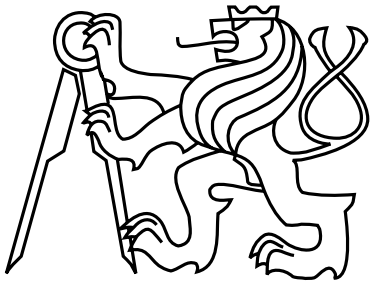
České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů



Bakalářská práce

Knihovna pro renderování dynamického textu dle šablon do 2D grafiky v jazyce Ruby

Dominik Mališ

Vedoucí práce: Ing. Pavel Strnad

Studijní program: Softwarové technologie a management, Bakalářský

Obor: Softwarové inženýrství

21. května 2012

Poděkování

Zde můžete napsat své poděkování, pokud chcete a máte komu děkovat.

Prohlášení

Prohlašuji, že jsem práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.
Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 24. 5. 2012 .....

Abstract

Translation of Czech abstract into English.

Abstrakt

Cílem práce je knihovna funkcí v jazyce Ruby, která by pomocí jednoduchých postupů dokázala efektivně vykreslovat dynamický text pomocí předem definovaných šablon.

Při využití knihoven Cairo (slouží pro vykreslování vektorové grafiky) a Pango (pro renderovaní textu), které se používají i v prostředí GTK2 (sada knihoven určených pro běh programů v grafickém uživatelském rozhraní) by bylo možné dosáhnout velice dobrých výsledků v rychlosti i kvalitě. Díky Cairu by bylo navíc možné výsledek exportovat do mnoha formátů a také GUI (grafické uživatelské rozhraní) aplikací v GTK2. Knihovna by nabízela jak interface pro formátování v jazyce Ruby, tak i možnost zpracování XML dokumentu se šablonou. Správnost XML dokumentů a také možnosti knihovny pro XML by specifikovalo DTD.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Využití knihovny vidím zejména v kombinaci se systémy, kde není možné používat lokální zdroje pro zobrazení a formátování požadovaného textového výsledku. Například Set-Top-Boxy, které mívají jeden druh písma, herní konzole nebo různé Video On Demand systémy. Problém těchto zařízení často tkví v tom, že jsou softwarově řešené na míru hardwaru a také účelu použití. Často pak obsahují velice omezený počet druhů písem, kterým lze ve výsledku maximálně změnit velikost nebo barvu. Nemluvě pak například o znacích s diakritikou nebo azbuce, hebrejštině, atd.

Obecně pak může řešení vypadat tak, že zřídíme centrální server, který pro koncová zařízení obsah předgenerovává (pak potřebujeme rychlou odezvu od serveru po síti) a nebo generuje „živě“, dle požadavků koncových zařízení (kde přibývá požadavek na rychlost renderování). Implementačním problémem takového řešení je, že potřebuje umět generovat veliké množství různých bloků textu a často s různým obsahem. Zpočátku nebude problém napsat nekolik skriptů pro generování textů do obrázků dle požadovaného rozložení, ale problém nastává v případě, že vytváříme rozsáhlou aplikaci. Tím pádem je takových skriptů mnoho - znepřehledňují zdrojový kód serverové aplikace, který se pak navíc velice často opakuje a o to hůř, když je potřeba rozložení některého obrázku změnit. Řešením takové situace je pak šablonovací systém, kde je každý obrázek definovaný jednoduchou šablonou v odděleném souboru.

Jsem si jistý, že sám budu mít příležitost, výslednou knihovnu použít například pro renderovaní vícejazykového textu pro takto omezená zařízení.

V případě splnění všech cílů bych rád poskytl knihovnu pro veřejné použití.

Kapitola 2

Popis problému, specifikace cíle

Knihovna musí nabízet co nejjednoduší a nejintuitivnější systém pro zadání šablon. Je důležité dodržet co nejpodobnější syntaxi při zadávání v různých formátech (ruby script, xml soubor). Knihovna musí být odolná vůči základním záměnám názvů atributů, jako je například velikost písmen či záměna pomlčky a podtržítka. Knihovna musí umožňovat jak volání metod na objektech, tak předání objektu pro vnořený blok a také volání metod v bloku bez použití předaného objektu.

Knihovna by měla zvládat generovat obrázky v co nejkratším čase oproti přímému zapsání bez použití knihovny. Výsledný čas by v žádném případě neměl přesáhnout dvojnásobek času, za jaký lze stejný obrázek vygenerovat bez použití knihovny.

Knihovna bude v základu podporovat načítání externích obrázků ve formátu PNG. Do téhož formátu musí jít také výsledek exportovat. Další formáty budou podporovány při volitelném načtení knihovny Gtk2 uživatelem, které nabízí zpracování mnoha dalších rastrových formátů. Knihovna bude bude umět automaticky detekovat knihovnu Gtk2 a použít ji.

Knihovna bude fungovat pod verzemi ruby 1.8 a 1.9.

Funkční požadavky lze rozdělit do tří skupin:

1. Zadání dat
   - Zadání počátečního stavu
     - Rozměry v pixelech
     - Žádného nebo pouze jednoho z rozměrů (ostatní se dopočítá podle obsahu, pokud je to možné)
     - Cestou k externímu obrázku nebo objektu Cairo::Surface či Cairo::Context
   - Vykreslení obrázku
     (a) Interface pro přímé zadání šablony v jazyce Ruby
       - Možnost získat zdrojový objekt (umožní zakreslení dat, pro které nejsou definovány metody)

     (b) Zadání šablony cestou k XML dokumentu nebo přímo obsahem dokumentu
       - Možnost specifikovat více obrázků v jednom dokumentu

2. Výstupní data

   (a) Export

   - Uložení výsledku ve formátu rastrového obrázku
   - Zápis do Ruby IO objektu (není potřeba obrázek zapisovat na disk, pokud ho chceme okamžitě odeslat do sítě)
   - Podpora rozdělení obsahu do několika stejně velkých obrázků pokud se elementy do jednoho nevejdou ("stránkování")

   (b) Žádný výsledek (například při prvotním zadání existujícího Cairo::Surface nebo Cairo::Conext objektu)

3. Textové/grafické operace

   Na elementech by mělo být možné definovat barvy výplně nebo ohraničení jednolitými barvami nebo různými druhy přechodů. Je nutností, aby bylo možné barvy zadávat v několika nejpřirozenějších formátech (od názvů, hex zápisu až po složkový zápis).

   Řešení bude obsahovat dva typy elementů:

   (a) Blokové elementy

   Jsou takové elementy, do kterých lze vkládat další elementy. Jsou to například:

   - Celá kreslící plocha
   - Libovolně vnořený blok
   - Napozicovaný blok
   - Tabulka

   Veškeré interní elementy se pak pozicují relativně vůči nadřazenému elementu.

   Tabulka je speciální případ blokového elementu, který je složený z několika bloků uspořádaných do matice a nabídne funkce pro dynamickou změnu rozměrů buněk dle obsahu.

   (b) Neblokové elementy

   Jsou zejména:

   - Externí obrázek
   - Textový blok
     - S možností dynamického zkracování textu (maximální počet řádků, slov, písmen nebo šířky v pixelech) a definování nahrazení chybějící části (např.: „...“).
     - Není nutné mu zadávat rozměry. Zdědí je automaticky z nadřazeného elementu.

Všem elementům bude možné zadat odsazení od ostatních elementů v kompozici. Samozřejmostí je zadání pro každou stranu svlášť. Blokové elementy budou mít možnost zadání také vnitřního odsazení.

2.1 Existující řešení a podobné projekty

Žádné existující implementace dle předešlého zadání nejsou známy.

Jedním z podobných řešení je formát SVG, který je XML dokument a nabízí velice rozsáhlé možnosti vetorové grafiky. K podobným účelům ho bohužel není možné použít, protože neřeší klíčové vlastnosti jako zkracování textu, závislosti elementů, „stránkování“ obrázků a pod.

