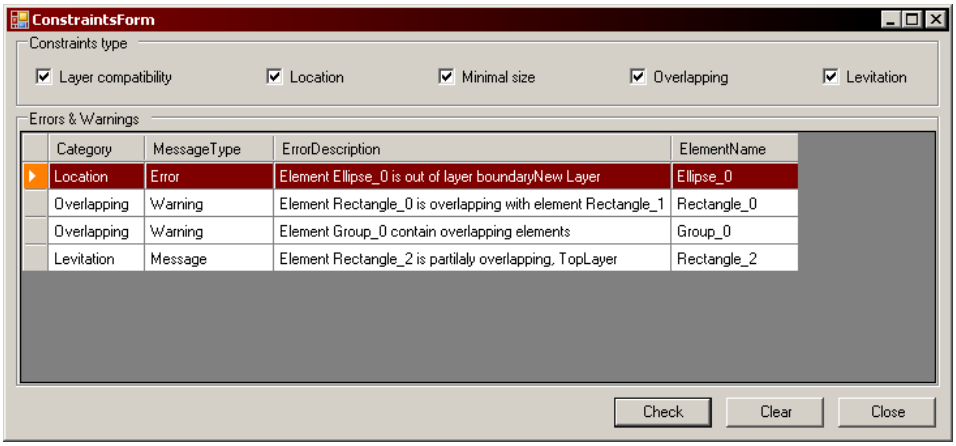
Obrázek 3-15: Dialogové okno zobrazující cenu čipu

3.6 Validace

Tato kapitola se zabývá kontrolou vyrobitelnosti čipu. Kontrola je prováděna z důvodu odhalení chyb, kterých se dopustil uživatel při návrhu. Chyby v návrhu mohou vést k nefunkčnosti čipu. Cílem validace je upozornit uživatele na nedostatky a chyby, kterých se dopustil při návrhu.

Formulář *ConstraintsForm* (Obrázek 3-16) zpřístupňuje funkce provádějící jednotlivé kontroly. Kontrolovány jsou tyto vlastnosti:

1. kompatibilita sousedních vrstev
2. umístění elementu uvnitř vrstvy
3. minimální velikost elementu
4. překrývání elementů v rámci jedné vrstvy
5. návaznost elementů v sousedních vrstvách



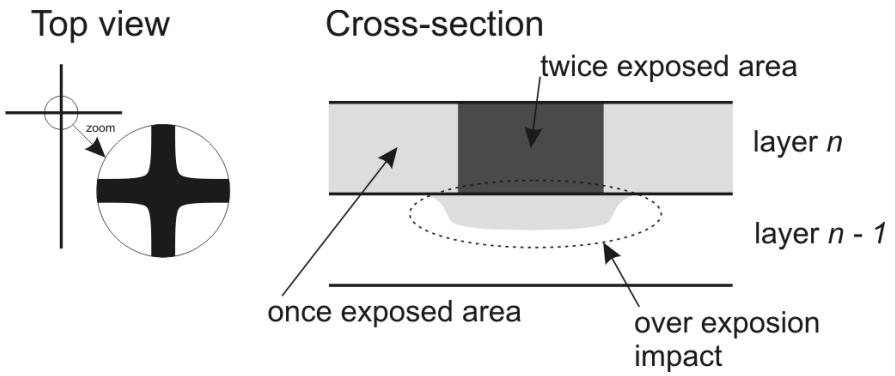
Obrázek 3-16: Validace čipu

Ad 1) Kompatibilita sousedních vrstev je vyhodnocena na základě tabulky, která určuje vzájemně kompatibilní vrstvy. Tabulka kompatibilních vrstev je sestavena výrobcem čipu na základě chemických vlastností materiálů vrstev.

Ad 2) Kontrola umístění elementů testuje pozice elementů vzhledem k hranicím vrstvy. Cílem kontroly je upozornit na elementy, které se nacházejí zcela za hranicí vrstvy, nebo ji alespoň částečně přesahují.

Ad 3) Při této kontrole je testována velikost elementů. Minimální velikost elementu je dána rozlišením laseru.

