MULTIPAGE WITH MACLAN

# Introduction

# Areas and Slots

# Software Elements

# Templates

# Integration of Systems

# External Providers

# Automation of Tasks

# Introduction

Základní myšlenkou je umožnit vytváření a výměnu softwarových elementů. Aby bylo možné uvedenou myšlenku realizovat, je potřeba mít nějaké šablony a makrojazyk, jež umožní softwarové elementy vytvářet nebo už vytvořené používat. Softwarové elementy musí být nejlépe k dispozici s otevřeným zdrojovým kódem psaném v daném programovacím jazyce tak, aby je mohl preprocesor pracující se šablonami pomocí makrojazyka sestavovat do výsledných zdrojových kódů aplikací a případně je také překládat do spustitelné podoby.

Chci poděkovat oběma svým rodičům, tátovi podnikateli v oboru dopravy a mamince, která se celý život věnovala programování počítačů. Zároveň svému mladšímu bratrovi, který je softwarovým inženýrem. Oběma za možnost povídat si o softwaru.

Populárně:

Multipage IDE s makrojazykem Maclan je program, který umožňuje vytvářet právě takovéto softwarové elementy. A nejen to. Umožňuje jejich vzájemnou výměnu mezi aplikacemi. Jeho používání je výhodné z důvodu, aby se stejné nebo podobné části aplikací nemusely psát znovu a znovu pro nové softwarové projekty. Výše zmíněné přepisování jistě zabírá čas a energii nejen lidí, kteří se starají o vývoj aplikací, tedy hlavně programátorů, ale také strojů, na kterých vývojová prostředí běží.

Představte si například, že napíšete ve svém hlavním jazyce JavaScript malého asistenta, který uživatele provádí podle jeho polohy nejbližšími muzei v jeho okolí. Možná budete posléze chtít, aby vámi naprogramovaného asistenta nepoužívala pouze a jenom vaše aplikace, ale aby jej mohl do svého programu bez větších potíží zakomponovat také další programátor. Navíc budete chtít, aby bylo možné vašeho softwarového asistenta nastavit podle přání toho, kdo tuto vaši část softwaru převezme.



Obrázek Šablona pro softwarový element



Obrázek Příklad vložení softwarového elementu do šablony

Aby bylo možné takovýmto způsobem elementy vytvářet a vyměňovat, je důležité určit postup, jak to má počítač provádět. A k tomu je nutný odpovídající programovací jazyk nebo ještě lépe makrojazyk. Protože jsou v tomto případě elementy softwaru v podobě zdrojových kódů, a tedy vlastně obyčejného textu, měl by takovýto makrojazyk umět dobře pracovat se zdrojovými texty. Jistě již podobné makrojazyky existují, například makrojazyk M4 určený pro operační systém Unix nebo makrojazyk zabudovaný v překladačích C a C++ nebo podobně. Důležité je, aby pomocí těchto jazyků bylo možné z kousků programů sestavit zdrojové kódy, který je možné přímo spouštět na cílovém počítači nebo je přeložit do spustitelných modulů.

Jak ale uspořádat zmíněné části softwaru efektivně, tak aby tento proces byl co nejjednodušší a nejpřehlednější? Určitě by bylo možné použít běžné složky a soubory obsahující softwarové elementy spolu se šablonami, do kterých by zapadaly jako kostky stavebnice LEGO. Všimněte si, že k uvedenému jsou zapotřebí dvě staré konstrukce: soubory a složky, nebo případně propojení pomocí soft a hard linků. Bylo by ale možné použít namísto dvou starých jedinou novou konstrukci?

Odpověď může znít: musely by se nějakým způsobem sloučit složky a soubory. Nabízí se následující řešení. Namísto oddělených souborů a složek použít jednotné „oblasti se zásuvkami (sloty) pro údaje“.



Obrázek 3Nahrazení souborů a složek oblastmi

Stejně jako mohou existovat propojené soubory a složky (soft a hard links v Unixu), mohou také sloty odkazovat na data v jiné oblasti nebo dokonce v externích souborech či složkách. Na oblasti můžeme pohlížet jako by to byly datové struktury (*struct*) známé z běžných programovacích jazyků. Oblasti tedy mohou následně sloužit jako šablony pro softwarové elementy, o nichž se psalo v předchozích odstavcích.