Kapitola 3

Analýza a návrh řešení

Pro udržitelnost a rozšiřitelnost kódu je již od počátku nutné počítat s dodržování návrhového vzoru GRASP. Zejména pak low coupling a high cohesion, díky nimž bude snadné knihovnu dále rozšiřovat a vylepšovat kód.

3.1 Struktura elementů

Knihovna bude založená na návrhovém vzoru composite. Blokový element bude zastávat funkci složeného objektu, který obsahuje kolekci jiných objektů z nichž může být každý buď další blokový element nebo element neblokový. Neblokové elementy nebudou obsahovat referenci na žádné jiné podřazené objekty. Samotná třída poskytující rozhraní pro tvorbu obrázku bude dědit třídu blokového elementu, tím bude tvořit výchozí uzel (kořen), a umožní se tak do obrázku vkládat další elementy a vytvářet tak stromovou hierarchickou strukturu.

Blokový element bude poskytovat metody pro elementy, které je v něm možné vytvořit. Bude tak možné snadněji vytvářet strukturu elementů, ale také se zajistí, že každý blokový element bude obsahovat pouze elementy, které obsahovat má. Blokový element tak může obsahovat všechny elementy, ale tabulka, která je také blokový element, bude obsahovat pouze řádky a buňky. Taková metoda pak vytvoří novou instanci třídy elementu, ale také přidá objekt do kompozice. Blokový element pak bude mít zodpovědnost creator.

Zejména zde pak bude dodrženo „don't talk to strangers“, kdy každý objekt v hierarchické struktuře bude komunikovat pouze s objekty o úroveň nad nebo pod.

3.2 Barvy a povrchy

Elementům bude možné zadávat barvu. Některým barvu pozadí, jiným textu, rámečku a pod. Barvu nebo povrch, kterým mají být tyto části obarveny bude možné specifikovat co nejvíce možnými způsoby. Pro stejný barevný atribut tak bude možné zadat jednolitou barvu nebo přechod.

Zde se použije návrhový vzor abstract factory, kdy se hodnota barevného atributu předá abstract factory. Ta vybere concrete factory, která umí danou hodnotu přečíst a nechá ji vytvořit instanci třídy daného povrchu.

Tímto způsobem se také docílí high cohesion, jelikož vytváření povrchu bude rozděleno na jednotlivé třídy, kdy se každá specializuje na jednu věc, kterou umí udělat nejlépe. Factory vytvářející jednolitou barvu se pak bude zabývat zejména tím, aby dokázala reagovat na různý zápis barvy např. ve formátu RGB, CMYK, HSV, hexa, atd.

Přidat podporu dalšího povrchu pak bude znamenat pouze vytvoření jedné třídy, jelikož je zde maximalizován low coupling a high cohesion.

3.3 Podporované formáty obrázků

V základu bude pro načítání externích obrázků a ukládání možné vždy použít rastrový formát PNG. Cairo podporuje pouze tento rastrový formát.

Bude potřeba umožnit použití i dalších formátů, čehož lze docílit použitím knihovny Gtk. Použití této knihovny však zůstane volitelné a bude na uživateli, zda chce využít jejích rozšíření. Pokud bude knihovna v době použití jiných rastrových formátů k dispozici, bude automaticky detekována a použita.

3.4 Zadání obrázku

3.4.1 Rozměry obrázku

Rozměry obrázku bude možné zadat několika různými způsoby. Nejzákladnější možnost je zadání šířky a výšky, výsledkem je obrázek o daných rozměrech. Bude však možné zadat i pouze jeden z rozměrů a nebo žádný, kdy se nezadaný rozměr dopočítá podle velikosti obsahu. Obrázek bude také možné zadat externím obrázkem/objektem, kdy se použijí rozměry daného zdroje.

3.4.2 Vytvoření struktury

Z důvodu hierarchické struktury bude potřeba se na objektech vnořovat do dalších úrovní. Aby se pokrylo co nejvíce případů užití, bude třeba řešit tři způsoby vytváření elementů v blokovém elementu najednou.

1. Metoda zodpovědná za vytvoření podelementu vrátí instanci třídy podelementu. V případě, že to bude opět blokový element, půjde na tomto vráceném objektu volat metody opět jakýmkoli z těchto tří způsobů.

2. Metoda zodpovědná za vytvoření podelementu poskytne u blokových elementů zadání bloku kódu, kterému bude předána právě vytvořená instance třídy podelementu. Použití bude totožné jako v předchozím případě. Zdrojový kód tak bude pouze o něco estetičtější a pro člověka lépe čitelný.

3. Poslední metoda vychází z předešlé s tím rozdílem, že pokud blok kódu nepřevezme nově vzniklou instanci, tak jsou metody volány přímo na nově vzniklé instanci.

3.4.3 Atributy elementů

Elementům bude možné předávat atributy, které budou ovlivňovat chování a vzhled elementu podle požadavků. Atributy bude možné předat elementu při vytváření jako seznam klíčů a hodnot. Zde je nutné ošetřit, aby nebylo nutné dbát přesně na zadání. Vždy se může stát, že uživatel knihovny je zvyklý na jiný zápis než by se dalo očekávat anebo se jednoduše překlepne. Pro ošetření co nejvíce případů je třeba atributy ošetřit alespoň těmito kroky:

1. Převedení všech klíčů na stejný datový typ
2. Převedení znaků na malá písmena
3. Nahrazení pomlček a podtržítek za stejný znak
4. Převedení numerických hodnot zadaných jako text na čísla

Názvy atributů budou mít velice podobné či stejné názvy jako v CSS, aby nebylo nutné si znovu zapamatovávat další názvy pro stejné věci. Pokud navíc někdo zná CSS přidá se tím na intuitivnosti.

Dalším způsobem usnadnění používání atributů bude možnost použití několika předpřipravených aliasů, které umožní použít pro stejný atribut jak původní klíč, tak i zkrácený alias.

3.4.4 Uložení obrázku

Pro uložení obrázku budou k dispozici dvě metody.

1. Klasické uložení plátna do souboru. V případě, že nebude zadán cíl, tak dojde pouze k vykreslení obsahu. Toho lze využít v případě, že chceme pouze vykreslit obsah do již předvytvořeného Cairo objektu.
2. Uložení do několika obrázků jako stránkování. V případě, že se detekuje, že nějaký element bude vykreslen mimo prostor obrázku, tak se nevykreslí a po uložení a vytvoření nového obrázku se přesune o místo již vykreslených elementů a pokusí se o vykreslení znovu. Tato metoda musí také umožňovat omezení počtu stránek.

3.5 Zadání ve formátu XML

Knihovna bude umět také zadání obrázku ve formátu XML. XML dokument bude přesně kopírovat hierarchickou strukturu kompozice. Zadání atributů bude stejné jako při běžném použití knihovny.

Do XML dokumentu bude možné zadat i více obrázků najednou jako elementy v kořenovém elementu.

Zpracování XML dokumentu pak bude probíhat vnořováním do elementů a vytvářením hierarchické struktury na rozhraní třídy obrázku, která bude kopírovat zadání v XML dokumentu. Zpracování bude rekurzivní a dovolí tak přenést jakoukoliv strukturu. I zde budou platit omezení pro elementy, které lze do kterého blokového elementu vnořit.

Pro kontrolu správnosti bude dostupné DTD, kterým bude možné ihned zkontrolovat správnost dokumentu.

Kapitola 4

Realizace

Knihovna bude implementována v jazyce ruby s použitím co nejmenšího množství dodatečných knihoven, na kterých bude závislá.

4.1 Externí knihovny

4.1.1 Renderování obrázku

Pro generování se využije knihovna Cairo pro kreslení vektorové grafiky a knihovna Pango pro renderování textu. Jsou velice rychlé a využívá je také Gtk2, pro vytváření GUI. Výsledek je také velice kvalitní a umožňují široké možnosti použití.

Knihovna Cairo podporuje pouze PNG rastrový formát a proto bude volitelně možné uživatelsky načíst knihovnu Gtk2, které umožňuje využívat mnoho dalších rastrových formátů.

