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

Bakalářská práce

Migrace grafického rozhraní z MAF2 na MAF3

Plzeň 2013 Lukáš Kroupa

Prohlášení

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

V Plzni dne 1. května 2013

Lukáš Kroupa

Abstract

This work is about examining how graphical user interface is implemented in applications based on MAF2 and MAF3 platform and exploring possibilities of automatic or semi-automatic conversion of its source code from MAF2 to MAF3. User interface of MAF2 is based on wxWidgets library and Qt library is used as base of MAF3. Differences between these libraries and user interface features of both platforms are described as well as possibilities of automatic source code editation.

The transformation tool is implemented in Python programming language and its output is evaluated based on transformation of modules from MAF2, MAF2 Medical and LHPApp projects. Results are supplemented by short overview of encountered problems.

Poděkování

Tímto bych chtěl poděkovat vedoucímu bakalářské práce docentu Ing. Josefu Kohoutovi, Ph.D. za zajímavé zadání práce, vynaložený čas, ochotu a užitečné rady.

Dále bych chtěl poděkovat Dr. Danieli Giunchimu z italské společnosti SCS Inc. za podněty a jedinečné informace, které mi jako vývojář platformy MAF poskytl.

Obsah

1 Úvod

Cílem této práce je prozkoumat a popsat rozdíly v architektuře platforem MAF2 a MAF3, zejména v komponentách spojených s tvorbou grafického uživatelského rozhraní a navrhnout nástroj pro automatický či poloautomatický převod zdrojového kódu grafického rozhraní do novější verze.

Projekt MAF (Multimod Application Framework) je zaměřený na usnadnění a zrychlení vývoje aplikací pro počítačovou podporu v medicíně a dalších vědních oborech, které vyžadují zpracování a zobrazení heterogenních dat. Platforma je vyvíjena ve spolupráci několika institucí, mezi hlavní patří BioComputing Competence Centre, The Rizzoli Institute, CINECA Supercomputing centre a University of Bedfordshire. Cílem projektu je integrovat technologie umožňující operovat s různorodými daty, především jejich zobrazení, úpravu, segmentaci a tvorbu vazeb mezi daty z více zdrojů, do jediné platformy, na které bude možné rychle a efektivně tvořit vertikálně strukturované aplikace.

Platforma MAF3 je vyvíjena s cílem odstranit nedostatky předchozí verze, například v oblasti podpory vícevláknového zpracování dat a skriptovacích jazyků, a usnadnit další vývoj podporou zásuvných modulů. Na platformě MAF2 je založeno množství aplikací, které by mohly využít nové technologie dostupné v MAF3, ale jejich převod na novou platformu je finančně a časově nákladný. Realizací nástroje umožňujícího automatickou či poloautomatickou úpravu zdrojového kódu by byl převod aplikací na modernější platformu usnadněn.

Tato práce byla zahájena s předpokladem, že platforma MAF3 obsahuje funkcionalitu ekvivalentní s platformou MAF2, avšak v průběhu realizace práce byla zjištěna nekompletní implementace funkcionality grafického uživatelského rozhraní, které je stěžejní pro tuto práci. V této práci budou rozebrány rozdíly v implementaci grafického uživatelského rozhraní i s ohledem na nekompletní implementaci v MAF3.

V rámci práce budou dále představeny knihovny *Qt4* a *wxWidgets*, na kterých jsou postaveny jednotlivé verze platformy MAF, prozkoumány možnosti syntaktické analýzy zdrojového kódu v jazyce C++ a navržena implementace nástroje usnadňujícího převod grafického uživatelského rozhraní projektů založených na MAF2 do MAF3.

2 Architektura platformy MAF

Aplikace založené na MAF se skládají z komponent, které umožňují zpracovat data a poskytují služby pro práci s těmito daty. Cílem je umožnit komponentově orientované programování, kdy programátor může zvolit z již hotových nebo vlastních modulů. Programátor nakonfigurováním vstupů a výstupů jednotlivých komponent a jejich souslednosti může přidávat a upravovat funkcionalitu aplikace.

Spolu s knihovnami uživatelského rozhraní využívá platforma také knihovny *VTK*, *ITK* pro zpracování a vizualizaci dat a knihovny pro podporu datových přenosů a šifrování dat. Podrobný popis architektury platformy MAF je dostupný z webových stránek projektu na portálu *Biomed Town* [10].

2.1 Přehled architektury MAF

Platforma MAF poskytuje moduly typu *VME* (*Virtual Medical Entity*) pro správu a uložení dat, typu *Views* pro zobrazení a analýzu dat, dále typu *Operations*, které umožňují úpravu či tvorbu nových dat a typu *Interaction* poskytující podporu pro vstupní/výstupní zařízení. Aplikace založené na platformě MAF mohou implementovat vlastní moduly uvedených typů s vlastní specializovanou funkcionalitou. Každá komponenta je instance určité třídy jazyka C++ a samotná aplikace je instance třídy *Logic*, která určuje aktivní komponenty a zajišťuje jejich komunikaci, konzistentní stav a jejich integraci do grafického uživatelského rozhraní. Jednotlivé komponenty určitého typu spravuje vždy třída *Manager* daného typu, přesněji *VME Manager*, *Views Manager*, *Operations Manager* a *Interaction Manager*.

Operation - *Operations* jsou rozděleny na tři kategorie typů modulů: *Importer*, *Exporter* a *Operation*. První dva zmíněné typy se starají pouze o získávání a export dat, poslední typ umožňuje vytvářet a modifikovat data. Každý modul typu *Operation* umožňuje zpracovat jen určitý typ *VME*. V průběhu operace nelze *VME* měnit a nelze spustit další operaci. Průběh operace lze ovlivnit konfigurací parametrů a každá operace by měla podporovat akci zpět a vpřed pro zrušení či opětovné obnovení transformace provedené danou operací.

VME - *VME Manager* uspořádává jednotlivé *VME* do hierarchického stromu, každé *VME* se skládá z časově proměnného datového souboru a matice, která určuje polohu vzhledem k nadřazenému *VME* ve stromu *VME* a metadata obsahující textové atributy. *VME Manager* dále umožňuje přidávat a odebírat prvky stromu, umožňuje ukládat a načítat stav stromu ze souboru.

View - Modul *View* zobrazuje interaktivní reprezentaci stromu *VME*. Pro každý *View* je udržován seznam *VME*, které budou zobrazeny a seznam kompatibilních typů *VME*, které mohou být v jedné instanci zobrazeny. *View Manager* umožňuje vytvářet a odstraňovat pohledy a zaznamenává, který pohled je aktuálně aktivní.

Jednotlivé moduly jsou základní prvky každé aplikace a poskytují její funkcionalitu. Moduly, zejména typu *Operation* a *VME*, mohou implementovat funkce použitelné ve více aplikacích. Migrováním modulů vytvořených pro platformu MAF2 na aktuální verzi MAF3 bude rozšířena možnost použití modulu v další aplikaci či převedení stávající aplikace na modernější platformu.

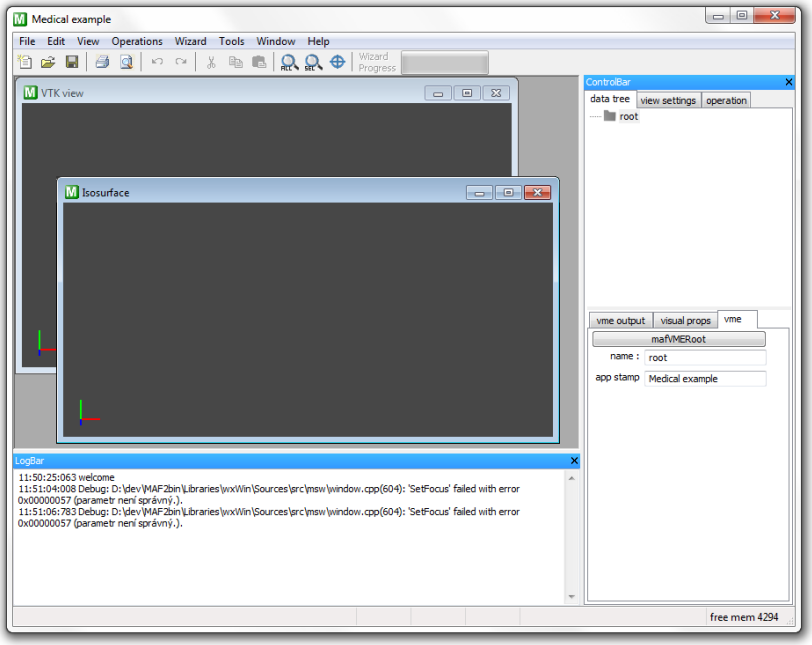
2.2 Grafické rozhraní MAF

Aplikace založené na platformě MAF mají výchozí hlavní okno s nabídkou, panelem nástrojů a postranním panelem, ve kterém mohou jednotlivé moduly zobrazovat ovládací prvky či svůj stav. Hlavní část okna aplikace je určena pro zobrazení dat poskytnutých aktivními *Views*. Vzhled a rozvržení tohoto okna je definován platformou MAF, konkrétní aplikace mohou do okna doplňovat položky nabídky a zobrazovat grafické rozhraní modulů v postranním panelu. Moduly dále mohou zobrazovat dialogová okna obsahující grafický náhled dat nebo náhled prováděné operace a odpovídající ovládací prvky.

Grafické uživatelské rozhraní jednotlivých modulů je závislé na verzi platformy MAF a použité knihovně uživatelského rozhraní. Platforma MAF2 využívá k tvorbě uživatelského rozhraní knihovnu *wxWidgets* a vlastní obalující třídy, které usnadňují definici vzhledu a rozvržení ovládacích prvku. V platformě MAF3 byla knihovna *wxWidgets* nahrazena knihovnou *Qt4*.

2.3 Grafické rozhraní MAF2

O grafickou reprezentaci modulů se stará třída `mafGui`, která obaluje rozhraní knihovny *wxWidgets* a poskytuje předdefinované bloky grafického rozhraní. Metody třídy `mafGui` poskytují prvky rozhraní *wxWidgets* s předkonfigurovanými vlastnostmi a v již sestavených blocích, které umožňují jednotné zobrazení grafického rozhraní bez nutnosti podrobné konfigurace každého jednotlivého prvku. Blok rozhraní obsahuje jeden nebo více prvků, například popisek a aktivní prvek typu posuvník nebo pole textového vstupu. Standardní vzhled okna aplikace MAF2 je znázorněn na obrázku 2.1.



Obrázek 2.1: Okno aplikace postavené na platformě MAF2.

Platforma MAF2 implementuje několik vlastních grafických prvků se specializovanou funkcí, které nejsou dostupné ve výchozí sadě prvků *wxWidgets*. Tyto prvky nemají zatím v platformě MAF3 ani v knihovně *Qt4* vhodnou alternativu.

Pokud je modul typu *Operation*, je grafické rozhraní modulu implementováno v metodě volané z metody `OpRun`. V ostatních modulech je metoda definující uživatelské rozhraní volána přímo. Tato metoda je standardně pojmenována `CreateGui`, pokud implementuje grafické rozhraní s ovládacími

prvky v postranním panelu, nebo `CreateOpDialog`, pokud obsahuje implementaci dialogového okna.

V uvedené metodě je ve většině modulů vytvořena instance třídy `mafGUI`, respektive `mafGUIDialog`, která obsahuje prvky rozhraní uložené ve stromové struktuře. V ojedinělých případech jsou z této metody volány další metody které vytvářejí grafické rozhraní. Prvky jsou do stromové struktury vkládány metodami objektu `mafGUI` s požadovanými parametry, tyto metody zajistí konfiguraci prvku, vytvoření funkčního bloku z více prvků a jejich připojení na první úroveň stromu. Příklad definice jednoduchého postranního panelu je uveden ve fragmentu kódu 2.1.

```
1  void customMafOperation::CreateGui()
2  {
3    m_Gui = new mafGUI(this); // objekt grafickeho rozhrani
4    m_Gui->SetListener(this);
5
6    m_Gui->Bool(ID_BOOL , "Boolean variable", &m_bool);
7    m_Gui->Divider(1); // oddelovaci cara
8    m_Gui->Label("Label text", true); // textovy popisek
9    m_Gui->Button(ID_BUTTON, "Button text"); // tlacitko
10   m_Gui->Divider(2);
11   m_Gui->Label("Next label", true);
12
13   m_Gui->OkCancel(); // tlacitka OK a Cancel
14 }
```

Fragment kódu 2.1: Definice jednoduchého rozhraní postranního panelu.

Dialogová okna modulů lze definovat přímo jednotlivými objekty knihovny *wxWidgets*. Tímto způsobem je možné prvky vkládat na libovolnou pozici stromu a vytvořit komplexnější uživatelské rozhraní než je možné dosáhnout použitím metod třídy `mafGui`. Oba způsoby definice grafického rozhraní je možné kombinovat. V metodě `CreateGui` lze vytvořit více instancí třídy `mafGUI`. Do takových objektů budou připojeny prvky voláním metod třídy `mafGUI` a následně je možné celý objekt se stromem prvků vložit jako podstrom do instance třídy `mafGUIDialog`. Tento postup je ilustrován ve fragmentu kódu 2.2.

```
void customMafOperation::CreateOpDialog()
{

  /*Inicializace objektu mafGUIDialog a mafGUI.*/
  m_Dialog = new mafGUIDialog("Dialog Title", mafRESIZABLE);

  m_GuiDialog = new mafGUI(this);
  m_GuiDialog->Reparent(m_Dialog);

  /* Inicializace prvku knihovny wxWdgets. */
  wxBoxSizer * sizer = new wxBoxSizer(wxHORIZONTAL);

  m_checkBoxShowData = new wxCheckBox( m_Dialog,
      ID_CHECK_SHOW_DATA, wxT("Show data"), wxDefaultPosition,
       wxDefaultSize, 0 );

  sizer->Add(m_checkBoxShowData);

  /* Prvky wxWidgets jsou pripojeny do objektu tridy
     mafGUIDialog. */
  m_Dialog->Add(sizer,1);

  /* Do objektu mafGUI jsou pridany prvky metodami tridy. */
  m_GuiDialog->Label("Label");
  m_GuiDialog->Button(ID_BUTTON,"Button text","");
  m_GuiDialog->Integer(ID_Integer,"Integer:",&m_Number,0,500,
     "Widget tooltip");

  /* Objekt tridy mafGUI je pripojen do stromu prvku objektu
     mafGUIDialog. */
  m_Dialog->Add(m_GuiDialog);
}
```

Fragment kódu 2.2: Definice uživatelského rozhraní s použitím wxWidgets a mafGUI.

Řízení a zachytávání událostí je implementováno třídou *mafEvent*. Zpráva reprezentující událost z libovolného zdroje včetně grafického rozhraní je implementována jako objekt zděděný z třídy `mafEventBase`. Zpráva obsahuje svůj identifikátor, identifikátor odesílatele a kanál, pomocí kterého lze zprávu zachytit. Zprávy mohou být zachyceny jedním nebo více různými příjemci. Objekty, které chtějí přijímat zprávy, musí být potomkem třídy `mafObserver` a implementovat metodu `OnEvent()`. Každý modul obsahující grafické rozhraní je potomkem třídy `mafObserver` a v metodě `OnEvent()` jsou zachyceny a zpracovány také události uživatelského rozhraní.

2.4 Změny v platformě MAF3

Platforma MAF3 vznikla z potřeby odstranit nedostatky v architektuře MAF2 a usnadnit správu a další vývoj platformy. Cíle nové verze platformy byly představeny v prezentaci *MAF3: the story so far* [16]. Architektura platformy je ve verzi MAF3 zjednodušena a projekt se zaměřil pouze na biomedicínské využití.