Nový makrojazyk, který by měl umožňovat skládání aplikací ze šablon a softwarových elementů, by měl tedy umět pracovat se zdrojovými texty uloženými v oblastech nebo v místech, na které tyto oblasti odkazují. Do jisté míry by mohl připomínat hypertextový značkovací jazyk. Postup zpracování by mohl být následující:

1 – Nejprve zvolí programátor ručně „domovskou“ oblast odkud má začít preprocesor svoji práci.

2 – Jakmile se preprocesor spuštěn a nachází se v „domovské oblasti“, pokusí se nalézt startovací část programu napsanou v makrojazyce, od které se vše ostatní odvíjí. Preprocesor si v této chvíli inicializuje důležité pomocné stavy.

3 – Pak začne zpracovávat zdrojový text následujícím způsobem:

4 – Neustále kopíruje do výsledného souboru vstupní zdrojový kód do té doby, než najde nějakou značku makrojazyka. Podle názvu značky zjistí, jestli jde o jednoduchou značku, kterou může zpracovat ihned tak, že vloží na její místo výsledný text. Nebo o složenou značku, která může obsahovat znovu zdrojový či také další, teď již vnořené značky makrojazyka.

5 – Narazí-li tedy preprocesor na složenou značku, pak ji začne jako celek i s vnitřním obsahem zpracovávat. V pomocné paměti, v tak zvaném „zásobníku“ si udělá preprocesor nový záznam o místě značky a aktuálních pomocných stavech a začne provádět algoritmus rekurzivně s vnitřním textem značky a s aktualizovanými stavy, znovu od kroku 3. Výsledkem vždy přepíše celou zpracovanou značku makrojazyka. (Je jistě nutné zajistit, aby nedocházelo k nekonečným cyklům v programu makrojazyka, například také časovým omezením zpracovávání.)

6 – Preprocesor pracuje do té doby, dokud nalézá zdrojový text anebo nové značky. Pokud již nový text či značky nejsou, ukončí se a všechny zkopírované a zpracované zdrojové kódy zapíše do výsledného souboru, který je už možné spustit nebo přeložit. Může také iniciovat spuštění nového procesu od bodu 1 s jinou než „domovskou“ oblastí.

Jednoduché značky makrojazyka mohou pouze vkládat do výsledného textu zdrojového kódu údaje ze slotů v oblastech, z externích souborů a složek na disku nebo také z webových aplikací. Mohou vkládat globální identifikátory aktuálně zpracovávaných oblastí nebo třeba přímo jejich názvy. Naopak složité značky s vnitřním obsahem, které se skládají vždy z počáteční a koncové značky a vnitřního obsahu, mohou například N-krát tento obsah zopakovat (podobně jako cykly *for*).

Všechny jednoduché a složené značky makrojazyka a zdrojové kódy softwarových elementů a šablon, které jsou dostupné v oblastech, zpracovává preprocesor makrojazyka, který je nejlépe implementován jako HTTP server. Na dotaz směrovaný na zvolenou oblast, například „domovskou“ oblast, pak může spustit algoritmus uvedený výše v této kapitole a složit zdrojový kód vaší aplikace nebo jenom dodat potřebný softwarový element. Výsledek server tedy v tomto případě vrací zpátky pomocí HTTP protokolu. To je první varianta. Aplikace je přenášená přes tento protokol a spouštěna například v internetovém prohlížeči.

Server preprocesoru také může ukládat na podobný požadavek výsledky do složek a souborů na disku. Ty pak mohou být spuštěny nebo automaticky přeloženy do výsledných spustitelných souborů.

Poslední varianta použití souvisí s tím, že sloty oblastí mohou být připojeny ke zdrojovým kódům nebo textům umístěným v obyčejných souborech a složkách na disku. Může jít o celou externí aplikaci napsanou například v jazyce Java nebo C# a vytvořenou v prostředí Eclipse nebo Visual Studio. Externí soubory \*.java nebo \*.cs pak mohou obsahovat také jako přídavek všechny značky našeho makrojazyk, jaké byly uvedeny v předchozím textu a jejichž celý seznam je uveden v referenční příručce. Na dotaz provede server preprocesoru v souborech \*.java nebo \*.cs značky například tak, že namísto nich vloží do souborů softwarové elementy z oblastí, třeba dialogová okna napsaná v JFX nebo ve WPF. Opačným dotazem se na server preprocesoru se vrátí značky makrojazyka zpět do zapamatované zdrojové podoby, aby je mohl vývojář případně změnit a přenastavit.