Ad 4) Vzájemné překrývání elementů způsobí vícenásobnou expozici. Vícenásobnou expozicí dojde k ovlivnění blízkého okolí daného místa a hlavně k ovlivnění rezistu vrstvy nacházející se pod aktuální vrstvou. Ve spodní vrstvě mohou vzniknout vzory, které v této vrstvě být nemají. Problém vícenásobné expozice je schématicky zobrazen na obrázku (Obrázek 3-17).



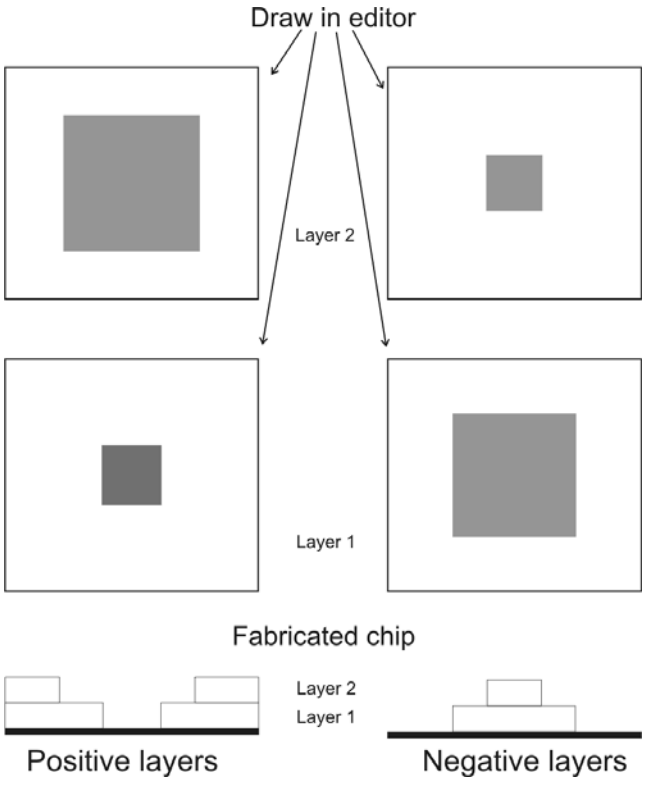
Obrázek 3-17: Negativní dopady vícenásobné expozice

Ad 5) Cílem této kontroly je zajistit návaznost elementů, které se nacházejí v sousedících vrstvách. Nesmí se stát, že se některý element vznášel ve vzduchu. Tato kontrola je prováděna pouze v případech, ve kterých se čip skládá z více vrstev. Kontrola vyplývá z technologických požadavků výroby čipu laserem, kdy je více vrstev vyráběno současně, tj. tvar vrchní vrstvy musí být takový, aby mohla být vyrobena spodní vrstva.

Podmínky, které určují jestli je dané uspořádání elementů správné, či nikoliv, jsou závislé na typu vrstvy. Vrstvy jsou dvojího typu:

- pozitivní vrstvy
- negativní vrstvy

Na obrázku (Obrázek 3-18) jsou v dolní části zobrazeny průřezy čipů odpovídající různým typů vrstev. Nad nimi jsou zobrazeny odpovídající návrhy vytvořené v editoru.



Obrázek 3-18: Vzhled čipu

Pokud je vrstva negativního typu, znamená to, že nakreslený element je součástí výsledné plochy vrstvy. Výsledná plocha vrstvy je tedy dána plochou jednotlivých elementů nacházejících se na této vrstvě. Při výrobě této vrstvy musí být tyto plochy rezistu osvíceny laserem. Pozitivní typ vrstvy znamená, že nakreslené elementy nejsou součástí výsledné vrstvy. Pozitivní vrstvy jsou tedy opakem negativních vrstev. Při výrobě takovéto vrstvy jsou osvíceny zbývající plochy rezistu.