MAF3 zachovává základní body návrhu platformy MAF jako je použití specializovaných modulů a jejich rozdělení do kategorií *Interaction*, *Operation*, *View*, *VME*. Jádro platformy má stabilní programové rozhraní, byla zavedena podpora zásuvných modulů, které umožňují snazší rozšíření platformy a část funkcionality z jádra MAF2 byla do těchto modulů přesunuta. Nová architektura umožňuje použití skriptovacích jazyků, lepší podporu vícevláknového výpočtu a výpočtu pomocí GPU. Struktura platformy MAF3 je znázorněna na obrázku 2.2.

Platforma MAF3 je založena na knihovně *Qt4*, která umožňuje jednodušší tvorbu grafického rozhraní a poskytuje rozsáhlou kolekci aplikačních tříd. V projektech postavených na knihovně *Qt4* jsou všechny objekty potomky třídy `QObject`, pro podporu technologií *Qt* jsou použity makra a jazyk C++ je rozšířen o nové příkazy.

Předávání dat mezi jednotlivými moduly aplikace se provádí pomocí signálů a slotů poskytnutých *Qt4* knihovnou a pomocí komunikační sběrnice *mafEventBus*, která podobně jako v MAF2 umožňuje vytvořit kanály pro posílání zpráv a pro daný kanál registrovat více příjemců. Jednotlivé kanály jsou označeny identifikátorem, složeným z názvů domén oddělených tečkami. Kořenová doména je uvedena nejvíce vlevo, celý identifikátor může vypadat například takto: `maf.local.resources.view.create`. Popis mechanismu předávání zpráv v platformě MAF3 je popsán na webových stránkách projektu (viz [9]). Třída *mafEventBus* tvoří spolu s *mafCore* základ aplikace a poskytují služby ostatním modulům.

Moduly *Interaction*, *Operation*, *View* a *VME* jsou implementovány jako objekty třídy *mafResource*. Jednotlivé objekty jsou zcela nezávislé a s využitím komunikační sběrnice mohou pracovat lokálně v samostatných vláknech nebo na samostatných počítačích. Platforma je dále tvořena následujícími moduly:

mafSerialization - Zajišťuje serializaci a deserializaci dat z objektů *mafRe-*

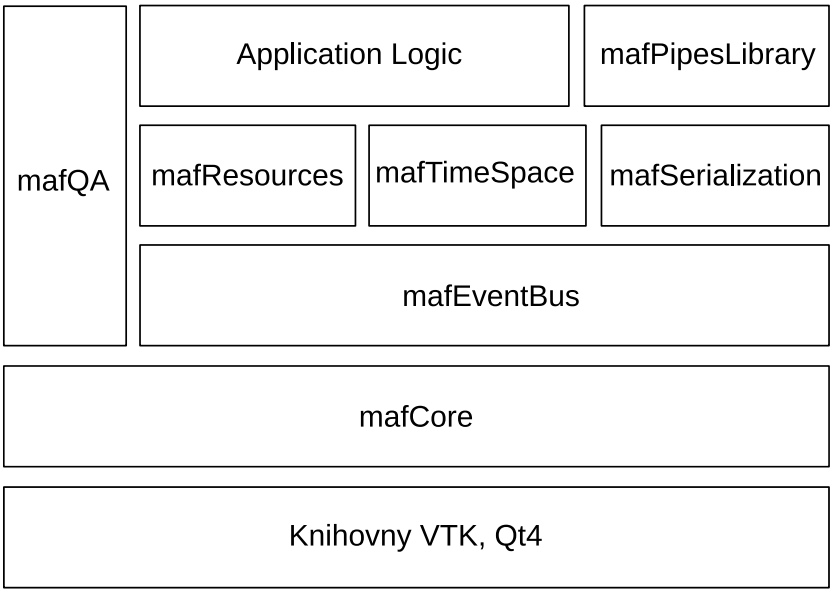
source do specifikovaného formátu.

mafQA - Poskytuje funkcionalitu automatických testů a profilování jednotlivých modulů.

mafTimeSpace - Generuje diskrétní báze času a prostoru pro operace a umožňuje nad nimi provádět lineárně algebraické operace.

mafPipesLibrary - Integrují knihovny pro vizualizaci a zpracování dat ve formě zásuvných modulů.

Application Logic - Reprezentuje konkretní aplikaci postavenou na platformě MAF3 a umožňuje zvolit aktivní moduly a jejich konfiguraci.

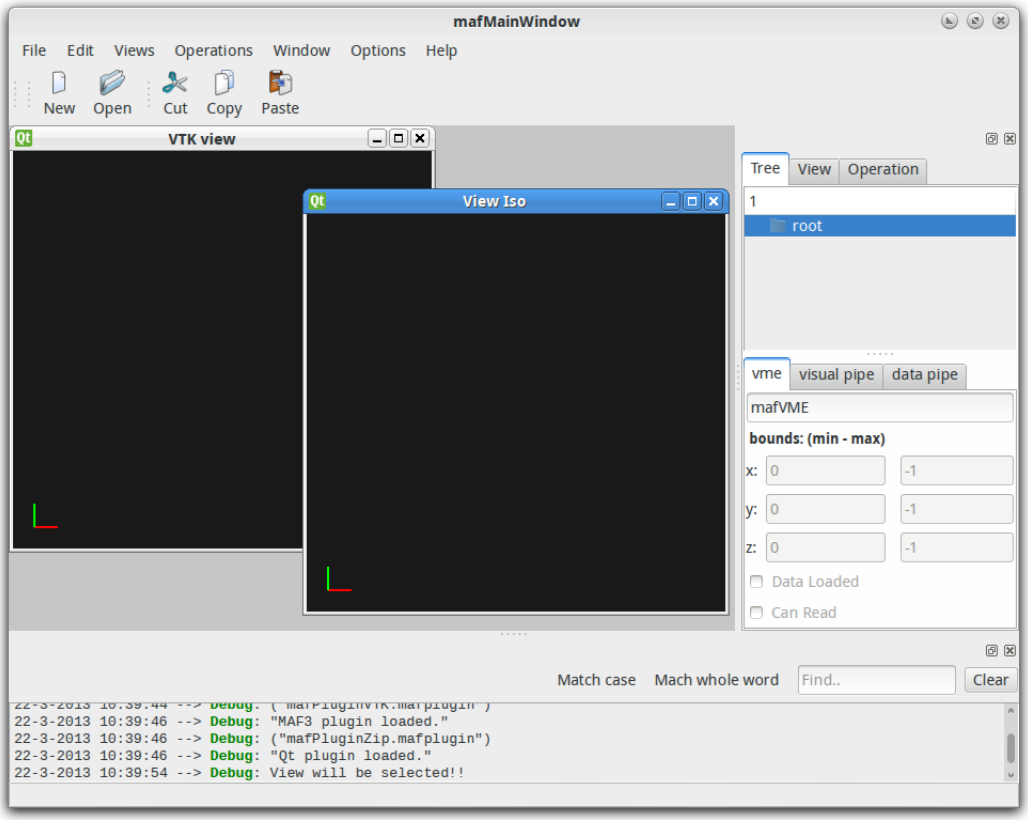


Obrázek 2.2: Diagram architektury platformy MAF3.

2.5 Grafické rozhraní MAF3

Grafické uživatelské rozhraní je v MAF3 implementováno pomocí knihovny *Qt4*. Rozhraní lze definovat pouze prostředky poskytnutými nástroji knihovny *Qt4*, platforma již neobsahuje žádně obalující třídy a metody pro definici rozhraní. Výchozí sada grafických prvku knihovny *Qt4* je rozšířena o prvky

rozhraní implementované v rámci MAF3 se specializovanou funkcí. Mezi tyto prvky se řadí například `mafLoggerWidget`, který je určen k záznamu událostí aplikace a umožňuje jejich rozdělení do kategorií podle závažnosti, nebo `mafFindWidget`, který implementuje vstupní pole s volbami vyhledávání. Hlavní okno aplikace založené na MAF3 je uvedeno na obrázku 2.3



Obrázek 2.3: Okno aplikace postavené na platformě MAF3.

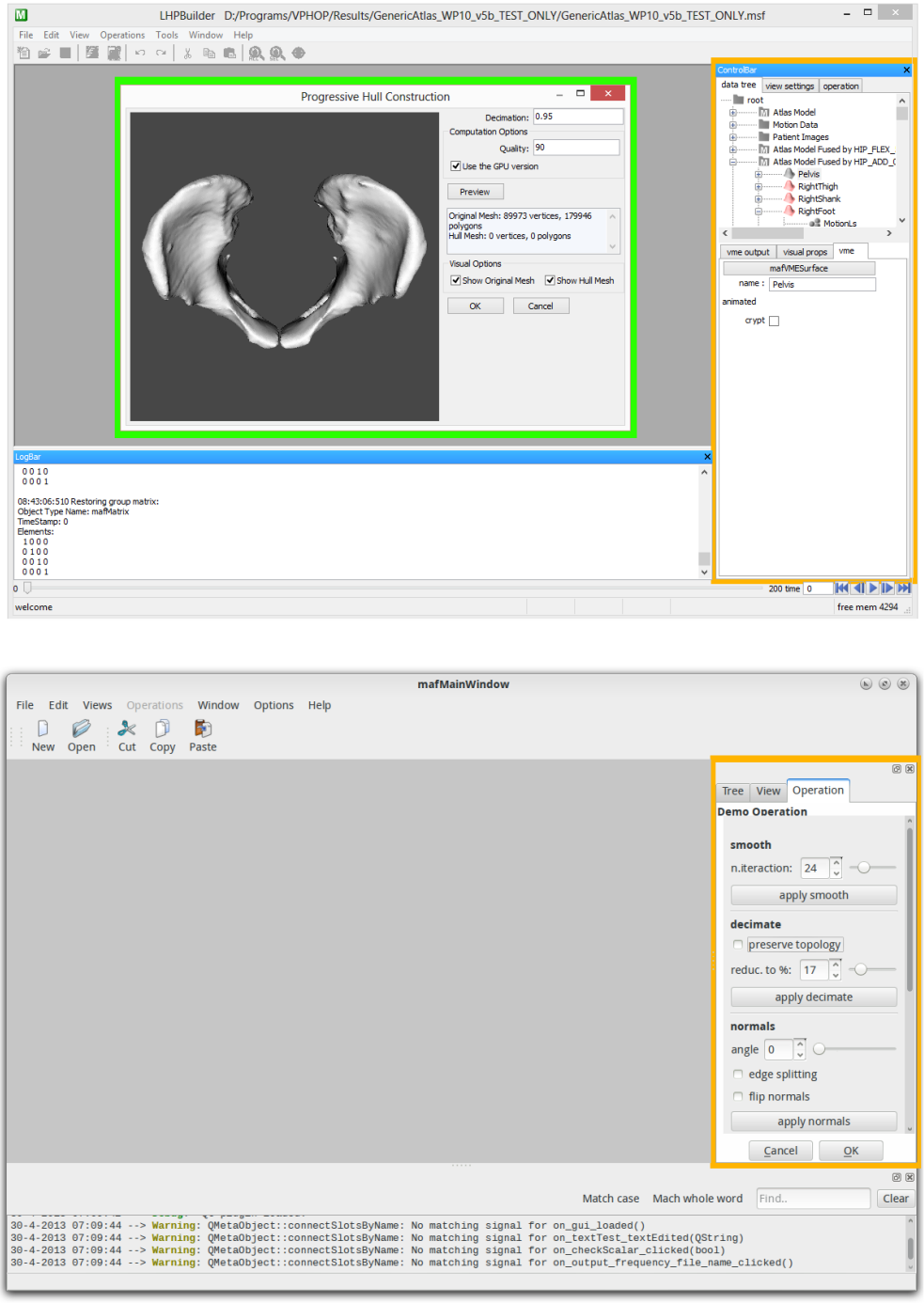
Modul může obsahovat grafické rozhraní postranního panelu, ale podpora zobrazení dialogových oken modulů není v době realizace této práce implementována. Moduly platformy MAF2, které obsahovaly uživatelské rozhraní v této formě, nebude možné převést na platformu MAF3.

Uživatelské rozhraní modulu je načteno z dokumentu XML specifikovaného v konstruktoru modulu, definice rozhraní je zpracována ve třídě `mafGUIManager`, která automaticky zjistí typ modulu a inicializuje rozhraní včetně propojení signálů a slotů. Aby byl slot správně rozpoznán a propojen s vhodným signálem, musí název slotu být ve formátu `on_puvodSignalu_signal()`.

Například pro slot, který reaguje na klepnutí na tlačítko s názvem `akce`, bude název slotu `on_akce_clicked()`.

Definice rozhraní modulu přímo ve zdrojovém kódu modulu není podporována, implementace grafického rozhraní je striktně oddělena od logiky modulů. Jak se v průběhu řešení ukázalo, z tohoto důvodu není možný přístup k prvkům uživatelského rozhraní z metod modulu, avšak tato funkcionalita je v modulech MAF2 často využívána a nebude je proto možné bez úprav zdrojového kódu MAF3 správně převést.

Na obrázku 2.4 je uvedeno porovnání oken aplikací postavených na platformě MAF2 a MAF3. V okně aplikace MAF2 je zobrazeno také dialogové okno pro nastavení parametrů operace. Jak je uvedeno výše, okna tohoto typu nejsou v aktuální verzi MAF3 podporována.



Obrázek 2.4: Možnosti GUI v MAF2 a MAF3 - verze z 24.1.2013.
Oranžovou barvou je zvýrazněn postranní panel aplikace.
Zelenou barvou je zvýrazněno dialogové okno aplikace MAF2.

3 Knihovny Qt a wxWidgets

Knihovna grafického uživatelského rozhraní je základní prvek moderních aplikací, který z velké části ovlivňuje funkčnost celého programu, nejen z hlediska grafické reprezentace, ale může také určovat do jisté míry architekturu aplikace. Knihovny uživatelského rozhraní jsou ve většině případů součástí sady knihoven, která poskytuje podporu více vláken, práci se sítí či různými datovými formáty. Použití všech knihoven ze sady není nutné, ale vývoj aplikace je tím znatelně usnadněn. Nevýhodou je, že pokud sada knihoven nesplňuje požadavky aplikace, například podporu kódování znaků evropských či asijských jazyků, je nutné nahradit všechny knihovny.

3.1 Použité GUI knihovny

Každá z uvažovaných verzí platformy MAF je založena na rozdílné knihovně pro tvorbu GUI. MAF2 je založen na knihovně *wxWidgets* ve verzi 2.6, MAF3 na knihovně *Qt4* verze 4.7. Obě knihovny mají svobodný otevřený zdrojový kód, jsou multiplatformní, vytvořené v jazyce C++, obě poskytují množství grafických prvků pro tvorbu uživatelského rozhraní a dodržují jejich nativní vzhled na všech podporovaných platformách. V obou knihovnách je implementováno množství pomocných tříd, například pro práci se sítí, časem a souborovým systémem. Knihovny jsou ale odlišné ve způsobu definice GUI a obsluhou událostí uživatelského rozhraní propojením jednotlivých prvků rozhraní s ostatními funkcemi programu. Následující popis knihoven se zaměřuje převážně na vlastnosti spojené s tvorbou uživatelského rozhraní aplikací.

3.2 wxWidgets

Knihovna *wxWidgets* vznikla v roce 1992 s cílem umožnit tvorbu multiplatformních uživatelských rozhraní, které vyžadují minimální nebo žádné změny v kódu při přenosu mezi jednotlivými platformami. Projekt založil na *University of Edinburgh* Julian Smart [5]. V projektu MAF2 je použita verze 2.6, uvedená v roce 2005. Aplikační rozhraní *wxWidgets* je v mnoha směrech podobné aplikačnímu rozhraní *Microsoft Foundation Class Library* (MFC), tedy knihovně zajišťující grafické uživatelské rozhraní operačního systému

Windows. Díky této podobnosti je snazší konvertovat aplikace ze systému Windows na jiné platformy použitím *wxWidgets* než použitím jiné knihovny podobného zaměření.