Ke správnému vykreslení nebude stačit pouze vytvořit stromovou hierarchickou strukturu, ale také dopočítávat veškeré pozice a rozměry elementů. Cairo umožňuje pouze vykreslit přesně zadanou vektorou grafiku na dané umístění. Nepodporuje žádné vztahy mezi elementy.

4.1.2 Parsování XML dokumentu

Parsování XML dokumentů bude zajištěno knihovnou REXML. Není sice nejrychlejší, ale je standardně dostupná a proto s touto závislostí neočekávám žádné problémy.

4.2 Struktura vlastní knihovny

Třídy knihovny se budou dělit do čtyř částí, které mezi sebou budou pouze minimálně provázané.

4.2.1 Třída rozhraní pro zadávání obrázku

Třída DynamicImage bude poskytovat veškeré potřebné metody pro vytvoření obrázku, zadání hierarchycké struktury, vekreslení a uložení obrázku.

4.2.2 Modul elementů

Modul DynamicImageElements bude obsahovat všechny elementy, které bude možné do kompozice umístit. Všechny elementy budou implementovat rozhraní ElementInterface, které bude implementovat návrhový vzor composite a bude řešit společné úlohy všech elementů.

Implementovat se budou následující elementy:

- BlockElement bude blokový element pro kompozici elementů: BlockElement, ImageElement, TableElement, TextElement.
- ImageElement bude neblokový element, který bude do kompozice vykreslovat externí obrázek.
- TableElement bude blokový element, který bude obsahovat pouze TableCellElement objekty. Bude poskytovat také metodu row, ale ta nebude vytvářet nové objekty, pouze virtuální řádek.
- TableCellElement bude dědit třídu BlockElement.
- TextElement bude neblokový element pro vykreslení textu do kompozice. Bude nabízet mnoho voleb pro dynamickou úpravu textu v závislosti na okolí.

4.2.3 Modul barev a povrchů

Modul DynamicImageSources bude obsahovat abstract factory, která bude zároveň sloužit jako connector celého modulu. Z ní budou dědit všechny concrete factory. Abstract factory pak bude zkoušet parsovat hodnotu na všech známých potomcích.

Implementovat se budou následující concrete factory:

- ColorFactory řešící jednolité barvy včetně průhlednosti. Barvy bude možné zadávat ve formátech RGB, CMYK, HSV, hexa a názvem barvy. Všechny číselné hodnoty půjde zadat jako celé číslo v rozmezí od 0 - 255 nebo desetinné číslo v rozmezí 0.0 - 1.0.
- GradientFactory řešící barevný přechod a to jak lineární tak i radiální. Bude možné snad zadat konkrétní souřadnice přechodu, ale i upravovat dynamicky vypočítané hodnoty. Barevné přechody budou využívat ColorFactory pro co nejširší možnosti zadání barev.

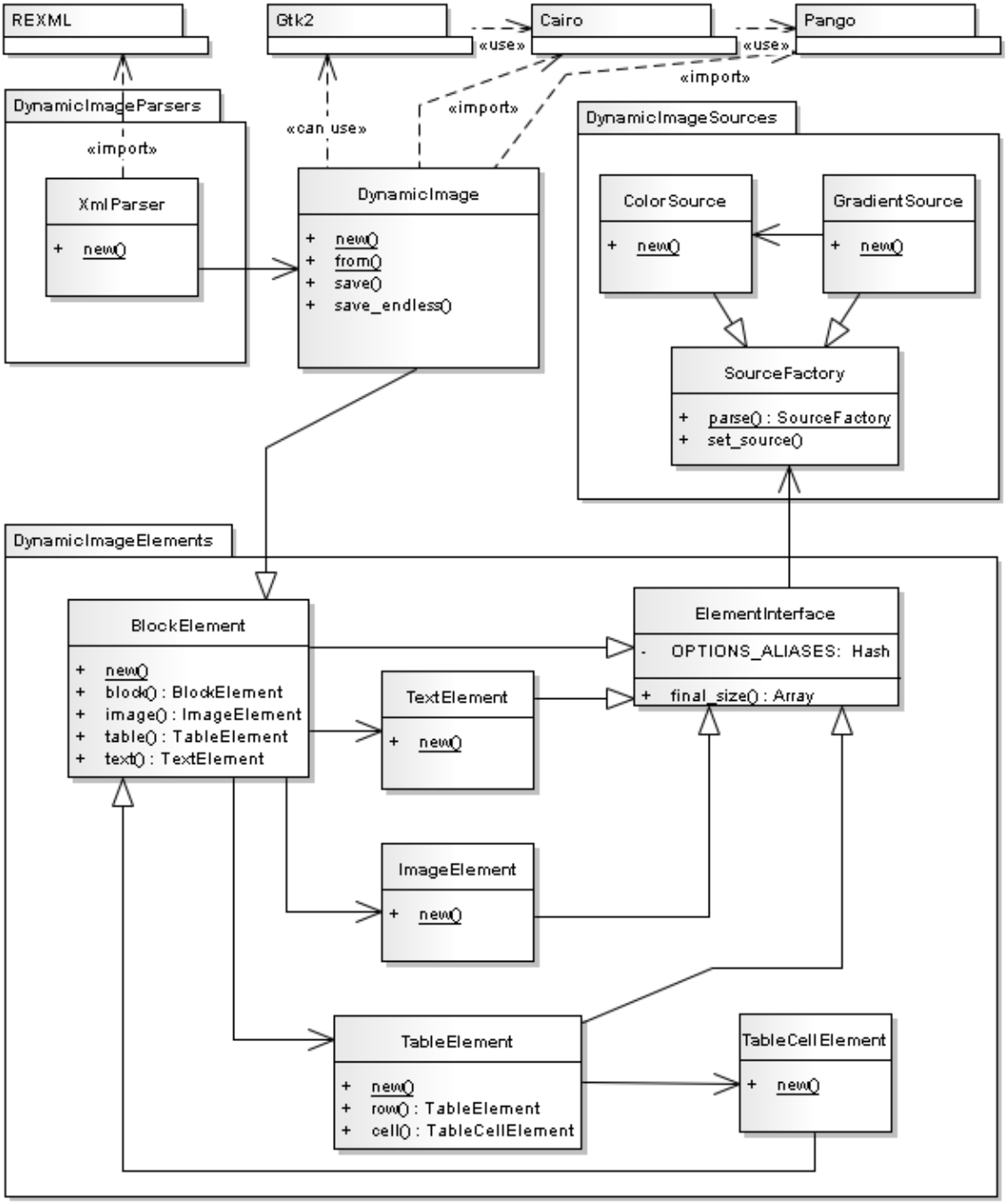
Concrete factory budou zároveň implementovat rozhraní ConcreteSources, které bude zadávat rovněž abstract factory třída SourceFactory.

4.2.4 Modul tříd pro parsování zadání v externích souborech

Modul DynamicImageParsers bude obsahovat jednotlivé třídy, které budou umět projích strukturou daného souboru a tuto strukturu vytvořit na třídě DynamicImage.

Implementovat se budou následujicí třídy:

- XmlParser pro parsování XML souborů s využitím externí knihovny REXML.



Obrázek 4.1: Diagram tříd

4.3 Implementace knihovny

Rozložení tříd do souborů a souborová struktura bude dodržovat používané konvence pro vytváření knihoven v jazyce ruby. Díky tomu pak bude velice jednoduché převést knihovnu na gem (archiv souborů) a umožnit její jednoduchou instalaci a distribuci.

Nejprve je potřeba implementovat třídu blokového elementu (BlockElement) a z ní lze následně vytvořit potomka třídy pomocí které bude možné vytvářet obrázky - DynamicImage. Do objektu typu DynamicImage tak bude možné vkládat další elementy stejně jako do blokového elementu a zároveň bude poskytovat metody pro inicializaci a ukládání do souboru.