Při vyhodnocování návaznosti elementů mohou nastat následující případy:

1. element z horní vrstvy protíná hraniční oblast elementu ze spodní vrstvy
2. element z horní vrstvy se nachází uvnitř elementu ze spodní vrstvy
3. element ze spodní vrstvy se nachází uvnitř elementu horní vrstvy
4. element z horní vrstvy leží mimo element ze spodní vrstvy

Vyhodnocení případů 2 a 3 je závislé na typu vrstvy. Zbývající případy lze vyhodnotit nezávisle na typu vrstvy. Případ 1 je vyhodnocen jako chybný, element z horní vrstvy leží pouze částečně na elementu spodní vrstvy. V případě 4 nelze učinit žádný závěr. Pro negativní vrstvy je případ 2 vyhodnocen jako správný, pro pozitivní vrstvy je vyhodnocen jako nesprávný. V případě pozitivní vrstvy nelze požitou technologií výroby vytvořit spodní vrstvu. Případ 3 je pro negativní vrstvu vyhodnocen jako chybný. Horní vrstva přesahuje hranici spodní vrstvy. Pro pozitivní vrstvu je případ 3 vyhodnocen jak správný.

Algoritmus vyhodnocení návaznosti elementů je následující: pro každý element spodní vrstvy jsou vybrány pouze ty elementy horní vrstvy, které se nacházejí v obdélníkové oblasti definované spodním elementem. Pro tyto vybrané elementy je vyhodnocena návaznost podle pravidel uvedených výše.

3.7 3D vizualizace

3D vizualizace slouží k prostorovému zobrazení navrhnutého čipu. Vizualizace umožňuje interaktivní prohlížení navrhnutého čipu. Při prohlížení má uživatel možnost určovat vrstvy, které budou zobrazeny, dále má možnost zobrazovat pomocné objekty sloužící k usnadnění orientace a možnost ukládání zobrazení do souboru.

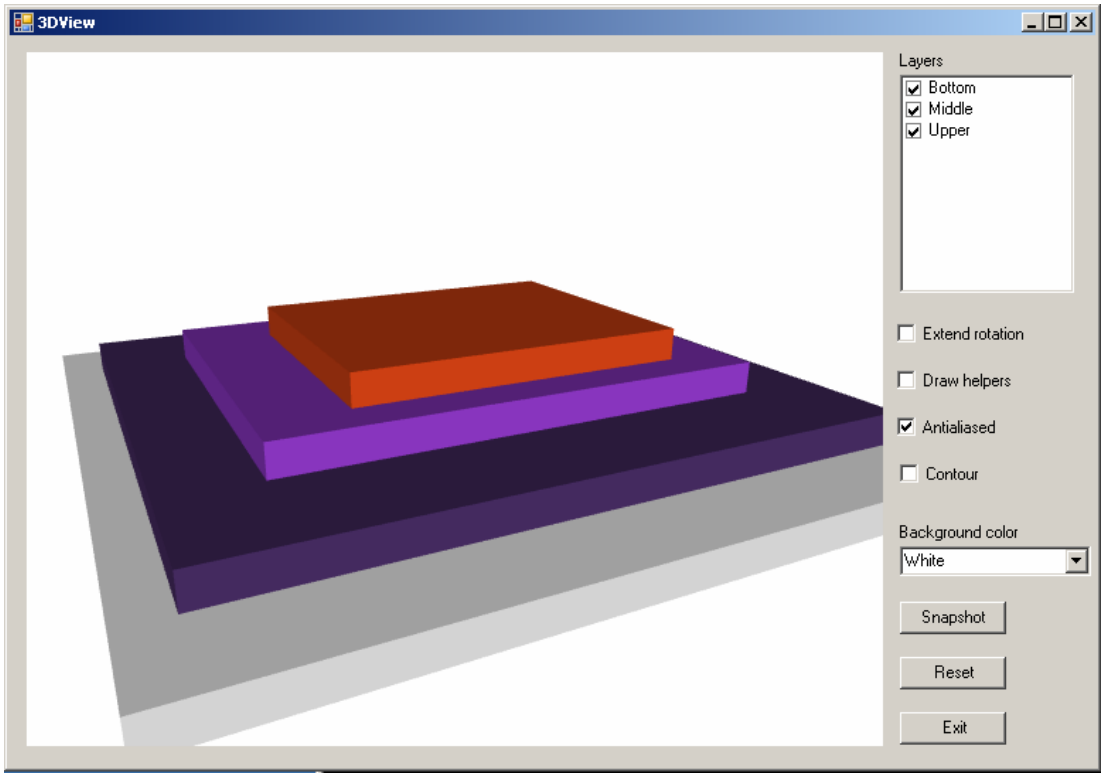
K akceleraci 3D zobrazení je použito rozhraní Direct3D 9.0c. Jednotlivé části čipu jsou uloženy v *.X souborech. Popis *.X souboru lze nalézt v diplomové práci kolegy Hladíka [Hlad].