# Areas and Slots

Pro lepší orientaci se v problémech je dobré používat množinovou abstrakci. Z tohoto pohledu je možné se dívat na oblasti se zásuvkami jako na množiny se svými prvky.

Například: ,

kde je oblast číslo 1; a pro



Obrázek 4Příklad oblasti s podoblastmi a sloty

Zápis předchozího vzorce není správný, protože v něm nezáleží na pořadí podoblastí , které jsou zde pouze jako prvky množiny . Podoblasti musí být uspořádané *M*-tice, protože svým uspořádáním určují postup zpracování softwarových elementů preprocesorem. Správný, trochu upravený množinový popis podoblastí by mohl být:

Popis oblasti pomocí množin je potom taky upravený do podoby:

Jelikož kterákoliv z oblastí může zároveň obsahovat mikro oblasti, mohlo by se v budoucnu stát, že zápis jejich struktury pomocí výrazů se závorkami, symboly a oddělovači, bude dosti nepřehledný. Jde totiž o souvislý zápis na jednom řádku v jednom směru, zleva doprava. Nebo chcete-li jednodimenzionální. Pro ilustraci lze uvést příklad takovéhoto nepřehledného zápisu, a to jen pro pár oblastí.

, kde , kde

, kde

nebo dokonce ještě méně přehledněji, bez explicitního uvedení názvů oblastí

Z výše uvedených důvodů je výhodnější přidat dodatečné rozměry, stupně volnosti, a využít 2D nebo 3D grafiku a zobrazit oblasti se zásuvkami, například na monitoru počítače, jako grafy na ploše nebo ve virtuálním 3D prostoru.



Obrázek 5Příklad dvourozměrného grafu

Z obrázku 5 plyne, že čím výš je oblast na ose , kterou můžeme nazvat „level“, tím více se stává oblast makro oblastí. A podobně, když je oblast více napravo ve směru osy , je umístěna dále v pořadí mikro oblastí u své makro oblasti. Kromě již uvedeného grafického zobrazení, jsou určitě možná ještě další. Na následujícím obrázku je uvedeno s velkou pravděpodobností to nejlépe realizovatelné pomocí dnešních GUI.



Obrázek 6Komponenty GUI pro zobrazení všech oblastí a souvisejících slotů

Na předchozím obrázku zobrazujícím možnou podobu GUI se po kliknutí na zvolenou oblast v grafické komponentě Tree View zobrazí vpravo v komponentě Table View sloty patřící do této oblasti.

Následující kapitoly se budou věnovat fungování preprocesoru tak jak je naprogramován v Multipage spolu s makro jazykem Maclan. Pro vysvětlení toho jak preprocessor a makro jazyk pracují, bude nutné používat grafy podobně, jako je tomu na obrázku 5. Při popisu činnosti preprocesoru se musíme nejprve zaměřit na pojem „inicializační oblast“. Jak víme, jsou programy zpracovávány procesory sekvenčně událost po události a zároveň v každém programu existují určité počáteční neboli „inicializační“ události. Preprocesor s makro jazykem není v tomto žádnou výjimkou. Aby mohl z oblastí sestavovat výsledný zdrojový kód a následně celou aplikaci, musí pokaždé někde v tomto svém postupu začít. Jedna z oblastí musí být proto „inicializační“. Je to podobné jako u webových aplikací, kde existuje například domovská stránka index.html, která je nalezena při dotazu na danou WWW adresu a obsahuje úvodní funkce. Podobně u aplikací napsaných v C++ je „inicializační“ událostí nalezení a spuštění funkce main (). Makro jazyk potřebuje také takovouto „inicializační“ část, stejně jako ostatní programovací jazyky. Na rozdíl od výše uvedeného jazyka C++ a jeho funkce main (), která je vstupním bodem aplikace, jde u programu Multipage o inicializační „resource“ (tuto datovou strukturu popisuje poznámka níže), který je zpracován preprocesorem na úplném začátku celého procesu. Pro ilustraci je na následujícím obrázku v diagramu zobrazen vstupní bod, „inicializační“ oblast spolu se sloty a červeným inicializačním resource.