4.3.1 Obecně o zadávání a výpočtu rozměrů elementu

Finální rozměry elementu se mohou lišit v závisloti na zadané hodnotě. Do rozměru se zahrnuje také vnější odsazení a rámeček (obrys). U blokového elementu navíc i vnitřní odsazení. Mohou tak nastat tři možnosti.

1. Je zadán absolutní rozměr v pixelech.
   - Pak se rozměr považuje za rozměr obsahu.
   - Finální rozměr je vypočítán jako rozměr obsahu + vnitřní odsazení + obrys + vnější odsazení.
2. Je zadán relativní rozměr v procentech.
   - Vypočítá se absolutní rozměr z nadřazeného elementu.
   - Absolutní rozměr se použije jako finální rozměr, do kterého se element musí vejít.
3. Není zadán žádný rozměr.
   - Vypočítá se rozměr obsahu z rozměrů, které zabírají podřazené elementy.
   - Finální rozměr je vypočítán jako rozměr obsahu + vnitřní odsazení + obrys + vnější odsazení.

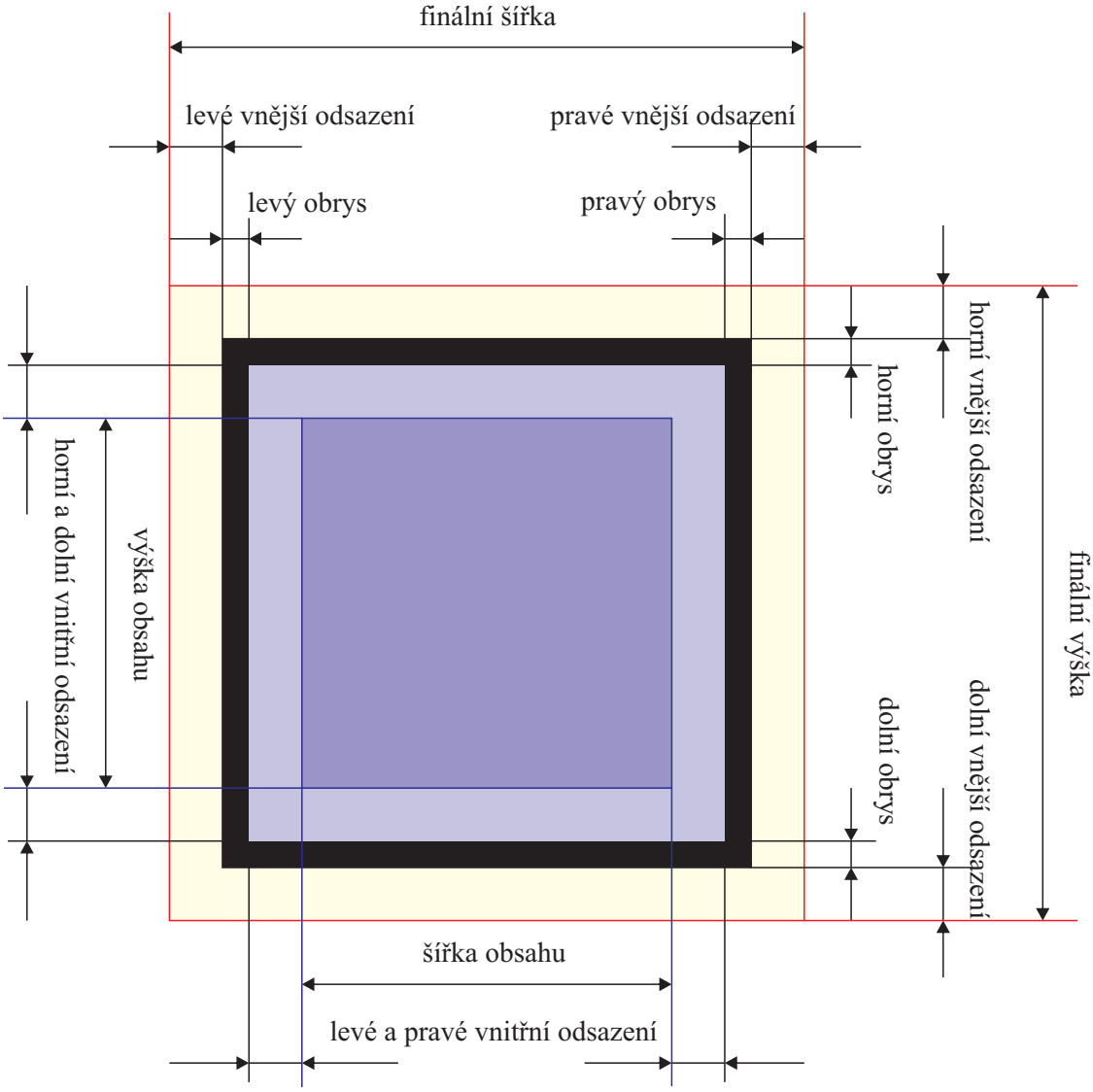
4.3.2 Obecně o zadávání a výpočtu pozice elementu

Pozice elementu je vypočtena relativně vůči nadřazenému elementu a jeho vnořeným elementům, které se jeví pro element jako okolní elementy. Pozice elementu je dána souřadnicemi levého horního rohu obdélníku o velikosti finálních rozměrů elementu.

Každý element před sebou a za sebou zalamuje řádek a ovlivňuje tak pouze vertikální pozici následujících elementů. Elementy je možné umísťovat vedle sebe pomocí tabulky.

Lze zvolit jeden ze tří způsobů pozicování elementu.

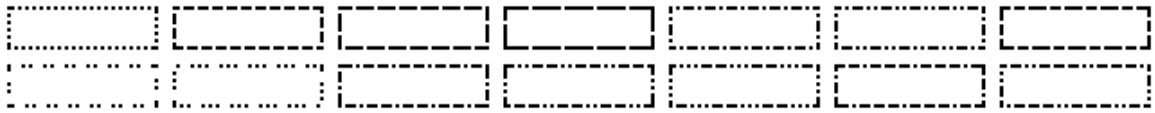
1. Statické - element je umístěn na vypočtené pozici v toku dokumentu.
2. Relativní - element je posunut o dané hodnoty ze své vypočtené pozice.
3. Absolutní - element je vyjmut z toku dokumentu a umístěn přesně na dané pozici vůči nadřazenému elementu.



Obrázek 4.2: Složky rozměrů elementu

4.3.3 BlockElement a DynamicImage

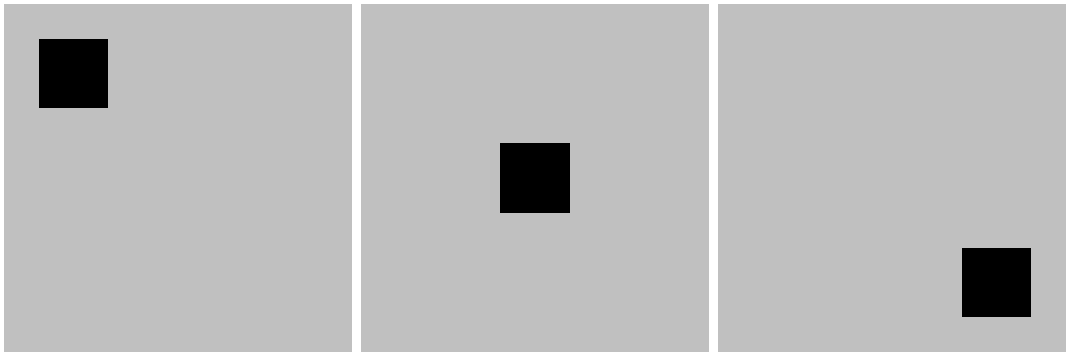
Blokový element je základním stavebním prvkem pro vytváření kompozice. Nemá žádné zvláštní vykreslovací schopnosti, ale lze u něho vykreslit rámeček a pozadí. Pro vykreslení rámečku jsou dostupné přednastavené vzory, které lze rozšiřovat do délky. Lze mu také předat barvu textu, kterou předá všem svým vnořeným elementům.