Každá vrstva je rozdělena na tři části:

- vrchní uzávěr vrstvy
- obvodové stěny
- vrchní uzávěry elementů vrstvy

Rozdělení na tyto tři části je z důvodu usnadnění triangulace vrstvy. Jednodušší je vytvořit triangulaci dílčích elementů, nežli triangulaci celé vrstvy a to zvláště

v případech pozitivních vrstev. Popis triangulace jednotlivých elementů je popsán v [Hlad]. K vykreslení výsledného tvaru vrstvy je použita metod, která spočívá v použití stencil bufferu[1] při vykreslování. Nejdříve jsou vykresleny obvodové stěny elementů vrstvy. Následně jsou do stencil bufferu vykresleny uzávěry elementů dané vrstvy. Vykreslení uzávěru vrstvy se řídí informacemi ve stencil bufferu. Na základě informací uložených ve stencil bufferu je rozhodnuto, zda bude hodnota na daný pixel zapsána či nikoliv. Pokud tedy chceme vykreslit pozitivní vrstvu, zapíšeme do stencil bufferu hodnotu 1 na pozice pixelů, na kterých se nachází uzávěry elementů. Při vykreslování uzávěru vrstvy zapíšeme hodnotu pouze na ty pozice pixelů, které mají hodnotu 0 ve stencil bufferu. Tímto způsobem vykreslíme pozitivní vrstvu bez nutnosti triangulace celé plochy vrstvy. Podobný způsob je použit při vykreslování nevyplněných elementů.



Obrázek 3-19: 3D vizualizace

[1] Stencil buffer – paměťová oblast na grafické kartě do které lze zapisovat celočíselné hodnoty, velikost je určena rozlišením vykreslované oblasti

4 Závěr

Diplomové práce je součástí mezinárodního projektu FlashPoM. V rámci tohoto projektu byl vytvořen grafický editor sloužící k návrhu čipů. Editor obsahuje všechny požadované nástroje a funkce. V současné době probíhá závěrečné testování a příprava odevzdání finální verze.

V průběhu vývoje docházelo k úpravám a rozšířením původního návrhu aplikace. Úpravy návrhu byly nejčastěji způsobené změnami požadavků uživatele. Rozšíření návrhu byla nejčastěji způsobena přidávání nových funkcí nebo opomenutím zahrnutí funkcionality. Cílem původního návrhu bylo vytvořit na sobě nezávislé vrstvy aplikace, které by mezi sebou komunikovali prostřednictvím definovaného rozhranní. Tento cíl byl dodržen a výhody tohoto návrhu se projevili při rozšiřování, úpravách a ladění aplikace. Výsledná aplikace obsahuje přes 300 tříd zajišťujících požadovanou funkcionalitu.

V průběhu vývoje jsme narazili na problémy různého druhu. Jednalo se o problémy vznikající omezenou přesností datových typů, problémy související s komunikací se zadavateli projektu, nefunkčnosti použitých volně dostupných komponent a nebo související s odhalením nedostatků v specifikovaných požadavcích. Všechny tyto problémy byly v průběhu vývoje vyřešeny. Některé z těchto problémů byly vyřešeny drobnou úpravou návrhu, jiné vyžadovali přepracování návrhu nebo zvolení jiného přístupu k danému problémů.

Diplomová práce pro mě byla velkým přínosem. Získal jsem praktické zkušenosti související s vývojem rozsáhlého projektu. Vyzkoušel jsem si práci ve vývojovém týmu, ve kterém je výsledek závislý na vzájemné spolupráci všech členech týmu.

Alfa verze byla předána ke dni 20.3.2007. Odhalené funkční nedostatky byly na základě těchto podkladů upraveny. Odevzdání finální verze je plánováno na 5.6.2007. V období mezi těmito termíny probíhalo průběžné testování jednotlivých verzí.

Obsah CD ROMu je popsán v příloze Příloha B.