Knihovna je vyvíjena s důrazem na zpětnou kompatibilitu, obsahuje velké množství maker a zastaralých jazykových konstrukcí, převzatých z původních verzí knihovny. Výhodou knihovny je podpora překladu pomocí téměř všech existujících překladačů a podpora mnoha operačních systémů a jejich grafických subsystémů, skrze které vykresluje jednotlivé prvky. Knihovna používá nativní grafické rozhraní systému na rozdíl od většiny multiplatformních knihoven, které vykreslují vlastní prvky a pouze napodobují vzhled použité platformy[17].

3.2.1 Základní prvky aplikace s použitím wxWidgets

Základními prvky *wxWidgets* aplikace jsou třídy `wxApp` a `wxFrame`. Od třídy `wxApp` je oděděna vlastní třída, která umožňuje nastavit či získat vlastnosti celé aplikace a zpracovat události, které nebyly zpracovány ostatními objekty v aplikaci. Pomocí metody `OnInit()` se zahájí vykonání programu, jedná se tedy o vstupní bod programu analogicky k funkci `main()`, která není pro programátora při použití *wxWidgets* přístupná. Každá aplikace musí obsahovat makro `IMPLEMENT_APP()`, které vytvoří funkci `main()` a instanci potomka třídy `wxApp`.

```
MyFrame::MyFrame(const wxString& title, const wxPoint& pos,
    const wxSize& size)
: wxFrame( NULL, -1, title, pos, size )
{
    wxMenu *menuFile = new wxMenu; // menu aplikace
    menuFile->Append( ID_Quit, _("E&xit") );

    wxMenuBar *menuBar = new wxMenuBar; // lista menu
    menuBar->Append( menuFile, _("&File") );
    SetMenuBar( menuBar );

    wxString  text = wxT("Label"); // textovy popisek
    wxStaticText *st = new wxStaticText(this, wxID_ANY, text,
      wxPoint(10, 10), wxDefaultSize, wxALIGN_CENTRE);
}
```

Fragment kódu 3.1: Definice okna aplikace s využitím knihovny wxWidgets.

Potomek třídy `wxFrame` reprezentuje vlastní okno aplikace. Jednotlivé prvky grafického rozhraní se organizují a rozmisťují v ploše okna samostatně, nebo pomocí prvku třídy `wxSizer`. Tento prvek vytvoří oblast, která může obsahovat prvky rozhraní, ale také další prvky typu `wxSizer`. Nastavením parametrů můžeme ovlivnit, jak budou prvky prostorově rozmístěné a jakým způsobem se bude měnit jejich rozložení při změně velikosti okna. Implementace jednoduchého okna s nabídkou a textovým popiskem je uvedena ve fragmentu kódu 3.1.

Akce vykonané po aktivaci jednotlivých prvků rozhraní jsou definovány v tabulce událostí. Tabulka musí být deklarována v hlavičkovém souboru v potomku třídy `wxEventHandler`, kterým je skrze třídu `wxWindow` každý prvek grafického rozhraní. Tabulka je pak definována v implementačním souboru, příklad definice tabulky je uveden ve fragmentu kódu 3.2. V hlavičce tabulky je uveden název třídy, pro kterou je tabulka implementována, a rodič této třídy. Tabulka obsahuje identifikátor události, identifikátor ovládacího prvku a funkci, která bude s událostí spojena. V případě vyvolání události se prohledává nejprve tabulka událostí prvku, který událost vyvolal a pokud není zachycena, prohledávají se předkové až na úroveň okna. Pokud není událost zachycena žádným oknem, je předána objektu aplikace, který je potomkem třídy `wxApp`.

```
1 BEGIN_EVENT_TABLE(MyFrame, wxFrame)
2     EVT_MENU(wxID_EXIT, My::OnExit)
3     EVT_SIZE(MyFrame::OnSize)
4     EVT_BUTTON(BUTTON1, MyFrame::OnButton1)
5 END_EVENT_TABLE()
```

Fragment kódu 3.2: Definice tabulky událostí ve wxWidgets.

3.3 Qt

Vývoj knihovny Qt sahá také do roku 1992, kdy byl započat vývoj s cílem umožnit co nejefektivnější a nejrychlejší tvorbu aplikací. Verze 4 je značně přepracována a obohacena o nové funkce. Oproti starší verzi obsahuje plnou podporu znakové sady Unicode, vlastní skriptovací jazyk, implementaci nástrojů na zpracování XML a podporu pro vektorové grafické formáty. Knihovna je multiplatformní - podporuje více než 10 platforem a na platformě *MeeGo* a Linux s desktopovým prostředím *KDE* je nativní knihovnou

grafického rozhraní. *Qt4* obsahuje více než 500 tříd a 9000 funkcí a je proto rozděleno do několika nezávislých modulů [14].

Základem knihovny je paradigma signál/slot, které umožňuje komunikaci mezi objekty. Každý objekt může definovat signály, které jsou vyslány vykonáním příkazu `emit nazevSignalu()`, když se změní stav objektu. Deklarací signálu je metoda s návratovým typem `void`, která však nesmí být nikde definována. Definice signálů jsou automaticky generovány nástrojem *moc* (*Meta - Object Compiler*). Na tyto signály lze reagovat pomocí slotů, což jsou funkce, které mohou být volány běžným způsobem, ale také mohou být napojeny na signály funkcí `connect()`. Na jeden signál může být připojeno více slotů a ty jsou pak vykonány po sobě.

Funkcionalita signál/slot je implementována jako rozšíření jazyka C++, zdrojový kód, ve kterém je deklarována třída obsahující makro `Q_OBJECT`, nebo klíčová slova `signal`, `slot`, `emit` a další klíčová slova definovaná v *Qt*, je analyzován nástrojem *moc*. Tento nástroj vytvoří soubor s předponou `moc_`, ve kterém jsou tyto nestandardní prvky nahrazeny běžným kódem C++, a tento soubor je pak přeložen a přilinkován k programu.

Grafické uživatelské rozhraní programů postavených na knihovně *Qt* lze definovat obdobně jako s použitím wxWidgets přímo v kódu aplikace běžnými konstrukcemi jazyka C++. *Qt4* ale podporuje také definici GUI pomocí XML souboru. Tento přístup umožňuje oddělit definici uživatelského rozhraní od kódu programu, který je pak přehlednější. Vzhled aplikace může být definován bez znalosti jazyka C++ a formát XML umožňuje jednodušší strojové zpracování. Soubor s definicí rozhraní lze poté použitím nástroje *uic* (*User Interface Compiler*) převést na běžný C++ hlavičkový soubor, který se přeloží spolu s aplikací. Soubor lze ale také zpracovat až při samotném běhu programu a dynamicky generovat uživatelské rozhraní z XML souboru, jednoduchá úprava vzhledu tedy nebude vyžadovat překlad programu.

3.3.1 Základní prvky aplikace s použitím Qt

Aplikace napsané pomocí knihovny *Qt* jsou založené na objektu `QApplication` a na potomcích objektu `QWidget`. Třída `QApplication` obsahuje hlavní smyčku událostí, zajišťuje inicializaci, správu sezení a nastavení aplikace. Objekt třídy `QWidget` je základ uživatelského rozhraní, získává události od systému a vykresluje svůj stav na obrazovku. Instance objektu `QWidget` mohou obsahovat další prvky `QWidget`, prvek, který není vložen do žádného rodičov-

ského prvku se stává nezávislým oknem, takto je ale vhodné použít jen třídy `QDialog` nebo `QMainWindow`. Každý prvek uživatelského rozhraní má předdefinované signály a sloty na běžné akce, například prvek typu `QPushButton` (tlačítko) vysílá signály při klepnutí, stisknutí tlačítka a uvolnění tlačítka.

Třída reprezentující hlavní okno aplikace a zpracovávající události je vytvořena děděním od třídy `QMainWindow` nebo `QDialog`. V této třídě je definován objekt uživatelského rozhraní, signály, které bude hlavní okno vysílat a sloty, které budou reagovat na události. Třída také obsahuje objekt uživatelského rozhraní, pomocí kterého je možné k jednotlivým prvkům přistupovat a měnit jejich vzhled, vlastnosti a stav. Příklad deklarace okna aplikace, založeného na třídě `QMainWindow` je uveden ve fragmentu kódu 3.3 a implementace ve fragmentu kódu 3.4.

```
1  /* Soubor myclass.ui je preveden nastrojem "uic" na
       hlavickovy soubor */
2  #include "ui_myclass.h"
3  #include <QMainWindow>
4
5  namespace Ui {
6  class MyClass;
7  }
8
9  class MyClass : public QMainWindow
10 {
11   /* Makro Q_OBJECT je rozvinuto nastrojem "moc". */
12    Q_OBJECT
13
14 public:
15    MyClass(QObject *parent = 0);
16    ~MyClass();
17
18 signals:
19   /* signal je pouze deklarovan, implementaci vygeneruje
        nastroj "moc" */
20    void mySignal();
21
22 public slots:
23   // slot je definovan jako klasicka funkce v souboru .cpp
24    void mySlot();
25
26 private:
27    Ui::MyClass *ui; // objekt uzivatelskeho rozhrani
28 };
```

Fragment kódu 3.3: Deklarace třídy vycházející z `QMainWindow`.

V implementačním souboru je v konstruktoru třídy inicializováno grafické uživatelské rozhraní a implementovány sloty. Funkcí `connect()` lze signály propojit se sloty jiných prvků rozhraní nebo sloty definovanými v aplikaci.

```
#include "ui_test.h"
#include <QMainWindow>

MyClass::MyClass(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MyClass)
{
 /* Graficke rozhrani je inicializovano a konfigurovano
     zavolanim funkce setupUi() */
    ui->setupUi(this);
 /* K jednotlivym prvkum je mozne pristoupit skrz objekt
     rozhrani */
    ui->label->setText("Label text");

 /* Signaly a sloty jsou propojeny volanim fce connect() */
    connect(this, SIGNAL(mySignal()),this, SLOT(mySlot()));
}

MyClass::~MyClass()
{
    delete ui;
}

/* Implementace slotu. */
void MyClass::mySlot()
{
    qApp->quit();
}
```

Fragment kódu 3.4: Implementace okna aplikace a definice slotů.

Struktura souboru uživatelského rozhraní ve formátu XML je znázorněna ve fragmentu kódu 3.5, význam povinných elementů první úrovně je následující:

`class` - Definuje název třídy, se kterou je rozhraní spojeno.

`connections` - Definuje spojení signálů a slotů jednotlivých prvků rozhraní. Například spojení posuvníku a textového pole, změnou hodnoty jednoho prvku se provede změna hodnoty i v propojeném prvku. Propojením signálů a slotů prvků GUI v souboru .ui je více oddělena programová logika aplikace od definice uživatelského rozhraní.

resources - Obsahuje cesty k binárním souborům použitým v rozhraní. Do této kategorie spadají například ikony a obrázky použité jako ovládací prvky nebo dekorace rozhraní.

widget - Kořenový prvek uživatelského rozhraní. Obsahuje další elementy rozhraní typu `widget` a `spacer`, nebo prvky `layout`, umožňující prostorovou organizaci svých potomků.