Obrázek 4.3: Ukázka předdefinovaných vzorů ohraničení

Blokový element také umí zarovnávat vnořené elementy horizontálně tím způsobem, že každý vnořený element má vertikální osu společnou s osou nadřazeného elementu. Umožňuje také vertikální zarovnání, kterého je docíleno tak, že jsou všechny vnořené elementy zarovnány jako skupina na střed elementu.

Blokovému elementu lze jako jedinému nastavit vnitřní odsazení, které udává, jak daleko budou vnořené elementy vzdálené od okrajů daného blokového elementu.



(a) Vlevo (b) Na střed v obou osách (c) Vpravo dolů

Obrázek 4.4: Ukázky zarovnání vnitřních elementů v blokovém elementu

Třída DynamicImage dědí třídu BlockElement, čímž se z ní stává kořenový element stromové hierarchické struktury. Navíc implementuje také metody save a save_endless a umožňuje inicializovat plátno načtením externího obrázku.

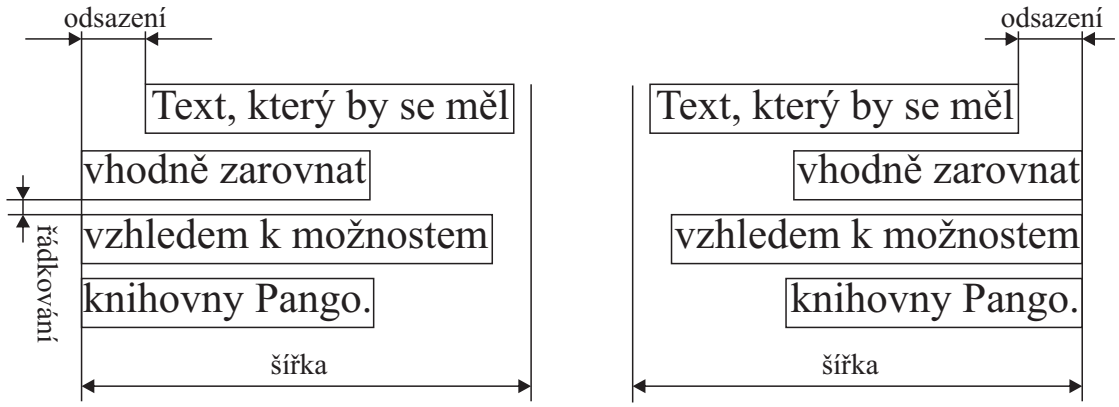
Metoda save uloží aktuální kompozici do souboru. Formát souboru se identifikuje podle přípony v názvu souboru. Metodě lze také předat IO objekt do kterého lze výsledek zapsat a následně jeho obsah například odeslat po síti. Pokud se metodě nepředá žádný parametr, tak kompozici pouze vykreslí. Toho lze využít pokud se bude krelit na již vytvořený Cairo objekt.

Metoda save_endless funguje stejně jako metoda save s tím rozdílem, že vykreslí pouze elementy, které se vejdou do rozměrů obrázku. Rozměry tak musí být předem zadány při vytváření obrázku. V případě, že existuje element, který se nevykreslil, vytvoří se nový

obrázek a vykresluje se znovu. V dalším vykreslování se přeskakují elementy, které již byly vykresleny.

4.3.4 TextElement

Textový element slouží k vykreslování blokového textu. K vykreslování používá knihovnu Pango a je tak možné použít všechny její možnosti stylování textu. Mezi základní možnosti patří zarovnání, řádkování, odsazení, zarovnání do bloku a automatické rozpoznání směru toku textu.



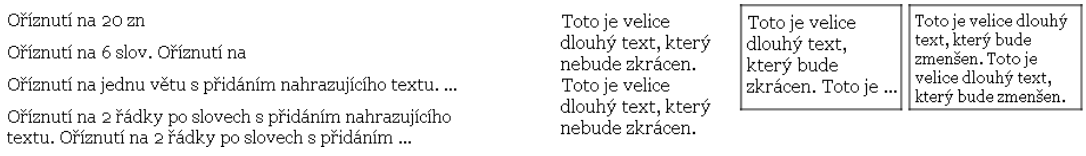
(a) Zarovnání vlevo (b) Zarovnání vpravo

Obrázek 4.5: Základní možnosti zarovnání textového elementu

Textový element pak disponuje dalšími metoda pro úpravu textu. První je oříznutí, kde element automaticky zkrátí jakýkoliv text na požadovanou délku ubíráním písmen, slov nebo vět. Lze také zadat maximální počet řádků, na které má být text zkrácen a jakým způsobem.

Druhou možností je zmenšení nebo oříznutí textu, aby se zobrazil v nadřazeném elementu o daných rozměrech celý, bez přetékání. Obě metody oříznutí nebo zmenšení lze kombinovat, ale mezi metodami je nutné zadat „stop“ hodnotu, od které se začne s další metodou v pořadí.

U obou případů, kdy může být text zkrácen je možné zadat text, který se připojí na konec zkrácené části. Např.: „ ...“.



(a) Oříznutí textu (b) Úpravy textu

Obrázek 4.6: Ukázky modifikace textu

4.3.5 ImageElement

Element obrázku umožňuje vkládat do kompozice externí rastrové obrázky. Vždy je dostupná podpora formátu PNG. V případě, že je uživatelem načtena knihovna Gtk2, třída ji automaticky detekuje a je schopna ji použít pro načtení dalších formátů.

Element umožňuje základní modifikace obrázku změnou rozměrů nebo oříznutím. Rozměry je možné zadat i v procentech, kdy se výsledné rozměry vypočítají z nadřazeného elementu nebo zadáním fixních rozměrů do kterých se obrázek přizpůsobí. Oříznutí je dáno obdélníkem, kdy je nutné zadat jeho rozměry a pozici levého horního rohu na originálním obrázku. U výřezu je pak možné také měnit rozměry stejným způsobem.

Obrázek 4.7: Ukázka změn velikosti a výřezu u externího obrázku

Obrázku je možné zadat i průhlednost, se kterou se má vykreslit.

Obrázek 4.8: Ukázka průhlednosti u externího obrázku

4.3.6 TableElement a TableCellElement

Tabulka je speciální případ blokového elementu, kterému je možné do kompozice vložit pouze elementy, které jsou buňkami tabulky. Buňka tabulky dědí blokový element a je tak možné v ní opět vytvářet stromovou strukturu ze všech elementů, které lze vložit do blokového elementu.

Tabulka udržuje vnořené elementy v maticové struktuře a před vykreslením přepočítává rozměry a pozice všech buněk tak, aby došlo k vykreslení buněk do pravidelné matice.

Pro rozdělování buněk do řádků je k dispozici metoda, která se z uživatelského pohledu jeví jako nový element, ale pouze posune vnitřní ukazatel v maticové struktuře na první pozici dalšího řádku. Element tabulky je možné používat i bez manuálního rozdělování buněk do řádků tak, že při jejím vytvoření zadáme fixní počet sloupců a tabulka pak sama vytvoří nový řádek, pokud se dosáhne takového počtu buněk v řádku.

Buňka tabulky lze rozříšit přes více sloupců nebo i řádků, čímž se docílí sloučení buněk do jedné.

4.3.7 SourceFactory, ColorSource a GradientSource

SourceFactory je abstract factory, která zároveň poskytuje interface pro concrete factory. Zavolání parsovací metody pak projde všechny potomky třídy a pokud nějaká dokáže parsovat zadání a vrátí objekt, tak se dál nepokračuje a vrátí se vytvořený objekt.

ColorSource vytváří jednobarevný povrch z červené, zelené, modré a průhledné složky. Umí parsovat pojmenované barvy z Cairo::Color podle názvu, volitelně lze speficikovat i průhledný kanál barvy. V základu se očekává 3 (RGB) až 4 (RGBA) složkový číselný zápis v intervalech <0 - 255> nebo <0.0 - 1.0>. Je možné použít i zápis v jiných složkách, ale je nutné to specifikovat před zápisem složek. Další podporované složkové zápisy jsou CMYK(A) a HSV(A). Lze použít také HEX zápis ve formátu začínajícím znakem „#“.