Literatura[1]

[Amos] *Macromedia FreeHand*,
[WWW]
http://www.amsoft.cz/Produkty/Adobe/freehand/pictures/win_xp_lg.jpg,
[2007-05-16]

[Clav0] E. Claverol-Tinturé, M. Ghirardi, F. Fiumarara , X. Rosell, J. Cabestany: *Multisite recording of extracellular potentials produced by micro channel-confined neurons in vitro*, TBME-00477-2005

[Clav1] E. Claverol-Tinturé, J. Cabestany, X. Rosell: *Multielectrode arrays with elastomeric microstructure overlays for extracellular recordings from patterned neurons*, J.Neural Eng. 2 (2005) L1-L7

[Hlad] Vojtěch Hladík, *Diplomová práce – 2D editor pro návrh topologie čipu – projekt FlashPoM – 1*, Plzeň, 2007

[Inks] *InkScape*,
[WWW] http://www.inkscape.org/screenshots/gallery/inkscape-0.46-engraving2.png, [2007-05-16]

[Pal] P. Paleta: *Co programátory ve škole neučí aneb Softwarové inženýrství v reálné praxi*. Computer Press, 2003, ISBN 80-251-0073-1

[Rera] *MATEO*,
[WWW] http://www.rera.cz/index.php?documentID=154, [2007-05-16]

[Rich] J. Richter: *.NET Framework programování aplikací*. Grada, 2003, ISBN 80-247-0450-1

[Uml] H. Janusová, M.Miller: *UML srozumitelně*, Computer Press, 2004, ISBN 80-251-0231-9

[Wiki] *Xfig*,
[WWW] http://en.wikipedia.org/wiki/Image:Xfig_screenshot.png,
[2007-05-16]

[1] Řazeno abecedně

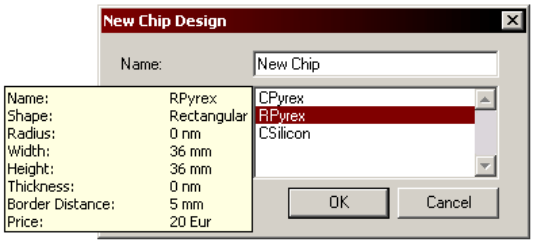
Přílohy

Příloha A: Vytvoření čipu

V této části si ukážeme jak pracovat s editorem. Navrhneme čip zobrazený v části (2.2). V jednotlivých krocích si ukážeme jak pracovat s editorem a jaké funkce můžeme při návrhu používat.

Krok 1)

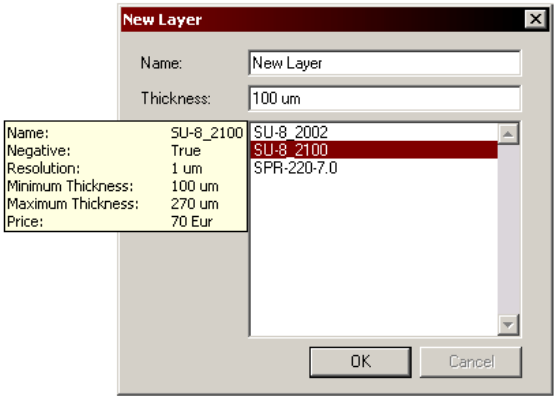
Prvním krokem který musíme provést je založení nového projektu. Založení nového projektu můžeme provést prostřednictvím zástupné ikony v panelu nástrojů nebo prostřednictvím položky menu (*File\New*). Po výběru jednoho ze způsobů založení projektu dojde k zobrazení formuláře (Obrázek A–1) určeného pro nastavení parametrů čipu. V tomto formuláři lze zvolit pojmenování čipu a substrát. Parametry daného substrátu se zobrazí ve formě nápovědy při výběru substrátu.



Obrázek A–1: Založení projektu

Krok 2)

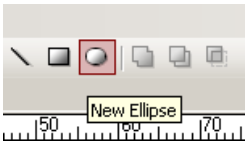
Po založení projektu dojde k zobrazení formuláře (Obrázek A-2) a nastavení parametrů vrstvy. Nastavitelnými parametry jsou jméno, tloušťka a typ vrstvy. Tloušťku je možné volit na omezeném intervalu definovaném pro danou vrstvu. Při výběru typu vrstvy jsou ve formě nápovědy zobrazovány parametry dané vrstvy.