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>Dialog</class>
 <!-- korenovy prvek rozhrani -->
 <widget class="QDialog" name="Dialog">
  <!-- vlastnosti prvku -->
  <property name="windowTitle">
   <string>Dialog</string>
  </property>
  <!-- rozvrzeni prvku v~okne -->
  <layout class="QVBoxLayout" name="verticalLayout">
   <item>
    <!-- prvek rozhrani - popisek -->
    <widget class="QLabel" name="popisek">
     <property name="text">
      <string>Popisek</string>
     </property>
    </widget>
   </item>
   <item>
    <!-- prvek rozhrani - tlacitko -->
    <widget class="QPushButton" name="zavrit">
     <property name="text">
      <string>Zavrit</string>
     </property>
    </widget>
   </item>
  </layout>
 </widget>
 <resources/>
 <connections/>
</ui>
```

Fragment kódu 3.5: Struktura souboru .ui jednoduchého uživatelského rozhraní.

Knihovny *Qt4* a *wxWidgets* mají z hlediska prvků grafického uživatelského rozhraní velmi podobné možnosti, ale jak je z uvedených příkladů vi-

dět, přístup obou knihoven je značně odlišný a nekompatibilní, jejich záměna se neobejde bez výrazných zásahů do kódu aplikace.

V průběhu vývoje aplikace se mění požadavky, cílová skupina uživatelů či prostředí, ve kterém bude aplikace provozována, a na tyto změny je potřeba reagovat například právě nahrazením použité knihovny za jinou, která lépe pokryje nové potřeby aplikace. Podobné změny se nevyhnuly ani platformě MAF3, pro platformu MAF2 ale již bylo vytvořeno množství tříd, které jsou s novou verzí platformy nekompatibilní. Tyto třídy by bylo vhodné upravit pro použití s novou platformou a kvůli jejich velkému počtu a shodným úpravám jednotlivých tříd by měl být převod automatizován.

4 Automatická úprava kódu

Údržba a oprava softwaru je v současné době nedílná část jeho životního cyklu. Aplikace se musí přizpůsobovat požadavkům uživatelů, změnám softwarového a hardwarového prostředí, ve kterém je používána, ale aplikace musí být často také přepracována z důvodu rozšiřitelnosti a udržitelnosti dalšího vývoje. Značná část vynaloženého úsilí a prostředků určených na vývoj aplikace je spotřebována na aktualizaci či výměnu komponent aplikace a přizpůsobení ostatních částí k novému rozhraní, tedy činnost, která přímo nevede k rozšíření funkcionality. Zásah do funkčního a již prověřeného zdrojového kódu také zvyšuje šanci zanesení nových chyb a tím dále zvyšuje náklady.

Při přechodu na novou knihovnu či novou verzi stávající je často nutné provést množství shodných úprav. Současné rozsáhlé aplikace přinášejí nutnost automatizovat tyto úpravy. Z velké části lze jednotlivé úkony formalizovat a umožnit tak jejich automatizaci, čímž se předejde mnoha výše uvedeným problémům. Do kódu nebudou zaneseny chyby z nepozornosti, programátoři budou mít více času na skutečně produktivní práci a software bude snáze převeden na modernější knihovny a technologie. Nástroje na automatickou úpravu ale mají omezenou možnost ověřit správnost upraveného kódu, jednotlivé kroky úpravy by měly být důkladně ověřeny pro případ, že by nástroj například změnil i kód mimo naplánovaný rozsah či provedl jinou neočekávanou operaci.

Automatická úprava kódu byla ve velkém rozsahu úspěšně použita například v projektu *Mozilla* (viz [7]). Zdrojový kód vykreslovacího jádra webového prohlížeče *Mozilla Firefox* je založen na zdrojových kódech aplikace *Netscape Navigator*. Tento kód obsahoval množství zastaralých jazykových konstrukcí a nevyužíval výjimky jazyka C++. Automatickou úpravou sadou nástrojů *Pork* byl zdrojový kód modernizován a bylo přepsáno programové rozhraní aplikace. Sada nástrojů *Pork* je založena na syntaktickém a sémantickém analyzátoru jazyka C/C++ *elsa* a překladači *GNU Compiler Collection*. Podrobný popis jednotlivých nástrojů je uveden na webové stránce projektu *Mozilla* (viz [8]).

Také společnost *Google* využívá automatické úpravy zdrojového kódu v mnoha svých projektech. Nástroje vyvinuté společností *Google* jsou založené na knihovně *libTooling*, která je vyvíjena jako součást překladače *Clang*.

Možnosti a výsledky knihovny a nástrojů na ní postavených byly prezentovány v přednášce *Refactoring C++ with Clang* (viz [1]). V přednášce *Clang MapReduce – Automatic C++ Refactoring at Google Scale* (viz [2]) byl prezentován nástroj postavený na technologii *MapReduce* (viz [4]) umožňující paralelní zpracování přes 100 milionů řádků kódu v řádu minut s využitím datového centra společnosti *Google*.

Nástroj realizovaný v rámci této práce bude zpracovávat moduly aplikací implementovaných na platformě MAF2, jejich zdrojový kód bude analyzován, definice uživatelského rozhraní modulu bude převedena do XML dokumentu ve formátu *Qt Designer*. Ve zdrojovém kódu modulu budou provedeny úpravy zajišťující kompatibilitu s platformou MAF3 a budou doplněny metody obsluhující událostí uživatelského rozhraní. Nástroj bude zpracovávat pouze funkcionalitu uživatelského rozhraní, pro použití modulu v aplikaci platformy MAF3 bude nutné manuálně upravit ostatní metody a části zdrojového kódu. V některých případech nebude možné modul na platformu MAF3 převést, nebo bude nutné změnit chování modulu. Je to dáno tím, že během porovnání obou verzí platformy MAF byly nalezeny značné rozdíly v dostupné funkcionalitě, platforma MAF3 v době realizace práce neposkytuje ekvivalentní implementaci části funkcí, zejména v oblasti GUI.

4.1 Analýza zdrojového kódu

Základní funkcí konverzního nástroje bude syntaktická analýza jazyka C++, ve kterém jsou moduly platformy MAF2 implementované. Nástroj musí umožňovat analýzu zdrojového kódu na úrovni odpovídající požadované transformaci, pro jednoduché úpravy postačí rozpoznání jednotlivých symbolů, komplexnější transformace mohou požadovat sémantickou analýzu a úpravu abstraktního syntaktického stromu.

K vytvoření syntaktického analyzátoru je možné využít generátory *GNU Bison* nebo *ANTLR*. Tyto nástroje generují ze souboru s popisem bezkontextové gramatiky syntaktický analyzátor v určeném programovacím jazyce. V souboru s popisem gramatiky lze k jednotlivým prvkům jazyka specifikovat kód, který vykoná syntaktický analyzátor při identifikaci daného prvku. Na základě vygenerovaného syntaktického analyzátoru je možné implementovat sémantický analyzátor a dále zpracovávat, či transformovat zdrojový kód. Gramatika jazyka C++ je velmi komplexní a nemusí být ve všech případech bezkontextová, pro korektní analýzu je také nutné zdrojový kód předzpraco-

vat preprocesorem a vyhodnotit direktivy určené pro preprocesor překladače.

Další možností je založit nástroj na již existujícím a ověřeném analyzátoru. Podle porovnání, provedeného komunitou vývojářů projektu *Clang* (viz [3]), mají z dostupných analyzátorů s otevřeným zdrojovým kódem nejlepší podporu jazyka C++ analyzátory *GCC* a *Clang*. Syntaktický analyzátor překladače *GCC* je silně provázán s ostatními částmi kompilátoru a není možné ho použít samostatně. Nevýhodou je také implicitní zjednodušování zdrojového kódu, které odstraňuje pro transformační nástroj důležité informace. Vhodnější základ transformačního nástroje je proto překladač *Clang*, který byl pro integraci do editačních a transformačních nástrojů navržen.

Clang je sada knihoven a kompilátor programovacích jazyků C, C++ a Objective-C založený na knihovnách *LLVM*. Soubor knihoven *LLVM* (viz [6]) poskytuje funkce pro implementaci virtuálních strojů, optimalizaci a generování strojového kódu pro statickou a dynamickou kompilaci. Oba projekty jsou vyvíjeny primárně pro operační systémy Unix a Linux. Podpora platformy *MS Windows* je v experimentálním stádiu, funkce pro analýzu a úpravu zdrojového kódu nejsou na této platformě použitelné.

Překladač *Clang* poskytuje infrastrukturu pro vývoj nástrojů využívajících syntaktickou a sémantickou analýzu a manipulaci s abstraktním syntaktickým stromem. Nástroje mohou být implementovány ve formě zásuvného modulu překladače nebo samostatného nástroje založeného na knihovně *libTooling*. Knihovna *libTooling* vyžaduje pro svoji funkci databázi kompilačních příkazů, ze které zjišťuje s jakými parametry byly jednotlivé soubory zpracovány. Databázi kompilačních příkazů je možné vygenerovat nástrojem pro automatizaci překladu *CMake* od verze 2.8.5. Popis možností knihovny *libTooling* a formátu kompilační databáze je dostupný na webových stránkách dokumentace projektu *Clang* (viz [15]).

4.1.1 Zvolený způsob syntaktické analýzy

Platforma MAF2 je založena na multiplatformních knihovnách, ale hlavní podporovaná platforma je pouze *MS Windows*. Dle webových stránek projektu MAF (viz [10]) byla platforma MAF2 úspěšně zkompilována také na platformě Ubuntu Linux 8.04, poté již nebyla podpora platformy Linux ověřována. Možnost kompilace na novější verzi operačního systému Ubuntu Linux byla ověřena i v rámci této práce, konkrétně na verzích 10.04 a 12.10, v obou případech se nepodařilo platformu MAF2 přeložit. Pro Ubuntu Linux ve verzi

8.04 však nejsou nástroje *Clang*, ani *Cmake* s podporou generování kompilační databáze dostupné, z tohoto důvodu není možné syntaktický analyzátor a další funkce nástroje *Clang* využít.

Na základě výše uvedených zjištění bylo rozhodnuto, že pro transformační nástroj bude implementován vlastní syntaktický analyzátor s minimální závislostí na použité platformě. Transformace modulů bude možné provádět na platformách *Mac OS*, *MS Windows* i *GNU/Linux*, jenž jsou projektem MAF3 podporovány. Transformace bude zpracovávat specifickou část modulů platformy MAF2 s předvídatelnou strukturou zdrojového kódu, syntaktický analyzátor proto bude zaměřen jen na tuto oblast.

Analyzátor bude zpracovávat odděleně hlavičkové a implementační soubory, v implementačním souboru .cpp nejprve nalezne definice metod a rozsah jejich řádek, z nalezených definic jsou dle názvu vybrány metody, které budou dále zpracovány. Analyzátor předpokládá standardní formu modulu platformy MAF2, kdy názvy cílových metod určených k transformaci jsou ve všech modulech shodné. Tento přístup neumožňuje správně zpracovat moduly, jejichž uživatelské rozhraní, nebo jeho část je definována v jiné než očekávané metodě `CreateGui`, respektive `CreateOpDialog`. Z vybraných metod jsou nejprve odstraněny komentáře a direktivy podmíněného překladu preprocesoru a dále jsou zpracovány specializovaným analyzátorem pro danou metodu, který identifikuje příkazy určené k transformaci.

Analýza hlavičkového souboru bude probíhat obdobným způsobem, nejprve bude určen rozsah řádek deklarace třídy a v určeném rozsahu budou zpracovány jednotlivé deklarace metod a proměnných v závislosti na jejich modifikátoru přístupu.

5 Návrh nástroje

Navrhovaný nástroj bude pracovat na úrovni jednotlivých souborů *.cpp*. Ve vstupním souboru identifikuje metody obsahující definice uživatelského rozhraní, pokud je soubor obsahuje. Rozhraní může být tvořeno jednotlivými prvky *wxWidgets*, voláním metod třídy *mafGUI*, které do okna přidávají bloky předformátovaných prvků, nebo kombinací obou uvedených možností.

Nástroj rozpozná definici objektů grafického rozhraní i volání metod třídy `mafGUI` a z nalezených prvků vytvoří stromovou strukturu, reprezentující okno nebo panel uživatelského rozhraní. Každý uzel stromu bude obsahovat typ prvku, jeho vlastnosti a formát. Z tohoto stromu pak nástroj sestaví dokument ve formátu XML s příponou .ui, který slouží k definici GUI s použitím knihovny *Qt4*. Dokument XML bude doplněn o elementy, které jsou povinné dle specifikace XML schéma .ui souboru [13]. Rozpoznané prvky rozhraní budou převedeny na odpovídající prvky z platformy *Qt4* a prvky, které nemají odpovídající element v *Qt4*, budou nahrazeny textovým prvkem s výrazným formátováním, který bude obsahovat název chybějícího prvku. Uživatel může následně v nástroji Qt Designer snadno identifikovat, na jaké pozici je chybějící prvek, a nahradit ho vhodnou alternativou.

Následně nástroj převede metodu implementující funkce spojené s aktivací prvku uživatelského rozhraní. Tato metoda bude rozdělena na *Qt4* sloty pojmenované podle jména jejich přidruženého prvku. Dále budou vygenerovány sloty zajišťující uložení hodnoty z vstupního prvku do přidružené proměnné. Deklarace vygenerovaných slotů bude vložena do hlavičkového souboru a deklarace třídy bude rozšířena o makra `Q_OBJECT` a `Q_PROPERTY` knihovny Qt4.

Komentáře, direktivy preprocesoru a části zdrojového kódu, které program neidentifikuje, budou zapsány do souboru logu, který se vytvoří v průběhu zpracování. Zápis v logu bude označen typem zaznamenaného případu a jeho pozicí v souboru.

5.1 Zpracování implementačního souboru

Prvním krokem analýzy je určení názvu třídy modulu, nástroj předpokládá dodržení konvence pojmenování souboru shodně s názvem implementované

třídy. Dodržení této konvence je ověřeno jednoduchou kontrolou obsahu souboru, část názvu souboru před příponou .cpp je považována za název třídy a obsah souboru je procházen ve smyčce po jednotlivých řádkách, dokud není nalezena definice metody této třídy. V případě, že metoda této třídy není nalezena, nástroj požádá o zadání správného názvu.

Nástroj poté zaznamená čísla řádek direktiv `#include` a definic jednotlivých metod třídy. Hlavička metody je identifikována pomocí regulárního výrazu, který obsahuje název třídy, metody jiných tříd ve zpracovávaném souboru jsou proto ignorovány. Názvy metod jsou podle slovníku upraveny na název odpovídající metody použitý v MAF3. Po nalezení hlavičky metody jsou vyhledávány složené závorky vymezující blok definice metody. Rozsah bloku je určen pomocí zásobníku, jehož hodnotu zvyšují nalezené levé závorky a pravé naopak snižují. Řádka s první nalezenou levou složenou závorkou je označena jako začátek bloku, konec bloku je určen po dosažení nulové hodnoty zásobníku. Rozsah řádek je uložen do slovníku spolu s hodnotou typu boolean, která určuje zda má být metoda zahrnuta v transformovaném souboru, název metody je použit jako klíč. Ze slovníku jsou poté vybrány a zpracovány metody `CreateGui`, `CreateOpDialog`, `OnEvent` a konstruktor třídy.

5.1.1 Zpracování metod GUI

Definice grafického uživatelského rozhraní je implementována v metodách `CreateGui` a `CreateOpDialog`, obě metody jsou zpracovány shodně. Nástroj nejprve v metodě nalezne příkazy vytvoření nového objektu typu `mafGUI` a `mafGUIDialog`. Pro každý objekt je vytvořen prázdný záznam ve slovníku objektů rozhraní, každý záznam je určen názvem objektu a jeho hodnota je strom prvků uživatelského rozhraní. Pokud metoda neobsahuje žádný objekt typu `mafGUI` ani `mafGUIDialog`, nelze v transformaci pokračovat, je vypsána chybová hláška a nástroj je ukončen.

Strom prvků uživatelského rozhraní implementovaný v nástroji odpovídá strukturou zjednodušenému stromu elementů XML schéma formátu *Qt Designer* (viz [13]). Na rozdíl od tohoto formátu, použitém v .ui souborech pro definici rozhraní aplikací *Qt4*, jsou ve stromu prvků pouze uzly definující uživatelské rozhraní a jeho vzhled. Uzel stromu může patřit do jedné ze tří možných tříd:

widget - Základní prvek grafického rozhraní, může být ve formě vlastního prvku, nebo libovolného prvku rozhraní podporovaného knihovnou *Qt4*.

layout - Prvek, který určuje rozvržení prvků v něm obsažených. Prvky mohou být rozvrženy horizontálně, vertikálně nebo ve formě tabulky do sloupců a řádků.

spacer - Tento prvek umožňuje manipulovat s prostorem mezi dvěma prvky a jeho chováním při změně velikosti okna. *Spacer* se může rozpínat v horizontálním či vertikálním směru, nebo mít statický rozměr.

Strom prvků dále může obsahovat uzly, které mohou existovat pouze s vazbou na jeden z výše uvedených prvků. Tyto uzly definují vlastnosti a data jednotlivých prvků uživatelského rozhraní. Možné vazby mezi jednotlivými typy uzlů jsou znázorněny na obrázku 5.1.

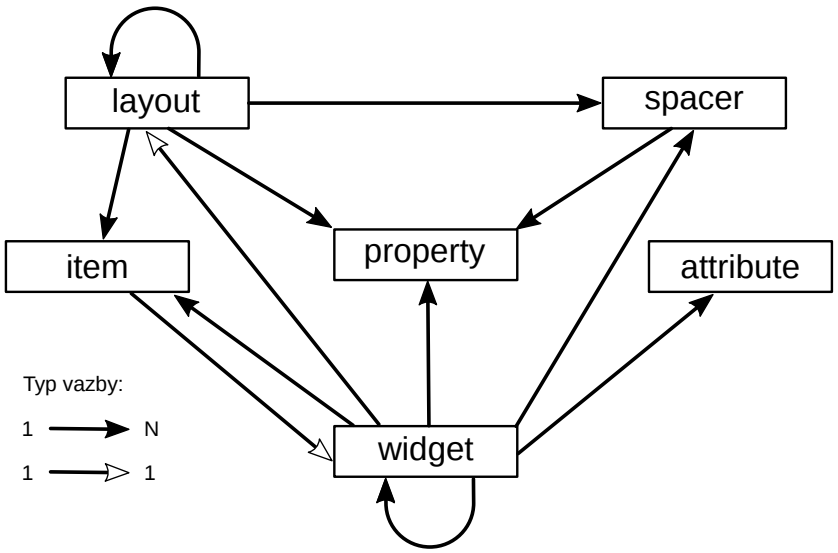
property - Definuje vlastnosti prvků rozhraní z hlediska vzhledu a chování. Tento uzel nemusí být povinně navázán na prvek uživatelského rozhraní, ale ve většině případů definuje základní vlastnosti, bez kterých prvek nemá smysl. Uzel *property* definuje například orientaci prvků *layout* a *spacer*, nebo text tlačítka prvku *widget*.

item - Pro prvky rozhraní, které zobrazují data ve formě seznamu, představuje uzel *item* data jedné položky seznamu.

attribute - Definuje vlastnost prvků rozhraní relevantní k jejich použití v určitém typu prvku. *Attribute* může například definovat titulek jednotlivých kořenových prvků vložených v prvku s přepínáním panelů.

Analýza příkazů

Uživatelské rozhraní platformy MAF2 je definováno třemi typy příkazů: volání metody objektu `mafGUI`, vytvoření nového objektu prvku *wxWidgets* a volání metody objektu *wxWidgets*. Pro zjednodušení analýzy příkazů je nejprve metoda předzpracována, čímž jsou nalezené komentáře a direktivy preprocesoru zapsány do logu a odstraněny z kódu metody. Nástroj nezpracovává příkazy `if`, `else`, `for` a další příkazy řídící běh programu, při zpracování definic uživatelského rozhraní jsou tyto příkazy a jejich pozice zaznamenány do logu a dále nejsou brány v potaz.



Obrázek 5.1: Schéma možných vazeb mezi typy uzlů stromu

Nástroj obsahuje slovník, ve kterém nalezne pro každý objekt či metodu funkci určenou k jejímu zpracování. Metody a konstruktory objektů mají definované výchozí hodnoty parametrů v hlavičkovém souboru, při jejich volání proto nemusí být všechny argumenty uvedeny. Funkce odpovídající určité metodě či objektu proto obsahuje seznam výchozích hodnot argumentů, které se využijí podle zjištěného počtu argumentů uvedených v analyzovaném příkazu. Jednotlivé typy příkazů jsou nalezeny pomocí regulárních výrazů a poté je každý typ dále zpracován vlastním analyzátorem následujícím způsobem:

Volání metody objektu mafGUI - Uživatelské rozhraní v tomto případě tvoří objekt třídy *mafGUI*, do kterého mohou být případně včleněny další objekty stejného typu. Prvky uživatelského rozhraní jsou do tohoto objektu přidávány voláním metod. Příkaz je rozdělen na název objektu, název metody a seznam jejích argumentů. Název objektu určuje, do kterého stromu prvků uloženém ve slovníku objektů rozhraní bude aktuální prvek umístěn, název metody odpovídá typu prvku, bloku prvků, nebo operaci prováděné nad stromem prvků. Argumenty metody obsahují identifikátor prvku a další data potřebná k inicializaci. Příklad volání metody objektu *mafGUI* je uveden ve fragmentu kódu 5.1 a výstup transformačního nástroje ve fragmentu kódu 5.2.