GradientSource vytváří barevný přechod a může být lineární nebo radiální. Vždy je popsán vektorem barev, k čemuž využívá třídy ColorSource, aby nabídl stejné možnosti zadání barev. U lineárního přechodu je možné specifikovat velikost vektoru a jeho směr. U radiálního přechodu je vektor mezi dvěmi kružnicemi, které lze od středu libovolně vychylovat nebo zadat jejich absolutní umístění i velikost.

Obrázek 4.9: Ukázka barevných přechodů

U obou typů přechodu je možné zvolit opakování vektoru, jeho odraz nebo pokračování krajními barvami.

4.3.8 XmlParser

Třída XmlParser pro renderování obrázků popsaných XML dokumentem využívá externí knihovny REXML. Prochází dokumentem jako stromovou strukturou a stejnou strukturu rekuzivně vytváří i na novém objektu typu DynamicImage.

Kořenový prvek XML dokumentu je „dynamic_images“, který obsahuje libovolný počet obrázků jako elementy „dynamic_image“.

Struktura obrázku v XML dokumentu je totožná jako při vytváření obrázku v ruby za použití bloků kódu. Atributy elementů jsou zapsány shodně jako atributy XML elementů. U názvů atributů je možné libovolně zaměňovat pomlčky s podtržítky a velikost písmen. V případě, že je třeba použít hodnotu atributu, která se v ruby zapisovala polem hodnot, tak se hodnoty zapíší za sebe oddělené mezerou.

Validnost dokumentu lze zkontrolovat veřejně dostupným DTD. Chytřejší vývojářské nástroje pak mohou nabídnout i automatické doplňování názvů elementů a atributů.

4.4 Příklady

Výčet základních příkladů použití knihovny. Vytvořením větší stromové hierarchické struktury lze dosáhnout i velice složitých výsledků.

4.4.1 Způsoby vytváření kompozice

První možností je volání metod přímo na objektech.

```
img = DynamicImage.new
img.text "<b>Warning</b>"
img.save! "warning.png"
img.destroy
```

Další možností je využít v bloku objekt, který předá metoda. V případě zápisu bloku není nutné volat metodu destroy, vykoná se automaticky na konci bloku.

```
DynamicImage.new do |img|
  img.text "<b>Warning</b>"
  img.save! "warning.png"
end
```

Poslední možností je zápis metod přímo. V případě, že v bloku nepřevezmeme objekt, vykonávají se metody přímo na objektu. Tento přístup se bude používat ve všech následujících příkladech.

```
DynamicImage.new do
  text "<b>Warning</b>"
  save! "warning.png"
end
```

4.4.2 Vnoření do blokového elementu

V ukázce se vytvoří dva čtverce stejně jako v obrázku 4.4b, kde vnitřní navíc bude obsahovat bílé písmeno „X“.

```
DynamicImage.new :w => 100, :h => 100, :bg => :silver,
                 :align => :center, :valign => :middle do
  block :w => 20, :h => 20, :bg => :black, :margin => 10 do
    text "X", :color => :white
  end
  save! "x.png"
end
```

4.4.3 Vytvoření tabulky

První způsob, jak vytvořit tabulku 2x3.

```
DynamicImage.new do
  table do
    row do
      cell { text "1. řádek, 1. sloupec" }
      cell { text "1. řádek, 2. sloupec" }
      cell { text "1. řádek, 3. sloupec" }
    end
    row do
      cell { text "2. řádek, 1. sloupec" }
      cell { text "2. řádek, 2. sloupec" }
      cell { text "2. řádek, 3. sloupec" }
    end
  end
  save! "table.png"
end
```

Druhý způsob vytvářející totožnou tabulku.

```
DynamicImage.new do
  table :cols => 3 do
    cell { text "1. řádek, 1. sloupec" }
    cell { text "1. řádek, 2. sloupec" }
    cell { text "1. řádek, 3. sloupec" }
    cell { text "2. řádek, 1. sloupec" }
    cell { text "2. řádek, 2. sloupec" }
    cell { text "2. řádek, 3. sloupec" }
  end
  save! "table.png"
end
```

Vytvoření tabulky 3x3, kde první buňka zabírá dva sloupce a dva řádky.

```
DynamicImage.new do
  table :cols => 3 do
    cell :colspan => 2, :rowspan => 2 { text "1. řádek, 1. sloupec" }
    cell { text "1. řádek, 3. sloupec" }
    cell { text "2. řádek, 3. sloupec" }
  end
  save! "table.png"
end
```

4.4.4 Modifikace textu

Oříznutí textu na danou délku z ukázky 4.6a.

```
DynamicImage.new :w => 400 do
  text "Oříznutí na 20 znaků. Oříznutí na 20 znaků.",
       :crop_to => [20, :letters], :margin_bottom => 10
  text "Oříznutí na 6 slov. Oříznutí na 6 slov.",
       :crop_to => 6, :margin_bottom => 10
  text "Oříznutí na jednu větu s~přidáním nahrazujícího textu. " +
       "Oříznutí na jednu větu s~přidáním nahrazujícího textu",
       :crop_to => [1, :sentences], :crop_suffix => " ...",
       :margin_bottom => 10
  text "Oříznutí na 2 řádky po slovech s~přidáním nahrazujícího textu. " +
       "Oříznutí na 2 řádky po slovech s~přidáním nahrazujícího textu.",
       :crop_to => [2, :lines], :crop_suffix => " ..."
  save! "image_text_crop.png"
end
```

Modifikace textu do daných rozměrů z ukázky 4.6b.

```
DynamicImage.new do
  table :w => 450 do
    cell :padding => 5 do
      text "Toto je velice dlouhý text, který nebude zkrácen. " +
           "Toto je velice dlouhý text, který nebude zkrácen."
    end
    cell :margin => [0, 5] do
      block :w => 1.0, :h => 0.75, :padding => 5,
            :border => [1, :solid, :black] do
        text "Toto je velice dlouhý text, který bude zkrácen. " +
             "Toto je velice dlouhý text, který bude zkrácen.",
             :to_fit => :crop, :crop_suffix => " ..."
      end
    end
    cell do
      block :w => 1.0, :h => 0.75, :padding => 5,
            :border => [1, :solid, :black] do
        text "Toto je velice dlouhý text, který bude zmenšen. " +
             "Toto je velice dlouhý text, který bude zmenšen.",
             :to_fit => :resize
      end
    end
  end
  save! "image_text_crop_resize.png"
end
```

4.4.5 Využití externích souborů

Příklad, jak načíst obrázek, vytvořit přes něj poloprůhlednou bílou vrstvu a jeho poloviční kopii umístit na střed.

```
DynamicImage.from "pict.jpg", :bg => [:white, 0.5],
                  :align => :center, :valign => :center do
  image "pict.jpg", :w => "50%", :h => "50%"
  save! "new_pict.jpg"
end
```

Je tak možné použít knihovnu k převodu z jednoho formátu do jiného.

```
DynamicImage.from("pict.png").save!("pict.jpg", :quality => 75)
```