Obrázek A-2: Založení vrstvy

Krok 3)

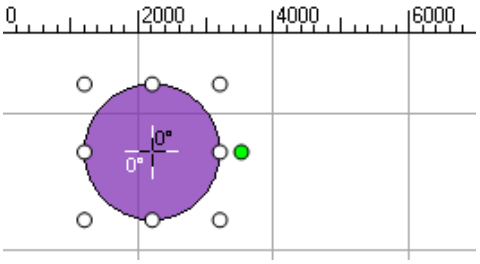
Nyní máme úspěšně založený projekt a můžeme začít navrhovat čip. Začneme nakreslením elipsy, elipsu vybereme v panelu nástrojů (Obrázek A-3).



Obrázek A-3: Výběr elipsy

Krok 4)

Elipsu začneme kreslit stisknutím levého tlačítka nad návrhovou plochou a tažením kurzoru. Uvolněním tlačítka myši dojde k ukončení kreslení.

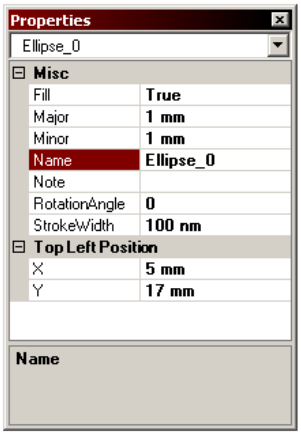


Obrázek A-4: Vytvoření elipsy

Krok 5)

V panelu vlastností se zobrazily parametry nakreslené elipsy. Prostřednictvím panelu vlastností je možné upravit jednotlivé atributy elipsy. Námi vytvořenou elipsu upravíme následovně:

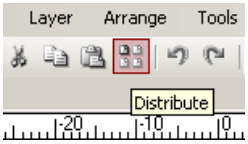
- délku hlavní a vedlejší poloosy nastavíme na 1mm
- umístíme objekt na pozici $x = 5mm$ a $y = 17\ mm$



Obrázek A-5: Panel vlastností

Krok 6)

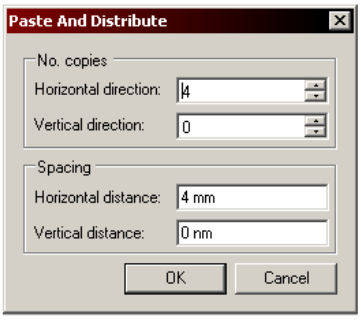
Nyní vybereme nástroj (Obrázek A-) pro vytvoření kopií, který se nachází v panelu nástrojů.



Obrázek A-6: Výběr kopírování objektů

Krok 7)

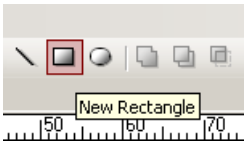
Vytvoříme čtyři kopie objektu s rozestupem v horizontální ose *4 mm*. Nyní máme vytvořeno pět elips, které jsou rovnoměrně rozmístěné.



Obrázek A-7: Nastavení parametrů kopírování

Krok 8)

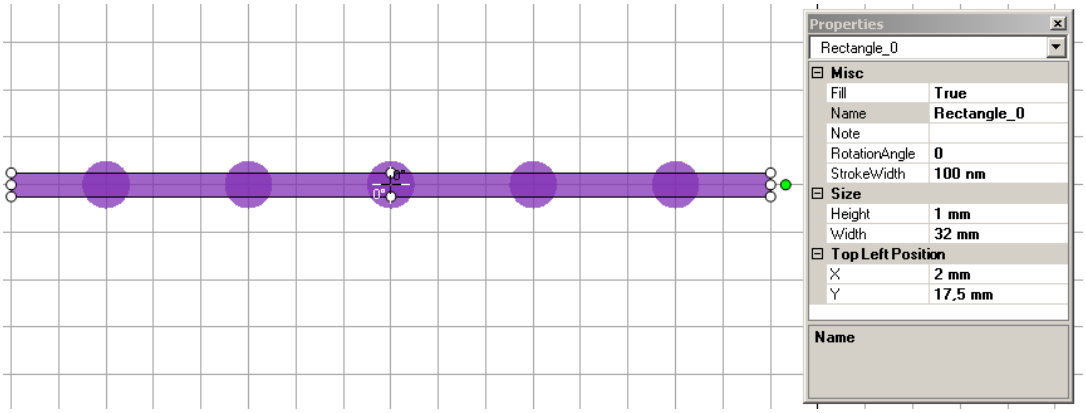
Nakreslíme obdélník, který bude procházet středem všech elips. V panelu nástrojů vybereme obdélník (Obrázek A) a obdélník nakreslíme. Můžeme se pokusit nakreslit obdélník přesně, nebo můžeme využít panelu vlastností a obdélník upravit na požadované parametry.