```
m_Gui->Double(ID_DOUBLE_INPUT, "Popisek", &promenna);
```

Fragment kódu 5.1: Definice vstupního textového pole přijímající pouze číselné hodnoty v rozsahu typu double.

```
<layout class="QHBoxLayout" name="horizontalLayout">
 <item>
  <widget class="QLabel" name="double_input_label">
   <property name="text">
    <string>Popisek</string>
   </property>
   <property name="buddy">
    <cstring>double_input</cstring>
   </property>
  </widget>
 </item>
 <item>
  <widget class="QDoubleSpinBox" name="double_input" />
 </item>
</layout>
```

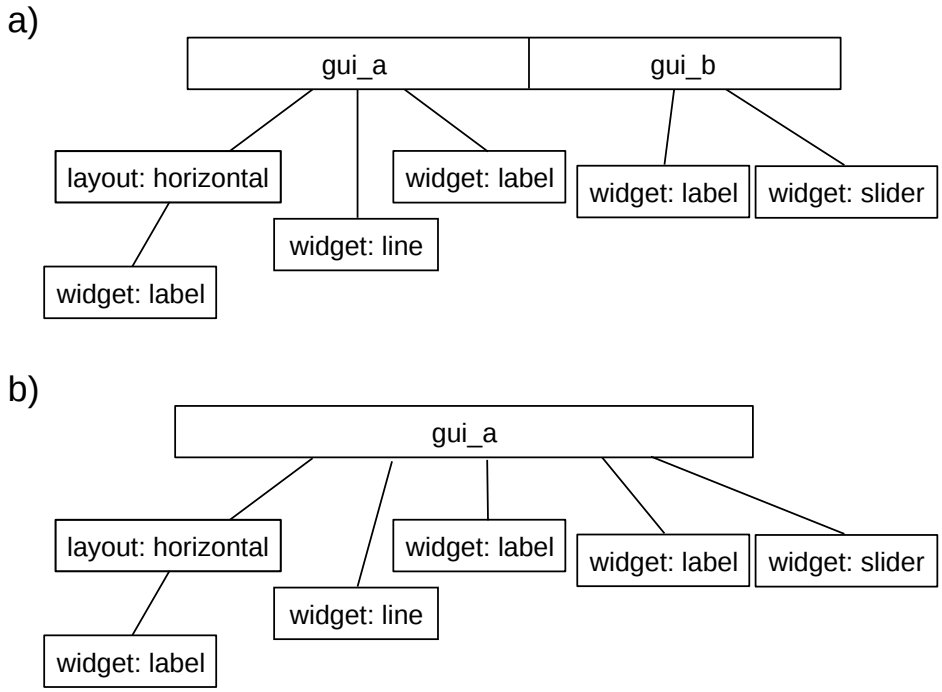
Fragment kódu 5.2: Vstupní pole typu double převedené do XML formátu *Qt Designer*.

Nástroj poté vytvoří uzly prvků uživatelského rozhraní typu *widget* a tyto prvky uspořádá do uzlů typu *layout*. Dle hodnoty argumentů metody nastaví jejich jména a podle potřeby připojí uzly typu *property*. Pokud je alespoň jeden z prvků rozhraní aktivní a umožňuje vstup dat, je vygenerována definice slotu, který uloží zadanou hodnotu do proměnné uvedené v argumentu metody. V případě, že metoda definuje více aktivních prvků pro nastavení hodnoty jedné proměnné, jsou prvky synchronizovány propojením vhodných signálů a slotů.

Podpora prvku rozhraní nemusí v platformě MAF3 existovat, v tomto případě je na místo tohoto prvku vložen prvek typu textový popisek s výrazným červeným pozadím a do log souboru je zapsána zpráva. Takto naformátovaný prvek je při úpravě souboru rozhraní v nástroji *Qt Designer* ihned patrný a může být nahrazen vhodnou alternativou dle výběru uživatele.

Odlišným způsobem je zpracována metoda `AddGui()`. Tato metoda umožňuje do objektu třídy *mafGUI* vložit uživatelské rozhraní jiného objektu této třídy. Strom prvků rozhraní objektu uvedeného jako argument metody je vložen do objektu, nad kterým je metoda volána, a odkaz na vložený strom je odstraněn ze slovníku objektů rozhraní. Výsledné uspořádání stromu uži-

vatelského rozhraní je znázorněno na obrázku 5.2. Ostatní metody objektu *mafGUI* nedefinují uživatelské rozhraní a nejsou zpracovávány, jejich funkcionalita v kontextu platformy MAF3 nemá smysl.



Obrázek 5.2: Příklad struktury stromů prvků rozhraní a) před a b) po provedení příkazu `gui_a->AddGui(gui_b);`

Vytvoření nového objektu *wxWidgets* - V tomto případě je uživatelské rozhraní tvořeno objektem *mafGUIDialog*. Prvky uživatelského rozhraní jsou definovány objekty knihovny *wxWidgets*. Příkaz vytvoření nového objektu je rozdělen na název objektu, třídu objektu a argumenty konstruktoru, prvek je vždy uložen do stromu prvků rozhraní pojmenovaném `wx`. Fragment kódu 5.3 představuje běžný příkaz vytvoření nového objektu prvku rozhraní *wxWidgets* a následující fragment kódu 5.4 stejný prvek převedený do XML formátu *Qt Designer*.

```
1 wxButton* bttn = new wxButton( m_Dialog, ID_BUTTON, wxT("OK")
    , wxDefaultPosition, wxDefaultSize, 0 );
```

Fragment kódu 5.3: Příkaz vytvoření nového objektu třídy *wxButton*.

```
<widget class="QPushButton" name="button">
 <property name="text">
  <string>OK</string>
 </property>
</widget>
```

Fragment kódu 5.4: Prvek *wxButton* převedený nástrojem na XML definici.

Vytvořený uzel ve stromu prvků je označen názvem objektu a pro aktivní prvky také identifikátorem uvedeným v argumentu konstruktoru. Příkaz vytvoření nového objektu neobsahuje informaci o jeho pozici ve stromu prvků, nové uzly jsou proto umístěny na první úroveň stromu. Průběh zpracování objektu je dále shodný se zpracováním metod objektu *mafGUI*.

Volání metody objektu *wxWidgets* - Metodami objektů je možné upravovat vlastnosti, měnit pozici ve stromu prvků či definovat sloty prvků rozhraní *wxWidgets*. Příkaz je rozdělen na název objektu, název metody a argumenty metody. Prvek s názvem odpovídajícím názvu objektu je vyhledán procházením do hloubky všech stromů prvků rozhraní.

Pro metody upravující vlastnosti objektu je nalezený prvek rozšířen o uzel typu *property*, který odpovídá názvu metody objektu a uvedeným argumentům. Doplnění prvku rozhraní o výsuvný popisek je ilustrováno na fragmentu kódu 5.5 a výstup nástroje pro tento příkaz je uveden ve fragmentu kódu 5.6.

```
bttn->SetToolTip( wxT("Text popisku tlacitka.") );
```

Fragment kódu 5.5: Příkaz přidávající výsuvný popisek prvku k tlačítku `bttn`.

```
<widget class="QPushButton" name="button">
 <property name="toolTip">
  <string>Text popisku tlacitka.</string>
 </property>
 <property name="text">
  <string>OK</string>
 </property>
</widget>
```

Fragment kódu 5.6: Příkaz přidávající výsuvný popisek prvku k tlačítku `bttn`.

Metoda `SetValidator()` připojí k prvku objekt třídy `mafGUIValidator`, který zajišťuje uložení hodnoty vstupu prvku do proměnné. Nástroj z argu-

mentů této metody získá název proměnné a vytvoří definici slotu implementujícího ukládání hodnoty prvku do proměnné.

Výchozí umístění nových prvků je na první úrovni stromu prvků rozhraní, metodou `Add()` je možné prvek přesunout na pozici podřízeného uzlu libovolného prvku typu *layout*. Takto je možné přesouvat jednotlivé prvky i celé větve stromu. Prvek uvedený jako argument metody bude umístěn jako podřízený uzel prvku, jehož metoda je volána.

Jako podřízený prvek nemusí být vložen pouze již existující objekt, argument metody může obsahovat také příkaz vytvoření nového anonymního objektu. Příklad příkazu vytvoření anonymního objektu je uveden ve fragmentu kódu 5.7. Pro umístění nového prvku do stromu je nutná jeho jednoznačná identifikace, prvku je přiděleno jméno složené z názvu nadřazeného uzlu a náhodně vygenerovaného čtyřmístného alfanumerického řetězce. Příkaz vytvoření nového objektu je poté zpracován shodně s ostatními objekty *wxWidgets*.

```
1 bSizer->Add( new wxStaticText( m_Dialog, wxID_ANY, wxT("Text
    popisku"), wxDefaultPosition, wxSize( 100,-1 ),
    wxALIGN_RIGHT ), 1, wxALL|wxALIGN|CENTER_VERTICAL, 1 );
```

Fragment kódu 5.7: Příkaz `Add()` s definicí nového anonymního prvku typu `wxStaticText`.

Uložení stromu prvků do formátu XML

Po zpracování všech příkazů metody `CreateGui()`, respektive `CreateOpDialog()` je strom prvků rozhraní dokončen a poté je vygenerován XML dokument ve formátu *Qt Designer* s příponou *.ui*. Nástroj vytvoří základní strukturu dokumentu, která je doplněna stromem prvků rozhraní převedeným do XML formy. Strom prvků je převeden průchodem stromu do hloubky a vytvořením elementu pro každý nalezený uzel. Uzly stromu a jejich obsah odpovídají elementům formátu *Qt Designer*, pro každý uzel je tedy vytvořen XML element shodného typu a obsah uzlu je zapsán bez dalších úprav jako obsah XML elementu. Strom XML elementů je následně zapsán do souboru pojmenovaném dle názvu třídy.

Na přiloženém DVD jsou dostupné vygenerované soubory uživatelského rozhraní modulů platformy MAF2 (viz příloha A).

5.1.2 Transformace událostí rozhraní

Při zpracování prvků uživatelského rozhraní byly zaznamenány jejich identifikátory, podle kterých je určen původce události uživatelského rozhraní a vyvolána odpovídající akce. Události nejen z uživatelského rozhraní jsou v modulech zpracovávány metodou `OnEvent()`, která obsahuje příkaz `switch` s bloky `case` pro jednotlivé prvky rozhraní.

Bloky `case` jsou vykonávány v závislosti na identifikátoru zpracovávané události. Pokud identifikátor bloku odpovídá některému identifikátoru prvku rozhraní, je blok transformačním nástrojem převeden na slot. Název slotu je vytvořen z identifikátoru prvku odstraněním předpony `ID_`, převedením na malá písmena a doplněním o předponu `on_` a příponu definovanou při analýze prvků grafického rozhraní. Slot odpovídající identifikátoru `ID_FIRST_BUTTON` bude nazván `on_first_button_clicked`. Takto nazvané sloty jsou automaticky spojeny s odpovídajícím signálem prvku při načtení souboru uživatelského rozhraní do aplikace.

Obsah bloku `case` je vložen do nově vytvořeného slotu, pokud akce odpovídá více blokům, jsou tyto bloky spojeny do jednoho slotu s názvem vytvořeným z identifikátoru prvního bloku. Takto spojené bloky jsou poté znovu zpracovány vyjma prvního bloku, pro který byl slot již vytvořen. Varianty uspořádání bloků `case`, které se vyskytují v modulech MAF2 jsou uvedeny ve fragmentu kódu 5.8 a jak budou jednotlivé bloky převedeny znázorňuje fragmentu kódu 5.9. Příkazy v blocích `case` nejsou zpracovány, obsah bloků je bez změny vložen do vygenerovaného slotu a uživatel musí následně zkontrolovat a upravit jeho obsah.

Ostatní příkazy metody `OnEvent`, které nejsou obsaženy v žádném bloku `case`, nebo bloky, jejichž identifikátor neodpovídá žádnému prvku rozhraní, jsou ignorovány. Metoda `OnEvent` po zpracování nástrojem obsahuje kód, který nebyl rozpoznán a transformován, ale může implementovat důležitou funkcionalitu. Metoda je proto přesunuta na konec souboru a vložena do bloku komentáře, aby mohla být ověřena.

```
void operationName::OnEvent(mafEventBase *maf_event)
{
  if (mafEvent *e = mafEvent::SafeDownCast(maf_event))
    {
      switch(e->GetId())
      {
        case ID_ADD_ONE :
        {
          /* tlacitko s identifikatorem ID_ADD_ONE po stisku
             vykona prikazy v odpovidajicim bloku case. */
          value += 1;
        }
        break;

        case ID_ADD_THREE :
        {
          /* Pri vyvolani udalosti tlacitka ID_ADD_THREE jsou
              vykonany take prikazy prvku s identifikatorem
             ID_ADD_TWO. */
          value++;
        }
        case ID_ADD_TWO :
        {
          value += 2;
        }
        break;

        case ID_SET_ZERO:
        case ID_RESET_COUNT:
        {
          value = 0; // prikazy mohou byt pro vice prvku
             shodne
        }
        break;
      }
    }
}
```

Fragment kódu 5.8: Možné uspořádání bloků `case` v příkazu `switch`

```
1  void operationName::on_add_one_clicked()
2  {
3    value += 1;
4  }
5
6  void operationName::on_add_three_clicked()
7  {
8    value++;
9    value += 2;
10 }
11
12 void operationName::on_add_two_clicked()
13 {
14   value += 2;
15 }
16
17 void operationName::on_set_zero_clicked()
18 {
19   value = 0;
20 }
21
22 void operationName::on_reset_count_clicked()
23 {
24   value = 0;
25 }
```

Fragment kódu 5.9: Metoda `OnEvent` převedená nástrojem na jednotlivé sloty.

Sloty pro zpracování vstupu

Uložení hodnoty zadané pomocí prvku uživatelského rozhraní do proměnné je v MAF3 řešeno také pomocí signálů a slotů. Při zpracování uživatelského rozhraní byly pro jednotlivé prvky vytvořeny záznamy obsahující název slotu a název proměnné. Již vytvořené záznamy jsou doplněny o definici slotu `on_gui_loaded`, který je vyvolán po vytvoření uživatelského rozhraní a umožňuje modulu získat přístup k objektu uživatelského rozhraní a měnit vlastnosti jednotlivých prvků. Z těchto definic jsou vygenerovány sloty a zapsány do upraveného souboru. Všechny sloty jsou doplněny o komentář, informující uživatele, že sloty byly automaticky vygenerovány transformačním nástrojem, a měly by být ověřeny.

5.1.3 Úprava konstruktoru a vytvoření nového souboru

V platformě MAF3 byl změněn obsah a názvy mnoha hlavičkových souborů, nástroj proto musí nahradit hlavičkové soubory projektu MAF2 vložené direktivou `#include` na jejich odpovídající alternativy z MAF3. Také název třídy, od které dědí zpracovávaná třída, je nutné upravit a s tím i parametry konstruktoru třídy.

Příkazy konstruktoru jsou doplněny o příkaz specifikující název vygenerovaného souboru uživatelského rozhraní, který bude použit třídou `mafGUIManager` při inicializaci uživatelského rozhraní modulu. Dále bude vložen příkaz volání metody `setObjectName`, která nastaví jméno *QObjektu* třídy, podle kterého je možné objekt identifikovat, či k němu přistupovat skriptovacími jazyky. Nástroj obsahuje slovník, podle kterého jsou nové názvy nalezeny a nahrazeny.

Ze seznamu řádek souboru jsou poté odstraněny řádky metod `CreateGui`, `CreateOpDialog` a `OnEvent`, na konec seznamu jsou vloženy vygenerované sloty a soubor je pojmenován názvem třídy a zapsán.

5.2 Zpracování hlavičkového souboru

V souboru je nejprve určen rozsah řádek zpracovávané třídy a rozdělení třídy na části podle modifikátorů přístupu. Hlavičkový soubor třídy je zpracován s využitím dat získaných analýzou a transformací implementačního souboru. Třída je doplněna o makra `Q_OBJECT` a `Q_PROPERTY`, dále makro platformy MAF3 `mafSuperclassMacro` a deklaraci vygenerovaných slotů uvozených modifikátorem přístupu `public Q_SLOTS`.

Makro `Q_OBJECT` implementuje funkcionalitu signálů, slotů a dalších technologií specifických pro knihovnu *Qt4*. Makro `Q_PROPERTY` implementuje sofistikovaný, na překladači nezávislý systém vlastností, podporující meziobjektovou komunikaci založenou na signálech a slotech. V zápisu makra je definován typ hodnoty, název vlastnosti a metody čtení a zápisu, příklad makra je uveden ve fragmentu kódu 5.10. Podrobný popis systému vlastností je dostupný na webových stránkách projektu (viz [12]).