Obrázek 7 Graf s inicializační oblastí A1

Poznámka: pro doplnění systému oblastí a slotů je potřeba zmínit ještě další, spíše technickou, záležitost. Sloty oblastí odkazující na data a zdrojové kódy aplikací by měly být postačující k fungování celé aplikace Multipage, nicméně v některých případech se ukazuje vhodné použít místo slotů o něco pozměněné datové struktury, tzv. „resources“. A to v těch případech, kdy musíme mít k dané oblasti připojené binární nebo textové soubory se kterými chceme pracovat jako s celkem. Mohou to být bitmapové nebo vektorové obrázky, videa v různých formátech nebo také například instalační soubory aplikací pro Windows, Linux a jiné operační systémy. Tyto resources jsou uloženy odděleně od oblastí a slotů. Oblasti na ně odkazují pomocí lokálních názvů platných pouze v daných oblastech nebo pomocí globálních názvů těchto uvedených resources. Inicializační části pro spuštění preprocesoru Multipage jsou uloženy jako resources.

Abychom se dostali dále v popisu toho, jak preprocesor začíná zpracovávat zdrojové kódy, bude nutné ještě zmínit další vlastnosti, jenž má systém oblastí a slotů. Jednou z těchto vlastností je dědičnost. Ta umožňuje v oblasti použít informace ze souvisejících oblastí (nadoblastí nebo podoblastí) aniž by se musel preprocesor v těchto oblastech aktuálně při zpracovávání nacházet. Algoritmus nalezení například slotu v oblastech, jež jsou v relaci s aktuální zpracovávanou oblastí, může být zvolen programátorem píšícím v makrojazyku Maclan. Kupříkladu následující algoritmus:

1 – Nejprve najdi daný slot přímo v aktuální oblasti.

2 – Nebyl-li nalezen, pak hledej uvedený slot v makro oblastech umožňujících dědění slotů.

3 – V případě, že ani takto nebyl slot nalezen, hledej jeho výskyt v mikro oblastech v první úrovni vnoření.

Předchozí algoritmus kombinuje použití informace z makro objektů s hledáním obdobné informace v mikro objektech. Vlastnosti oblasti tedy může ovlivnit to, ve které šabloně se oblast nachází a také jaký konkrétní mikro element oblast aktuálně obsahuje. Následující graf zobrazuje postup získání informace ze slotu podle uvedeného algoritmu.



Obrázek 8 Kombinace kroků 1,2 a 3 při hledání slotu

Skutečnost, že lze ve směru některé hrany informaci dědit, může odpovídající hrana povolit nebo zakázat. Podobně mohou také sloty v oblastech, které jsou v relaci s aktuální oblastí, povolovat nebo zakazovat preprocesoru své nalezení. Z toho plyne, že ne všechny hrany (relace) umožňují dědění vlastností. Následující obrázek ukazuje zákaz dědění z makro oblasti do aktuální oblasti .



Obrázek 9 Omezení dědičnosti z oblasti A2

V předchozím grafu má tedy oblast zakázáno dědit z oblasti a k ní navazujících makro oblastí. Poslední obrázek je sice složitý, ale také vyčerpávajícím způsobem prezentuje to co lze říci o dědění informací v oblastech.

Makrojazyk Maclan umožňuje pomocí parametru *level* určit, kolik úrovní makro nebo mikro oblastí může být maximálně použito při hledání zvoleného slotu. Není-li *level* uveden, provádí preprocesor hledání v makro oblastech až po speciální oblast označenou jako *Project*. Je to z toho důvodu, že základní makro oblasti mohou obsahovat celé aplikace nebo chcete-li softwarové projekty, jejichž instance mohou být dokonce mnohonásobně zopakovány v dané základní oblasti jako „klony“ nebo kopie téže softwarové aplikace.

Sloty oblastí obsahují atribut *access*, který určuje zda jsou děditelné v oblastech. Tento atribut může nabývat hodnot *true* a *false*. Má-li hodnotu *false*, pak se pro daný slot dědičnost nikdy nepoužije. Příkladem může být slot obsahující identifikátor tlačítka GUI, jenž není vhodné dědit v jiných ovládacích prvcích kvůli jeho požadované jednoznačnosti v celé aplikaci.