Obrázek A–8: Výběr obdélník

Krok 9)

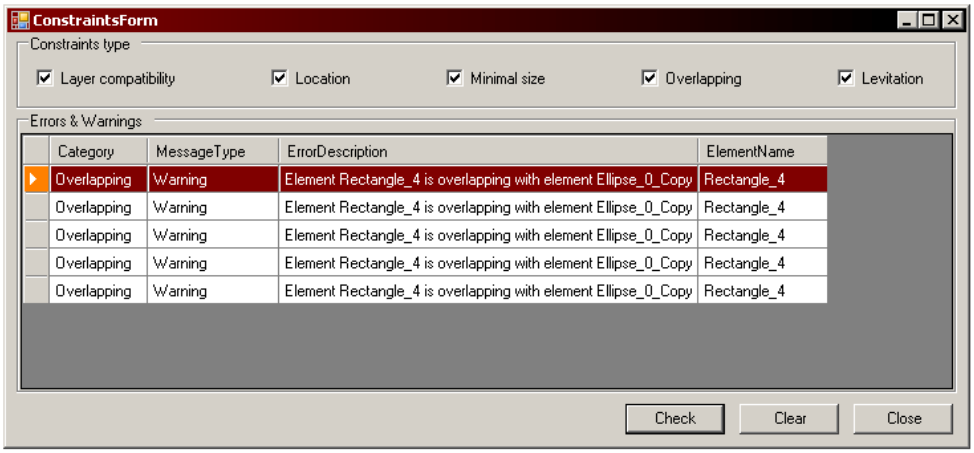
V panelu vlastností upravíme pozici obdélníku tak, aby byl vysoký *1 mm* a *32 mm* dlouhý. Obdélník umístíme na pozici $x = 2\ mm$ a $y = 17{,}5\ mm$. Návrh čipu je zobrazen na obrázku (Obrázek A-).



Obrázek A-9: Návrh čipu

Krok 10)

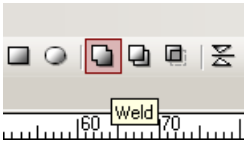
Námi navrhnutý čip necháme zkontrolovat. Tuto funkci vykonává nástroj validace čipu, který se nachází v záložce *Tools\Check Constraints*. Po výběru této položky se zobrazí formulář (Obrázek A-10). Nastavení necháme beze změn a stiskneme tlačítko *Check*, které spustí kontrolu navrhnutého čipu. Po dokončení validace se zobrazí varovné hlášení, které oznámí, že čip obsahuje překrývající se objekty. V tomto případě ve kterém čip obsahuje jenom jednu vrstvu můžeme varování ignorovat, v případech, kdy čip obsahuje více vrstev, je vhodné věnovat varování bližší pozornost. Tomuto problému se věnuje kapitola Validace 3.6.



Obrázek A-10: Validace čipu

Krok 11)

Varování odstraníme převedením objektů na jeden objekt a to objekt cesta. Objekt cesta vznikne aplikací booleovských operací na dva objekty. V tomto případě použijeme operaci sloučení (Obrázek A-11), která se nachází v panelu nástrojů.