4.4.6 XML dokument

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE dynamic_images PUBLIC "-//malis//dynamic_images//EN"
  "https://raw.github.com/malis/dynamic_images/master/lib/parsers/xml.dtd">
<dynamic_images>
 <dynamic_image from_source="earth.jpg" align="center" valign="middle"
                background="red 0.5" save="test_from.jpg" quality="90">
  <text font="Arial bold 15">testing <u>adding</u> some text to image</text>
  <block background="red">
    <image w="50" h="50">kostky.png</image>
  </block>
  <table>
    <row>
      <cell>
        <text><b>Left table header</b></text>
      </cell>
      <cell>
        <text><b>Right table header</b></text>
      </cell>
    </row>
    <row>
      <cell>
        <text>Table value 1</text>
      </cell>
      <cell>
        <text>Table value 2</text>
      </cell>
    </row>
  </table>
 </dynamic_image>
</dynamic_images>
```

Kapitola 5

Testování

Testování bude probíhat několika po sobě jdoucími metodikami, které ověří funkčnost i výkon knihovny.

Nejprve je nutné ověřit, zda je knihovna schopna vůbec fungovat a pak do ní lze „mlátit kladivem“. Tyto testy jsou charakteristické jako dynamické testování černé skříňky. Otestují software z pohledu celku, bez znalosti vnitřních principů a algoritmů. Je možné je pouštět i pravidelně a testovat tak software během vývoje, kdy se zjistí, zda se nepoškodila nějaká část programu.

Nakonec se otestuje snížení výkonu při vykreslování pomocí knihovny oproti využití knihoven Cairo a Pango přímo. Zjistí se také průběh nárůstu času pro vykonání v závislosti na velikosti obrázku.

5.1 Test splněním

Od testu se očekává pozitivní výsledek pro každé testované zadání. Během testování nesmí dojít v knihovně k výjimce nebo fatální chybě. Test tak slouží zejména pro nalezení chyb v programu nebo chyb v syntaxi.

Tímto testem musí knihovna bezpodmínečně úspěšně projít. V případě, že se vyskytnou nějaké chyby, je nutné je opravit.

5.1.1 Návrh testů

Test bude program, který bude předem znát všechny elementy a jejich atributy. U každého atributu bude nulová hodnota (nepoužito) a minimálně jedna validní hodnota. Test bude pomocí knihovny zkoušet renderovat zadání pro kombinace z množiny elementů a jejich atributů.

V případě, že daná kombinace způsobí výjimku, program vypíše znak „E“ v opačném případě znak „.“.

5.1.2 Provádění testů - výsledky

Po vykonání testu unit1.rb vznikne následující výsledek. Jelikož bylo teček mnoho, tak jsou zkráceny.

```
..... # ...

Passed: 1900/1900
Failed: 0/1900
```

Z výsledku testu je patrné, že bylo vygenerováno 1900 obrázků. 1900 z nich bez chyby a 0 s chybou. Z toho vyplývá, že se v knihovně s největší pravděpodobností nenalézají žádné chyby. Knihovna tak úspěšně prošla tímto testem.

5.2 Test selháním

Test bude knihovně zadávat neplatné, chybné, nesprávné a nesmyslné údaje za účelem vyvolat chybné nebo nepředvídatelné chování. Takové chování by nemelo nastat. Knihovna by měla zvládat zpracovat co nejširší množství vstupů. V případě, že vstupu nebude rozumět, tak by neměl být zpracován.

5.2.1 Návrh testů

Test bude program vycházející z testu splněním. Bude také znát všechny elementy a jejich atributy, ale bude mít zadané chybné hodnoty pro atributy. Žádný z testů nesmí vyvolat vyjímku. Knihovna musí umět nesprávnou hodnotu přejít a pokračovat dál. To, že hodnota nebyla rozpoznána a nezpracovala se musí uživatel zkontrolovat vizuálně. Toto chování pomáhá v případě, že hodnota zadaná uživatelem bude získanána z externího zdroje a neprojde kontrolou. Pak není možné způsobit chybu v programu vnějším vlivem.

Jako hodnoty atributů se budou testovat:

- Prázdné, nevyplněné hodnoty
- Hraniční případy
- Neplatné, chybné, nesprávné a nesmyslné údaje

5.2.2 Provádění testů - výsledky

Po vykonání testu unit2.rb vznikne následující výsledek. Jelikož bylo teček velice mnoho, tak jsou zkráceny.

```
..... # ...

Passed: 6342/6342
Failed: 0/6342
```

Výsledek testu říká, že bylo vygenerováno 6342 obrázků. 6342 z nich bez chyby a 0 s chybou. Žádný z testovaných nestandartních vstupů nezpůsobil nefunkčnost knihovny. Knihovna tak úspěšně prošla tímto testem.

5.3 Testování výkonu

Jedním z požadavků na knihovnu je rychlost generování obrázků. Knihovna by měla zvládat generovat obrázky v co nejkratším čase oproti přímému zapsání bez použití knihovny. Výsledný čas by v žádném případě neměl přesáhnout dvojnásobek času, za jaký lze stejný obrázek vygenerovat bez použití knihovny.

5.3.1 Návrh testů

Testem bude program, který bude znát několik zadání obrázků. Každé zadání bude mít verzi s použitím knihovny a druhou s použitím knihoven Cairo a Pango přímo. Výsledkem takové dvojice by měl být totožný obrázek. Generování každého obrázku se spustí mnohokrát a změří se celkový čas generování, který se vydělí počtem spuštění. Tím se získá průměrný čas, za který se každý obrázek vygeneroval a porovná se v dané dvojici zadání.

5.3.2 Provádění testů - výsledky

Po vykonání testu performance.rb vznikne následující výsledek pro několik různých konfigurací. (Uvádím alespoň základní informace o konfiguraci.)

Pro konfiguraci: Intel Core 2 duo 2.0Ghz, 2GB RAM, OS: Windows XP Professional, ruby 1.8.7

```
====================== TEST 1 ======================
Time without lib: 40.625 ms
Time with lib:    48.4375 ms
Diference is:     7.8125 ms (19%)

====================== TEST 2 ======================
Time without lib: 33.59375 ms
Time with lib:    46.875 ms
Diference is:     13.28125 ms (40%)

====================== TEST 3 ======================
Time without lib: 78.28125 ms
Time with lib:    102.96875 ms
Diference is:     24.6875 ms (32%)
```

Pro konfiguraci: Intel Xeon 2.66Ghz, 32GB RAM, OS: Debian GNU/Linux 6.0 (virtual), ruby 1.8.7

```
==================== TEST 1 ====================
Time without lib: 28.57616 ms
Time with lib:    55.9034 ms
Diference is:     27.32724 ms (96%)

==================== TEST 2 ====================
Time without lib: 22.03885 ms
Time with lib:    28.19817 ms
Diference is:     6.15932 ms (28%)

==================== TEST 3 ====================
Time without lib: 52.43772 ms
Time with lib:    89.0935 ms
Diference is:     36.65578 ms (70%)
```

Pro konfiguraci: Intel Xeon 2.66Ghz, 32GB RAM, OS: Debian GNU/Linux 6.0 (virtual), ruby 1.9.2

```
==================== TEST 1 ====================
Time without lib: 28.58125879 ms
Time with lib:    54.18401913 ms
Diference is:     25.60276034 ms (90%)

==================== TEST 2 ====================
Time without lib: 21.93487545 ms
Time with lib:    26.73286004 ms
Diference is:     4.79798459 ms (22%)