Obrázek A-11: Sloučení objektů

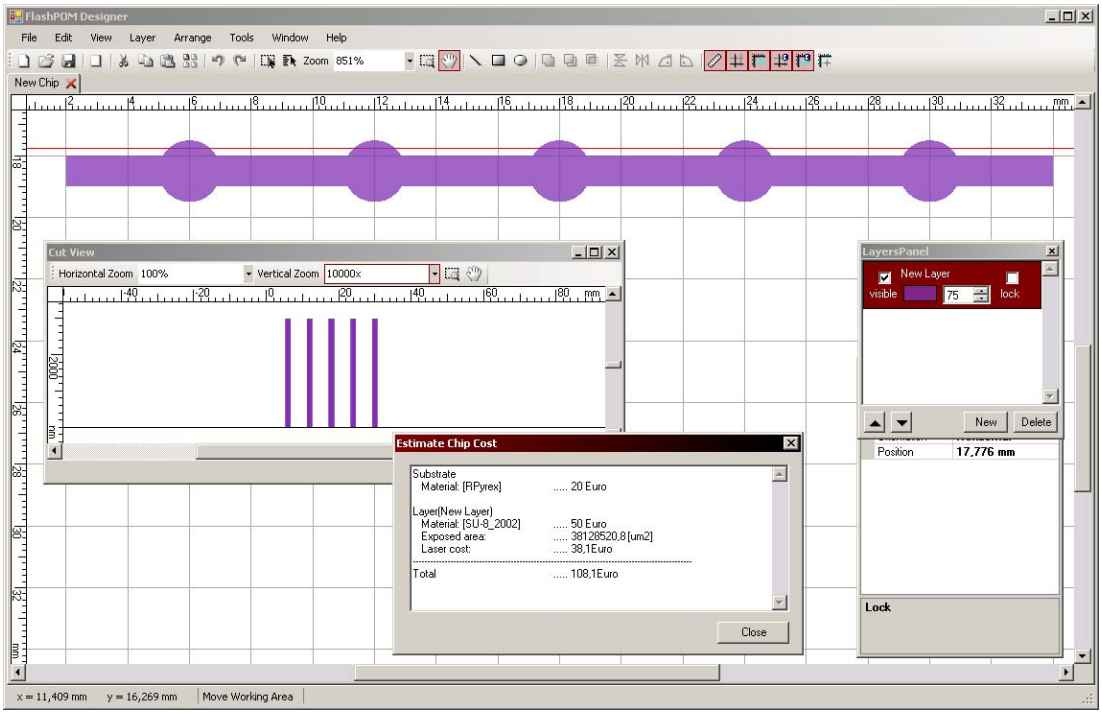
Krok 12)

Sloučení dvou elementů provedeme následovně. Vybereme obdélník, zvolíme operaci sloučení (Obrázek A-11) a vybereme jednu z elips. Tímto dojde k sloučení těchto elementů v jeden. Tento postup aplikujeme i na zbývající objekty.

Návrh čipu je tímto hotový. Editor nabízí nástroj, které umožňuje odhadu ceny navrhnutého čipu. Nástroj odhad ceny se nachází pod položkou menu *Tools\Estimate Chip Cost*. Dalším nástrojem, který může být využit, je nástroj *2D Cut.* Tento nástroj zobrazuje průřez čipu podle aktuálně nastavené pozice dělící roviny. Dělící rovina je zobrazena na návrhové ploše červenou čarou (Obrázek A-). Dalším podobným nástrojem je nástroj *3D View*. Tento nástroj zobrazuje 3D model navrhnutého čipu.

Pokud budeme chtít navrhnutý čip vyrobit, musíme vytvořit .fpsvg soubor. Výroba čipu bude probíhat ve Španělsku v Barceloně. Přesný způsob jak bude proces výroby probíhat je v současné době v přípravě. Představa je, že na základě zaslaného FPSVG

souboru bude čip vyroben. Bankovním převodem bude zaplacena výroba a materiály použité při návrhu a zpět bude zaslán vytvořený čip.



Obrázek A-12: Návrhová ploch

Příloha B: Přiložené CD

Obsah přiloženého CD:

- *FlashPoMEditor* – FlashPoM editor verze 0.99, spustitelný soubor .exe.
- *DiplmovePrace* – Text diplomové práce ve formátech .pdf a .doc.
- *Source* – Ukázky zdrojových kódů FlashPom editor verze 0.99.
- *Install* – Součásti nutné ke spuštění aplikace – Microsoft DirectX 9.0c a Microsoft Framework 2.0.

Verze CD pro interní použití Katedry Informatiky Západočeské Univerzity v Plzni navíc obsahuje:

- *FullSource* – Kompletní zdrojové kódy editoru FlashPoM.