==================== TEST 3 ====================
Time without lib: 52.66458473 ms
Time with lib:    84.95335805 ms
Diference is:     32.28877332 ms (61%)
```

Po vytvoření tabulky a spočítání průměrů vyjde následující výsledek.

Konfigurace - Test	Bez knihovny	S knihovnou	Rozdíl	Rozdíl
WinXP ruby 1.8.7 - 1	40.625 ms	48.4375 ms	7.8125 ms	19%
WinXP ruby 1.8.7 - 2	33.59375 ms	46.875 ms	13.28125 ms	40%
WinXP ruby 1.8.7 - 3	78.28125 ms	102.96875 ms	24.6875 ms	32%
Debian ruby 1.8.7 - 1	28.57616 ms	55.9034 ms	27.32724 ms	96%
Debian ruby 1.8.7 - 2	22.03885 ms	28.19817 ms	6.15932 ms	28%
Debian ruby 1.8.7 - 3	52.43772 ms	89.0935 ms	36.65578 ms	70%
Debian ruby 1.9.2 - 1	28.58125879 ms	54.18401913 ms	25.60276034 ms	90%
Debian ruby 1.9.2 - 2	21.93487545 ms	26.73286004 ms	4.79798459 ms	22%
Debian ruby 1.9.2 - 3	52.66458473 ms	84.95335805 ms	32.28877332 ms	61%
Výsledný průměr	39.8592721 ms	59.705173 ms	19.8459	51%

Tabulka 5.1: Výsledky výkonového testu

Z výsledků je zřetelné, že v testech na OS Debian se projevilo zrychlení mezi verzemi ruby 1.8.7 a ruby 1.9.2. Oproti tomu test spuštěný na Windows XP vrátil ve všech případech opačné výsledky. To co bylo na Windows rychlé je na Debianu pomalé a opačně.

Výsledným průměrem všech testů se zjistí, že knihovna je pro testované případy v průměru o 51% pomalejší než vytvoření stejného obrázku bez použití knihovny. Tento výsledek není úplně uspokojivý, ale má daleko k nejhoršímu očekávanému výsledku, což bylo 99%. Knihovna tak testem přošla s uspokojivým výsledkem.

5.4 Průběh nárůstu času v závisloti na velikosti obrázku

Vzhledem k tomu, že je knihovna velice komplexní a využívá i externí knihovny, tak nelze její složitost jednoduše vypočítat. Je ale možné zjistit přibližný průběh rychlosti růstu času pro vykonání.

5.4.1 Návrh testu

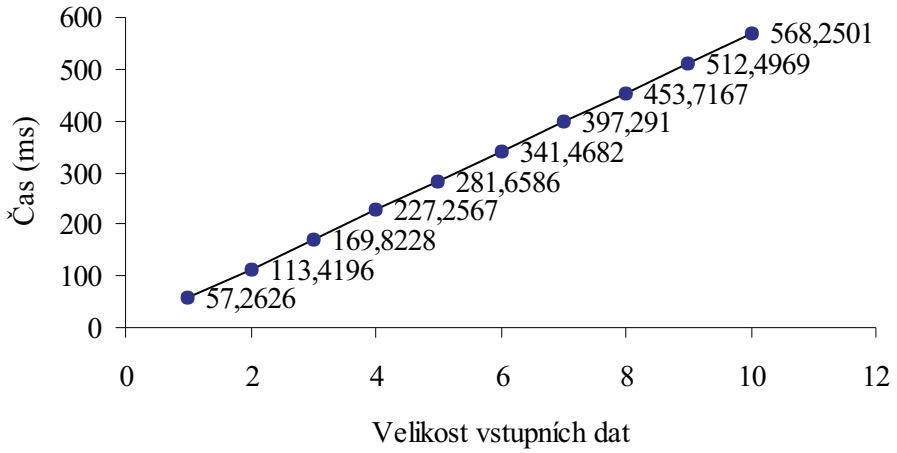
Pro test bude použit stejný program jako ve výkonovém testu. Bude vybrán ten, který dosáhl nejhoršího procentuálního výsledku, abychom zjistili nejhorší možný výsledek. Tím je TEST 1 pro konfiguraci Debian, ruby 1.8.7.

Tento test se bude brát jako základní vstupní jednotka. Testovat se bude na množině $N = \{1, 2, ..., 10\}$. Obrázek se bude na výšku zvětšovat a vykreslovat pod sebe tolikrát, kolik je dané N.

5.4.2 Provádění testu - výsledky

Po vykonání testu asym.rb vznikne následující výsledek.

```
N = 1   57.2626 ms
N = 2   113.4196 ms
N = 3   169.8228 ms
N = 4   227.2567 ms
N = 5   281.6586 ms
N = 6   341.4682 ms
N = 7   397.291 ms
N = 8   453.7167 ms
N = 9   512.4969 ms
N = 10  568.2501 ms
```



Obrázek 5.1: Průběh rychlosti růstu času

Z výsledku testu lze zjistit, že čas potřebný pro vykreslení obrázku narůstá lineárně v závislost na velikosti obrázku.

Kapitola 6

Závěr

- Popis možných perspektivních vylepšení
- Zhodnocení splnění cílů DP/BP a vlastního přínosu práce (při formulaci je třeba vzít v potaz zadání práce).
- Diskuse dalšího možného pokračování práce.

Literatura

[1] web:infodp. K336 Info — pokyny pro psaní diplomových prací. `https://info336.felk.cvut.cz/clanek.php?id=400`, stav ze 4. 5. 2009.

Příloha A

Seznam použitých zkratek

CSS Cascading Style Sheets

DTD Document Type Definition

GRASP General Responsibility Assignment SW Patterns

GUI Graphical User Interface

XML EXtensible Markup Language

Příloha B

Instalační a uživatelská příručka

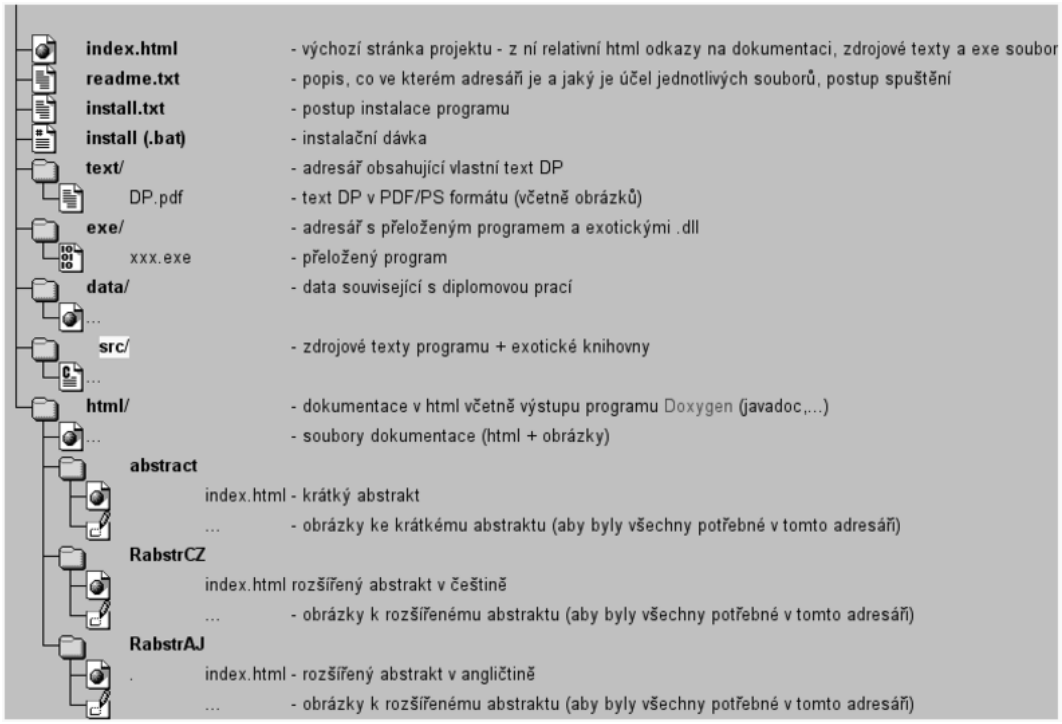
Jak sem asi tak napsat, ze to je vsechno v rdoc dokumentaci u zdrojovych kodu?...

Příloha C

Obsah přiloženého CD

Tato příloha je povinná pro každou práci. Každá práce musí totiž obsahovat přiložené CD. Viz dále.

Může vypadat například takto. Váš seznam samozřejmě bude odpovídat typu vaší práce. (viz [1]):



Obrázek C.1: Seznam přiloženého CD — příklad

Na GNU/Linuxu si strukturu přiloženého CD můžete snadno vyrobit příkazem:

```
$ tree . >tree.txt
```

Ve vzniklém souboru pak stačí pouze doplnit komentáře.

Z **README.TXT** (případne index.html apod.) musí být rovněž zřejmé, jak programy instalovat, spouštět a jaké požadavky mají tyto programy na hardware.

Adresář **text** musí obsahovat soubor s vlastním textem práce v PDF nebo PS formátu, který bude později použit pro prezentaci diplomové práce na WWW.