```
1 Q_PROPERTY(QDate date READ getDate WRITE setDate)
```

Fragment kódu 5.10: Příklad implementace makra `Q_PROPERTY`. V zápisu je definován typ hodnoty, název vlastnosti a metody čtení a zápisu.

Veřejné metody třídy jsou analyzovány a pokud je nalezena dvojice metod, které se bez předpony *get* a *set* shodují v názvu, a návratový typ jedné metody odpovídá typu parametru druhé, je vygenerováno makro `Q_PROPERTY`. Dále je třída doplněna o deklaraci proměnné typu `QWidget` pro přístup k objektu uživatelského rozhraní. Shodně s transformací implementačního souboru jsou podle slovníku upraveny názvy hlavičkových souboru v direktivách `#include` a názvy metod.

5.3 Log soubor

Během transformace jsou zaznamenávány nerozpoznané příkazy, direktivy preprocesoru, nepodporované prvky rozhraní a komentáře ze zpracovaných metod do logu. Záznam obsahuje informaci o typu záznamu, číslu řádku ve zpracovávaném souboru, na kterém se problematický příkaz vyskytl, a text záznamu.

Log je zapsán do textového souboru nazvaném shodně se zpracovávanou třídou s příponou *.log*, nebo použitím přepínače `-html` je log zapsán do souboru ve formátu *html*. Příklad záznamů v log souboru je znázorněn ve fragmentu kódu 5.11 a dále také v příloze E. Počet záznamů jednotlivých typů je po dokončení transformace vypsán do konzole a může tak sloužit jako orientační ukazatel úspěšnosti transformace.

```
1  Method CreateGui/CreateOPDialog begins at line 194:
2
3  Unknown command at line 207:
4        ((mafVME *)m_Input)->GetOutput()->GetVMEBounds(b);
5
6  Comment at line 209:
7      // bounding box dim
8
9  Preprocessor directive at line 302:
10     #ifdef _DEBUG_BONEMAT_GUI
```

Fragment kódu 5.11: Log soubor v textovém formátu.

6 Implementace, testy a výsledky

6.1 Implementace nástroje

Pro implementaci nástroje byl zvolen programovací jazyk *Python*. Tento jazyk je podporován na více než 20 platformách včetně *Mac OS*, *GNU/Linux* i *MS Windows*, v případě prvních dvou uvedených platforem je součástí výchozí instalace. V jazyce *Python* jsou vytvořeny testy kvality platforem MAF2 a MAF3 a další nástroje usnadňující instalaci či správu MAF aplikací. Z tohoto důvodu je *Python* ideální jazyk pro implementaci transformačního nástroje, nepřináší závislost na dalších technologiích, využívá pouze již zavedené prostředky pro vývoj platformy MAF.

Výhodou je také množství knihovních modulů distribuovaných s interpretem jazyka, které usnadňují práci s XML soubory, regulárními výrazy a manipulaci s textem. Výkon programů vytvořených v jazyce *Python* je velmi dobrý a výkonově kritické moduly jsou implementovány v jazyce C. Popis vlastností jazyka a jeho dokumentace je dostupná z webových stránek projektu (viz [11]). Návod na instalaci interpretu jazyka, přídavného modulu a použití nástroje je uveden v příloze B. Nástroj využívá modul *re*, který poskytuje funkce regulárních výrazů, k identifikaci komplikovaných příkazů ve zdrojovém kódu a modul *lxml* pro vytvoření XML dokumentu uživatelského rozhraní.

Zpracování 409 modulů trvá na počítači s konfigurací Intel Core i5 2500K, 8GB RAM s operačním systémem Ubuntu Linux 12.10 průměrně 17 vteřin.

6.2 Testovací data

Nástroj byl otestován transformací modulů typů *Operation*, *View* a *VME* z projektů MAF2, MAF2 Medical a aplikace LHPBuilder vytvořené v rámci projektu VPHOP WP10. Projekt MAF2 Medical je rozšíření platformy MAF2 o moduly specializované na zpracování biomedicínských dat. MAF2 Medical není samostatná aplikace, ale je určen jako základ pro vývoj medicínských aplikací. Aplikace LHPBuilder je vyvinuta na platformě MAF2 s využitím MAF2 Medical.

V tabulce 6.1 jsou uvedeny počty modulů z uvedených projektů rozděleny podle typu. Je uveden celkový počet modulů, počet modulů obsahujících metodu `CreateGui` a počet obsahujících metodu `CreateOpDialog`.

Počet modulů, které definují uživatelské rozhraní částečně, nebo zcela v jiných metodách než `CreateGui` a `CreateOpDialog` je uveden v tabulce 6.2. U těchto modulů nástroj nedokázal uživatelské rozhraní zpracovat, nebo zpracoval pouze část. V 11 případech nedefinovala metoda `CreateGui` žádné prvky uživatelského rozhraní, nebo obsahovala pouze příkaz `AddGui` pro přidání objektu rozhraní vytvořeného v jiné metodě. V tomto případě nástroj nevytvořil soubor uživatelského rozhraní přestože implementační i hlavičkové soubory byly upraveny.

	celkem	CreateGui	CreateOpDialog
MAF2	119	48	2
Operation	52	10	2
VME	67	38	0
MAF2 Medical	154	66	8
Operation	74	24	7
Views	25	21	0
VME	55	28	1
LHPApps	140	52	9
Operation	112	37	8
Views	6	2	0
VME	22	13	1

Tabulka 6.1: Celkový počet modulů a počet modulů s grafickým rozhraním v metodách `CreateGui` nebo `CreateOpDialog`.

	bez GUI	část GUI v jiné metodě	GUI v jiné metodě
MAF2	68	1	6
MAF2 Medical	78	8	5
LHPApps	74	5	1
Celkem	220	14	12

Tabulka 6.2: Počet modulů bez grafického rozhraní, s prvky rozhraní definovanými i mimo metody `CreateGui` nebo `CreateOpDialog` a s rozhraním definovaným v jiné metodě.

6.3 Výstup nástroje

Výše uvedené moduly, které obsahují grafické uživatelské rozhraní v očekávané metodě, nástroj transformoval, ostatní moduly byly ignorovány. V tabulce 6.2 je počet modulů bez uživatelského rozhraní, nebo s definicí uživatelského rozhraní v jiné metodě upřesněn. Nástroj vytvořil upravené implementační a hlavičkové soubory, soubory uživatelského rozhraní a log soubory obsahující neznámé příkazy, komentáře a direktivy preprocesoru.
Všechny vygenerované soubory uživatelského rozhraní v XML formátu *Qt Designer* jsou správně strukturovány („well-formed“) a obsahují všechny požadované elementy definované v XSD (viz [13]).

Výstupem transformace je také počet nerozpoznaných příkazů, nepodporovaných prvků a direktiv preprocesoru, které se vyskytly při zpracování modulu. Pro testovací data je v tabulce 6.3 uveden počet modulů s žádnými, s jedním až deseti a více než deseti problematickými příkazy, rozdělených podle projektu a typu modulu.

	Počet zjištěných problémů		
	0	<=10	>10
MAF2	7	38	5
Operation	5	4	3
VME	2	34	2
MAF2 Medical	9	45	20
Operation	6	16	9
Views	0	20	1
VME	3	16	10
LHPApps	23	26	12
Operation	22	15	8
Views	1	1	0
VME	0	10	4
Celkový počet	40	109	36

Tabulka 6.3: Počet modulů rozdělených do kategorií s žádnými, s jedním až deseti a více než deseti problematickými příkazy.

Z uvedených tří projektů nebyl při zpracování 22% modulů identifikován žádný problém, u 59% modulů se při zpracování vyskytlo mezi jedním a deseti problémy, více než 10 problémů bylo rozpoznáno v 19% modulů. V příloze C je uveden rozpis počtu identifikovaných problémů pro jednotlivé moduly.

6.4 Diskuze výsledků

Úspěšnost převodu nelze hodnotit pouze na základě počtu problémů rozpoznaných transformačním nástrojem. Vysoký počet neznámých příkazů může souviset s jednou izolovanou částí uživatelského rozhraní a ve vygenerovaných souborech tak bude nutné provést jen minimum úprav. Naopak jeden nebo několik málo příkazů může ovlivnit celé uživatelské rozhraní a s ním spojené funkce.

Nástroj vygeneroval pro zpracované moduly uživatelské rozhraní se správným rozvržením ovládacích prvků, které odpovídá rozhraní definovanému prostředky platformy MAF2. Ve 34 případech obsahuje uživatelské rozhraní prvek, který nemá odpovídající implementaci v MAF3. Všechny nepodporované prvky nalezené v modulu jsou spolu s nerozpoznanými příkazy zaznamenány v log souboru, podle kterého může uživatel nástroje prvky nahradit vhodnou alternativou a funkcionalitu nerozpoznaných příkazů ve vhodné formě doplnit do upraveného zdrojového kódu. Vybrané příklady výstupu nástroje jsou uvedeny v příloze D.

Nulový počet nerozpoznaných příkazů, nepodporovaných prvků a direktiv preprocesoru neznamená, že modul lze bez dalších úprav použít v aplikaci platformy MAF3. Metody modulu pracují s objekty platformy MAF2, které v naprosté většině případů nejsou shodné s objekty platformy MAF3. Při migraci modulu je kromě převodu grafického uživatelského rozhraní také nutné upravit ostatní metody modulu.

Uživatelské rozhraní některých modulů v MAF2 je vytvářeno dynamicky v závislosti na aktuálním stavu aplikace a nelze ho korektně transformovat zpracováním zdrojového kódu. Modul může obsahovat například podmínku, jejímž splněním bude uživatelské rozhraní rozšířeno o další prvky. Transformační nástroj nemá možnost tyto podmínky vyhodnotit, při zpracování takové příkazy zaznamená do logu a při vytváření stromu prvků jsou zanedbány. Uživatel poté může dle záznamů v log souboru odpovídající funkcionalitu implementovat prostředky platformy MAF3.

6.4.1 Omezení platformy MAF3

Platforma MAF3 je plánována jako nástupce MAF2, obsahující mnoho vylepšení a nových technologií, ale v době realizace této práce je stále ve vývoji

a některé funkce z platformy MAF2, jak bylo v průběhu řešení této práce zjištěno, nejsou ještě implementovány. MAF3 má oproti svému předchůdci značné nedostatky v možnostech grafického uživatelského rozhraní.

Moduly, které definují uživatelské rozhraní ve formě dialogového okna, není možné úspěšně převést, protože platforma MAF3 neobsahuje podporu pro dialogová okna modulů. Dialogová okna zpravidla obsahují prvek uživatelského rozhraní typu `mafRWI`, do kterého je vykreslován náhled konfigurace dialogového okna. Prvek `mafRWI` a další prvky specifické pro dialogová okna není možné na novou platformu převést, nástroj pro moduly vytvoří uživatelské rozhraní dialogového okna s náhradním prvkem namísto nepodporovaného prvku. Transformované moduly nelze bez podpory platformy MAF3 použít a je nutné je upravit, či doplnit MAF3 o podporu dialogových oken modulů.

Také možnosti uživatelského rozhraní v postranním panelu jsou omezené. Z metod modulu není možné přistupovat k objektu uživatelského rozhraní a tím je znemožněna například změna popisků a textu tlačítek či vyplnění položek do prvku rozhraní typu seznam. Tato funkce je v modulech platformy MAF2 často využívána a její nedostupnost v MAF3 omezuje funkci transformovaných modulů. Zdrojový kód platformy MAF3 je veřejně dostupný a jeho úpravou podle přílohy F by byl přístup k uživatelskému rozhraní modulu umožněn, avšak za cenu možné budoucí nekompatibility. Moduly transformované nástrojem využívají těchto úprav, ale jednotlivé příkazy přistupující k uživatelskému rozhraní není možné kvůli rozdílnému přístupu knihoven *wxWidgets* a *Qt* automaticky transformovat a uživatel musí provést úpravy ručně.

6.4.2 Nutné úpravy transformovaného kódu

Aby bylo možné použít transformované moduly v aplikaci založené na platformě MAF3, je nutné nejprve zkontrolovat a opravit problémy uvedené v logu transformace a případné prvky rozhraní nepodporované platformou MAF3 nahradit vhodnou alternativou.

Platforma MAF3 byla od základu nově vytvořena a nevychází ze zdrojových kódů svého předchůdce. Nový systém předávání zpráv, přesunutí některých funkcí do zásuvného modulu a další úpravy vyžadují změny ve všech modulech převedených z MAF2.

Také základní třídy převáděných modulů `mafVME`, `mafOperation` a `mafView` byly přepracovány. Transformační nástroj provede základní úpravy, jako je přejmenování metod na jejich nejbližší ekvivalent v MAF3, ale kód jednotlivých metod již nelze automaticky upravit.

Základní datový objekt platformy MAF2 `mafNode` není ve verzi MAF3 dostupný a namísto něj jsou použity potomci třídy `mafObject`. Tato abstraktní třída je obsažena v obou platformách jako základ pro většinu objektů, v případě platformy MAF3 došlo odstraněním třídy `mafNode` a dalšími úpravami k změně hierarchie tříd a převedené moduly je nutné odpovídajícím způsobem upravit. Sloty pro obsluhu jednotlivých prvků rozhraní jsou nástrojem vytvořeny z metody `OnEvent` a mohou stejně jako ostatní metody obsahovat kód závislý na platformě MAF2.

7 Závěr

Cílem této práce bylo prozkoumat rozdíly platformy MAF2 a platformy MAF3, zejména v oblasti grafického uživatelského rozhraní, a implementovat nástroj, který dokáže rozhraní jednotlivých modulů platformy MAF2 převést do nové verze. Dále bylo nutné seznámit se s knihovnami *wxWidgets* a *Qt4*, které platformy využívají k tvorbě uživatelského rozhraní a zvolit vhodný přístup k zpracování zdrojového kódu modulů platformy, jenž jsou napsané v jazyce C++.

Srovnáním obou verzí platformy byly zjištěna kromě očekávaných rozdílů plynoucích z použití odlišných knihoven také chybějící podpora dialogových oken modulů a omezení přístupu modulů k uživatelskému rozhraní v MAF3. Tyto nedostatky nebyly při zahájení práce známy a jsou způsobeny stále probíhajícím vývojem platformy. V rámci práce byla navržena úprava platformy MAF3, aby bylo možné měnit uživatelské rozhraní z metod modulu a tak zachovat funkce rozhraní shodné na obou platformách.

Implementovaný transformační nástroj dokázal zpracovat všechny poskytnuté moduly a vygenerovat pro ně soubory uživatelského rozhraní a upravené zdrojové kódy. Úspěšnost převodu lze ale hodnotit velmi obtížně, jediný objektivní ukazatel je počet nerozpoznaných příkazů, nepodporovaných prvků a direktiv preprocesoru zaznamenaných do log souboru. Tato hodnota ovšem nekoreluje se skutečným počtem chyb v upraveném modulu a s náročností nutných úprav. Vygenerované uživatelské rozhraní je podle vizuální kontroly ve většině případů správně rozvrženo a vyžaduje jen minimální úpravy dle záznamů v log souboru.

Nedostatkem nástroje je neschopnost zpracovat dynamicky generované části uživatelského rozhraní. Kvůli rozdílnosti obou platforem není možné definovat přesný postup nástroje při zpracování těchto příkazů. Vygenerované soubory modulu není možné bez dalších úprav použít, protože nástroj zpracovává pouze metody uživatelského rozhraní a jeho událostí. Pro úspěšný překlad modulu na nové platformě je nutné také upravit ostatní metody pro použití s třídami platformy MAF3.

Literatura

[1] CARRUTH, C. Refactoring C++ with Clang [online]. Google Inc., 2012. Dostupné z: http://www.youtube.com/watch?v=yuIOGfcOH0k.

[2] CARRUTH, C. Clang MapReduce - Automatic C++ Refactoring at Google Scale [online]. Google Inc., 2011. Dostupné z: http://www.youtube.com/watch?v=mVbDzTM21BQ.

[3] CLANG PROJECT. Clang vs Other Open Source Compilers [online], 2013. Dostupné z: http://clang.llvm.org/comparison.html.

[4] DEAN, J. – GHEMAWAT, S. *MapReduce: Simplified Data Processing on Large Clusters [online].* 2004. Dostupné z: http://research.google.com/archive/mapreduce.html.

[5] JULIAN SMART, V. Z. l. R. D. e. a. R. R. History of wxWidgets [online], 2007. Dostupné z: http://www.wxwidgets.org/about/history.htm.

[6] LATTNER, C. The LLVM Compiler Infrastructure [online], 2013. Dostupné z: http://www.llvm.org/.

[7] MOZILLA. Mozilla 2 - Semi-automated refactoring work/Oink [online], 2010. Dostupné z: https://wiki.mozilla.org/Mozilla_2.

[8] MOZILLA. Pork, 2008. Dostupné z: https://wiki.mozilla.org/Pork.

[9] MUCCI, R. *Events inside MAF3 [online].* OpenMaf, 2010. Dostupné z: https://www.biomedtown.org/biomed_town/MAF/MAF3%20Floor/documentation/events/.

[10] OPENMAF. *MAF2 Floor [online].* OpenMaf, 2009. Dostupné z: https://www.biomedtown.org/biomed_town/MAF/MAF2%20Floor/.

[11] PYTHON COMMUNITY. Python Programming Language [online], 2013. Dostupné z: http://www.python.org/.

[12] QT PROJECT. The Property System [online], 2011. Dostupné z: http://qt-project.org/doc/qt-4.8/properties.html.

[13] QT PROJECT. Qt Designer's UI File Format [online], 2010. Dostupné z: http://qt-project.org/doc/qt-4.8/designer-ui-file-format.html.

[14] QT PROJECT. Qt Features Overview [online], 2011. Dostupné z: http://qt-project.org/doc/qt-4.8/qt-overview.html.

[15] THE CLANG TEAM. Clang 3.3 documentation [online], 2013. Dostupné z: http://clang.llvm.org/docs/index.html.

[16] VICECONTI, M. MAF3: the story so far [online]. BioComputing Competence Centre, 2009. Dostupné z: https://www.biomedtown.org/biomed_town/MAF/MAF3%20Floor/history/maf3_documents/pres_maf3_june2009/pres_maf3_june2009_v4.pdf.

[17] WXWIDGETS COMMUNITY. WxWidgets Compared To Other Toolkits [online], 2012. Dostupné z: http://wiki.wxwidgets.org/WxWidgets_Compared_To_Other_Toolkits.

A Adresářová struktura DVD

python - Instalační soubory interpretu jazyka Python ve verzi 2.7.

qt creator - Instalační soubory aplikace Qt Creator pro zobrazení náhledu vygenerovaného souboru uživatelského rozhraní.

moduly maf - Moduly založené na platformě MAF2 pro otestování transformačního nástroje.

- **LHPApps** - Moduly aplikace LHPApps z projektu VPHOP WP10.
- **MAF2** - Moduly platformy MAF2.
- **MAF2 prevedene** - Moduly platformy MAF2 transformované pomocí nástroje.

transformační nástroj - Soubory transformačního programu vytvořeného v rámci této bakalářské práce.

- **src** - Zdrojový kód nástroje. Nástroj lze spustit pomocí instalovaného interpretu jazyka Python příkazem `python maf_convert.py`.
- **win32_bin** - Transformační nástroj ve formě spustitelného souboru pro operační systém MS Windows. Použití nástroje v této formě nevyžaduje instalaci interpretu jazyka Python.

bp.pdf - Text této práce ve formátu PDF.

index.html - Soubor s popisem obsahu DVD.

B Uživatelský manuál

B.1 Instalace interpretu jazyka Python

Transformační nástroj je vytvořen v jazyce Python a pro jeho použití je nutné mít nainstalovaný interpret tohoto jazyka. Nástroj využívá Python verze 2.7 a modulu pro zpracování XML souboru *lxml*.

Interpret jazyka pro platformu MS Windows je dostupný z oficiálních stránek projektu `www.python.org` a modul *lxml* z webového katalogu balíčků *PyPI* dostupného na adrese `https://pypi.python.org/pypi/lxml/`. Na přiloženém CD jsou umístěny instalační soubory interpretu, modulu *lxml* a verze transformačního nástroje ve formě spustitelného souboru s integrovaným interpretem. Interpret jazyka i přídavný modul lze nainstalovat jako běžný program pomocí grafického instalátoru. Po instalaci je vhodné zkontrolovat, zda proměnná systému obsahuje cestu k spustitelnému souboru `python.exe`.

Většina distribucí operačního systému GNU/Linux obsahuje interpret jazyka Python v základní instalaci, stejně jako operační systém Mac OS X. Modul *lxml* pro tyto platformy je možné získat také z katalogu balíčků *PyPI*, nebo přímo z repozitářů konkrétní distribuce operačního systému.

B.2 Použití nástroje

Po instalaci interpretu jazyka Python je nástroj spustitelný z příkazové řádky příkazem `python maf_convert.py nazevsouboru.cpp`. Název vstupního souboru je jediným povinným argumentem aplikace a pokud není zadán, vypíše program zprávu, jak nástroj správně použít s přehledem dostupných argumentů v následujícím tvaru:

```
Usage: maf_convert.py INPUT.cpp [OPTIONS]
Available options are:
 -csv   - Print end message as CSV formatted string (for data analysis).
 -d     - Set output directory (default="output").
 -html  - Save log in HTML format.
```

Význam jednotlivých argumentů nástroje:

-csv - Po skončení transformace je do příkazového řádku vypsána zpráva o počtu identifikovaných problematických příkazů. Volbou `-csv` budou jednotlivé hodnoty odděleny středníkem. S využitím přesměrování výstupu programu je tento formát vhodný pro analýzu dat.

-d - Volba `-d` umožňuje nastavit adresář, do kterého budou uložena transformovaná data.

-html - Log je uložen namísto textového souboru do souboru ve formátu HTML s barevně odlišenými typy záznamů.

Po úspěšné transformaci jsou do zadaného adresáře umístěny zpracované soubory a log soubor se záznamem zjištěných problémů. Dále nástroj vypíše zprávu s počty nerozpoznaných příkazů, nepodporovaných prvků a direktiv preprocesoru. Zprávy pro modul pojmenovaný `JmenoModulu` s nulovým počtem problematických příkazů bude v následujícím tvaru:

```
Transformation of JmenoModulu completed. Unknown commands: 0,
Unsupported widgets: 0, Preprocessor directives: 0
```

Pokud zpracovávaný modul neobsahuje metodu s definicí uživatelského rozhraní, v metodě není nalezen objekt uživatelského rozhraní, nebo hlavičkový soubor neobsahuje deklaraci, je vypsána chybová hláška s názvem třídy a popisem problému a program je ukončen. Například v případě, kdy v metodě není nalezen objekt uživatelského rozhraní by zpráva vypadala takto:

```
Method CreateGui in class JmenoModulu doesn't contain GUI object.
```

C Výsledek transformace modulů

Následující tabulky uvádí výsledek transformace jednotlivých modulů z projektů *MAF2*, *LHPApps* a *MAF2 - Medical*. V tabulce jsou zelenou barvou zvýrazněny moduly s nulovým počtem záznamů nerozpoznaných příkazů, nepodporovaných prvků a direktiv preprocesoru. Naopak červenou barvou jsou zvýrazněny moduly s celkově více než deseti záznamy.

Název modulu	U	W	P
MAF2 - Operations			
mafOpApplyTrajectory	0	0	0
mafOpClipSurface	3	0	0
mafOpConnectivitySurface	18	0	0
mafOpCrop	9	0	0
mafOpExtractIsosurface	40	0	0
mafOpFilterSurface	0	0	0
mafOpFilterVolume	0	0	0
mafOpImporterMesh	0	0	0
mafOpImporterVMEDataSetAttributes	0	0	0
mafOpMAFTransform	4	1	0
mafOpRemoveCells	21	0	0
mafOpVolumeResample	1	0	0
MAF2 - VME			
mafPipeIsosurface	3	0	0
mafPipeIsosurfaceGPU	3	0	0
mafPipeLandmarkCloud	2	2	0
mafPipeMesh	1	3	0
mafPipeMeshSlice	1	3	0
mafPipeMeter	3	2	0
mafPipePolyline	7	2	0
mafPipePolylineSlice	1	0	0
mafPipeScalar	0	0	0
mafPipeScalarMatrix	0	0	0
mafPipeSurface	9	3	0
mafPipeSurfaceSlice	1	0	0
mafPipeSurfaceTextured	7	3	0

U - Nerozpoznaný příkaz, W - Netransformovaný příkaz, P - Direktiva preprocesoru

Název modulu	U	W	P
mafPipeVector	1	1	0
mafPipeVolumeSlice	7	2	0
mafVMEGenericAbstract	1	0	1
mafVMEGroup	1	0	0
mafVMEInfoText	10	0	0
mafVMELandmark	5	0	0
mafVMELandmarkCloud	2	0	0
mafVMEMeter	6	0	0
mafVMEOutputImage	3	0	0
mafVMEOutputLandmarkCloud	3	0	0
mafVMEOutputMesh	6	0	0
mafVMEOutputMeter	4	0	0
mafVMEOutputPointSet	6	0	0
mafVMEOutputPolyline	7	0	0
mafVMEOutputScalar	4	0	0
mafVMEOutputScalarMatrix	8	0	0
mafVMEOutputSurface	3	0	0
mafVMEOutputVolume	8	0	0
mafVMEPolylineSpline	4	0	0
mafVMEProber	6	0	0
mafVMERefSys	23	0	0
mafVMEScalarMatrix	2	0	0
mafVMESlicer	3	0	0
mafVMEVolumeGray	1	0	0
mafVisualPipeVolumeRayCasting	1	0	0
LHPApps - Operations			
lhpGuiDialogCreateOpenSimModelWorkflow	11	0	0
lhpOp3DVolumeBrush	1	0	0
lhpOpBonemat	11	0	1
lhpOpBonematBatch	0	0	0
lhpOpComputeGradientStructureTensor	0	0	0
lhpOpComputeHausdorffDistance	0	0	0
lhpOpComputeProgressiveHull	20	0	1
lhpOpComputeTensor	21	0	2
lhpOpEditTagRefactor	1	0	0
lhpOpExporterAnsysInputFile	1	0	0
lhpOpExporterMatrixOctave	0	0	0

U - Nerozpoznaný příkaz, W - Netransformovaný příkaz, P - Direktiva preprocesoru

Název modulu	U	W	P
lhpOpExporterMeshFeBio	0	0	0
lhpOpExtractGeometry	3	2	0
lhpOpExtractLandmarkEOS	28	2	0
lhpOpImporterC3D	8	0	0
lhpOpImporterC3DBTK	8	0	0
lhpOpImporterMatrixOctave	2	0	0
lhpOpMeshGeneratorExecutable	0	0	0
lhpOpModifyOpenSimModel	0	0	0
lhpOpModifyOpenSimModelCreateBallJoint	0	0	0
lhpOpModifyOpenSimModelCreateBodyFromSurface	0	0	0
lhpOpModifyOpenSimModelCreateBodyFromSurfacesGroup	0	0	0
lhpOpModifyOpenSimModelCreateCustomJoint	0	0	0
lhpOpModifyOpenSimModelCreateFreeJoint	0	0	0
lhpOpModifyOpenSimModelCreateGeometry	0	0	0
lhpOpModifyOpenSimModelCreateMarkerSet	0	0	0
lhpOpModifyOpenSimModelCreateMuscle	0	0	0
lhpOpModifyOpenSimModelCreateMuscleFromMeter	0	0	0
lhpOpModifyOpenSimModelCreateMuscleMultiPoint	0	0	0
lhpOpModifyOpenSimModelCreatePinJoint	0	0	0
lhpOpModifyOpenSimModelRunIDSimulation	0	0	0
lhpOpModifyOpenSimModelRunIKSimulation	0	0	0
lhpOpModifyOpenSimModelRunIKSimulationAPI	0	0	0
lhpOpMultiscaleExplore	13	0	0
lhpOpMuscleDecompose	18	0	1
lhpOpScaleMMA	4	0	0
lhpOpShowHistogram	9	1	0
lhpOpTextureOrientation	11	0	0
lhpViewVolumeBrush	4	1	0
mmoAFSys	4	0	0
mmoAverageLM	1	0	0
mmoBuildHierarchy	1	0	0
mmoHelAxis	0	0	0
mmoStickPalpation	1	0	0
mmoTimeReduce	4	0	0
LHPApps - Views			
lhpViewAnalogGraphXY	0	0	0
mafViewIntGraph	9	0	0

U - Nerozpoznaný příkaz, W - Netransformovaný příkaz, P - Direktiva preprocesoru

Název modulu	U	W	P
LHPApps - VME			
lhpOpSetLowerRes	1	0	1
lhpPipeAnalogGraphXY	1	0	0
lhpVMEPoseGroup	1	0	0
lhpVMEScalarMatrix	1	0	0
lhpVMESurfaceScalarVarying	9	0	0
lhpVisualPipeSurfaceScalar	2	1	0
mafVMEAFRefSys	12	0	0
mafVMEHelAxis	3	0	0
mafVMEShortcut	6	0	2
medPipeJoint	1	0	0
medVMEJoint	14	0	14
medVMEMuscleWrapper	15	0	16
medVMEMusculoSkeletalModel	8	0	6
medVMEOutputJoint	3	0	0
MAF2Medical - Operations			
medGeometryEditorPolylineGraph	1	0	0
medOpClassicICPRegistration	5	0	0
medOpCleanSurface	0	0	0
medOpComputeInertialTensor	1	0	0
medOpCreateEditSkeleton	0	1	0
medOpEqualizeHistogram	0	0	0
medOpExporterGRFWS	2	0	0
medOpExporterMeters	1	0	0
medOpExtractGeometry	3	2	0
medOpExtrusionHoles	15	0	0
medOpFillHoles	15	0	0
medOpFlipNormals	21	0	0
medOpImporterRAWImages	27	0	8
medOpInteractionDebugger	2	1	0
medOpInteractiveClipSurface	2	3	0
medOpIterativeRegistration	4	2	0
medOpLabelizeSurface	3	1	0
medOpMakeVMETimevarying	0	0	0
medOpMeshDeformation	25	0	2
medOpMeshQuality	13	0	0

U - Nerozpoznaný příkaz, W - Netransformovaný příkaz, P - Direktiva preprocesoru

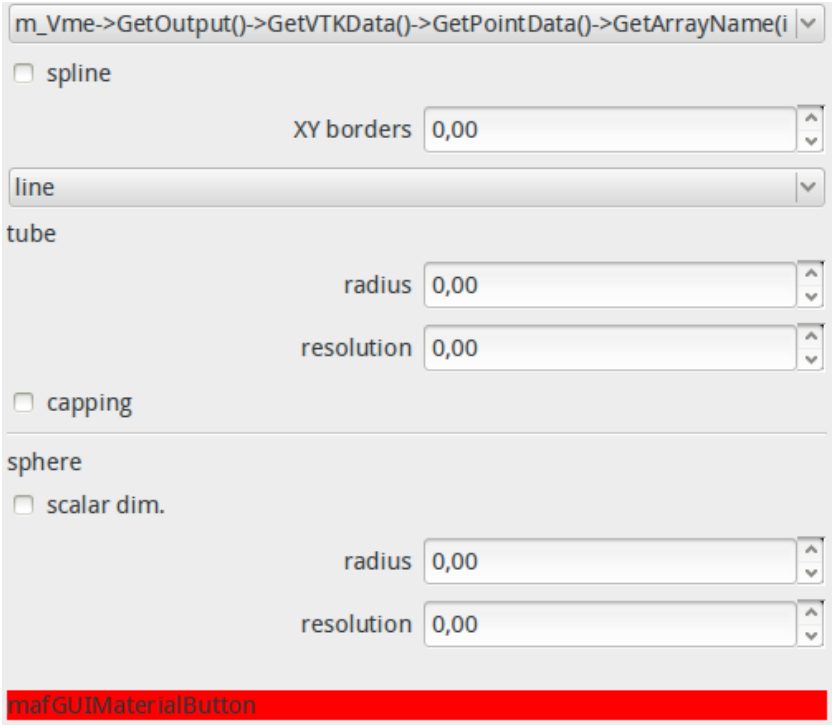
Název modulu	U	W	P
medOpMove	3	1	0
medOpScaleDataset	5	2	0
medOpSegmentation	65	3	0
medOpSegmentationRegionGrowingConnectedThreshold	0	0	0
medOpSegmentationRegionGrowingLocalAndGlobalThresho	12	1	0
medOpSmoothSurface	0	0	0
medOpSmoothSurfaceCells	20	0	0
medOpSplitSurface	3	0	0
medOpSubdivide	1	0	0
medOpTriangulateSurface	0	0	0
medOpVolumeResample	4	0	0
MAF2Medical - Views			
mafView3D	10	0	0
mafViewArbitrarySlice	2	1	0
mafViewCT	1	0	0
mafViewGlobalSlice	6	0	0
mafViewGlobalSliceCompound	1	2	0
mafViewImage	2	1	0
mafViewImageCompound	1	2	0
mafViewRX	2	1	0
mafViewRXCT	14	0	0
mafViewRXCompound	4	0	0
mafViewSingleSlice	2	0	0
mafViewSingleSliceCompound	1	2	0
mafViewSlice	7	1	0
medViewArbitraryOrthoSlice	2	1	0
medViewSliceBlend	7	0	0
medViewSliceBlendRX	4	1	0
medViewSliceGlobal	6	0	0
medViewSliceNotInterpolated	0	1	0
medViewSliceNotInterpolatedCompound	0	1	0
medViewSlicer	2	1	0
medViewVTKCompound	2	1	0
MAF2Medical - VME			
medPipeComputeWrapping	3	2	0
medPipeDensityDistance	22	0	0

U - Nerozpoznaný příkaz, W - Netransformovaný příkaz, P - Direktiva preprocesoru

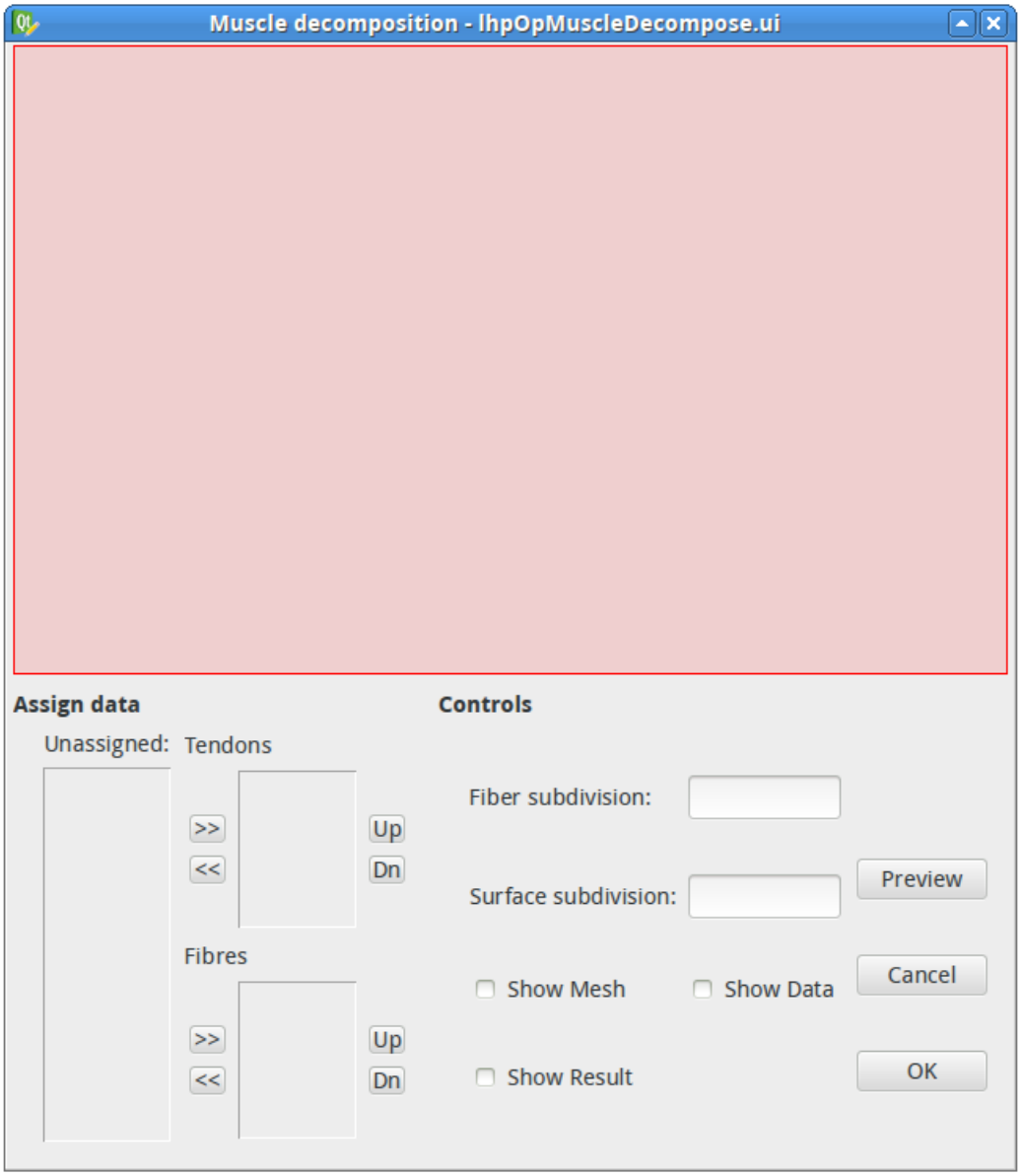
Název modulu	U	W	P
medPipeGraph	19	0	0
medPipePolylineGraphEditor	1	0	0
medPipeRayCast	0	0	0
medPipeSurfaceEditor	1	0	0
medPipeTensorFieldGlyphs	20	2	5
medPipeTensorFieldSurface	10	1	0
medPipeTrajectories	0	0	0
medPipeVectorFieldGlyphs	21	2	5
medPipeVectorFieldMapWithArrows	11	1	0
medPipeVectorFieldSurface	9	1	0
medPipeVolumeDRR	13	0	0
medPipeVolumeMIP	3	1	0
medPipeVolumeSliceBlend	0	1	0
medPipeVolumeSliceNotInterpolated	0	1	0
medPipeVolumeVR	1	0	0
medPipeWrappedMeter	3	2	0
medVMEAnalog	1	0	0
medVMELabeledVolume	24	0	0
medVMEMaps	2	0	0
medVMEOutputComputeWrapping	6	0	0
medVMEOutputPolylineEditor	2	0	0
medVMEOutputSurfaceEditor	2	0	0
medVMEOutputWrappedMeter	15	0	0
medVMESegmentationVolume	3	0	0
medVMEWrappedMeter	31	0	0
medVisualPipeCollisionDetection	0	0	0
medVisualPipePolylineGraph	3	0	0
medVisualPipeSlicerSlice	1	0	0

U - Nerozpoznaný příkaz, W - Netransformovaný příkaz, P - Direktiva preprocesoru

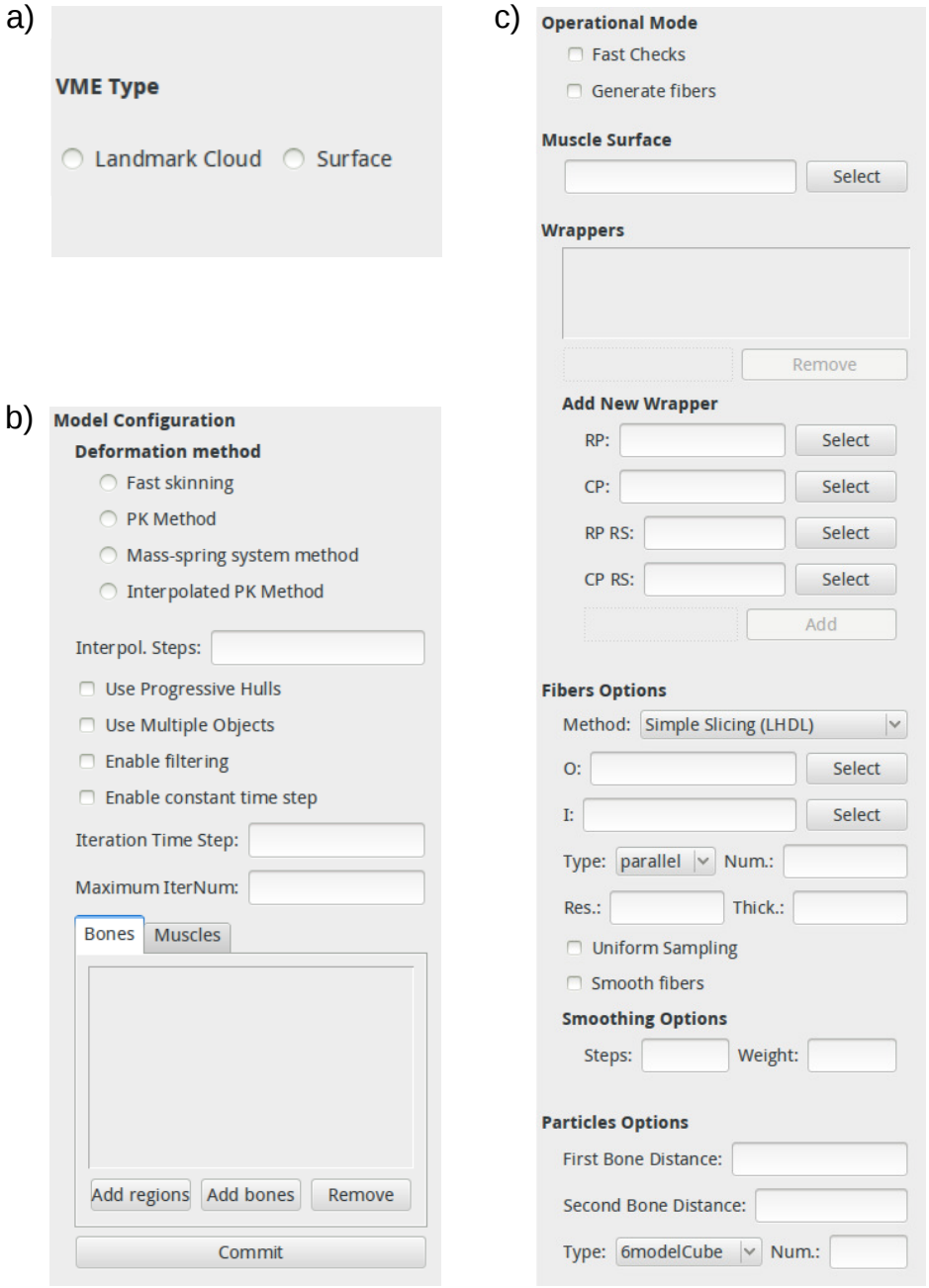
D Vygenerované uživatelské rozhraní



Obrázek D.1: Uživatelské rozhraní panelu se seznamem obsahujícím položky získané z výstupu funkce a s výrazně formátovaným prvkem, nahrazujícím prvek neimplementovaný v platformě MAF3.



Obrázek D.2: Nástroj dokáže transformovat také dialogová okna modulů, ale jejich podpora není v platformě MAF3 implementována. Červenou barvou je vyznačen prvek zobrazující náhled operace.



Obrázek D.3: Uživatelské rozhraní postranního panelu.
a) Jednoduché rozhraní skládající se pouze z dvou prvků.
b) Rozhraní s panely a prvky typu *Radio button*.
c) Komplexní rozhraní s deaktivovanými prvky a rozbalovacími seznamy.

E Vygenerovaný log soubor

moduleName

Method CreateGui/CreateOPDialog begins at line 273:

```
Unknown command at line 294:
294   m_FrequencyFileName = wxGetWorkingDirectory();

Unknown command at line 295:
295   m_FrequencyFileName += "\\" ;

Preprocessor directive at line 302:
302  #ifdef _DEBUG_BONEMAT_GUI
304   const wxString densityRelationship[] = {"points
coordinates","intercept slope"};
305   m_Gui->Label("density relationship");
306   m_Gui->Combo(ID_DENSITY_RELATIONSHIP_LISTBOX, "",
&m_DensityRelationshipListbox, 2, densityRelationship);
309   m_Gui->Double(ID_HU0,"HU0", &m_HU0_d0_el0);
315   m_Gui->Enable(ID_HU0, false);
321   m_Gui->Divider(2);
322  #endif

Comment at line 414:
414  // EnableTwoIntervals(true);

Unknown command at line 421:
421   DisableRhoAshThreeIntervals();

Comment at line 423:
423  /*
424   m_Gui->Double(ID_SECOND_EXPONENTIAL_COEFFICIENTS_VECTOR_V3, "",
m_Ea1_Eb1_Ec1_V3);
425   m_Gui->Label("hu threshold");
426   m_Gui->Double(::ID_HU_THRESHOLD, "", &m_HUThreshold);*/
```

Method OnEvent begins at line 483:

```
Comment at line 493:
493  //WORKAROUND CODE
```

Obrázek E.1: Log soubor uložený ve formě HTML dokumentu.

```
1
2  Method CreateGui/CreateOPDialog begins at line 273:
3
4  Unknown command at line 294:
5        m_FrequencyFileName = wxGetWorkingDirectory();
6
7  Unknown command at line 295:
8        m_FrequencyFileName +=  "\\" ;
9
10 Preprocessor directive at line 302:
11     #ifdef _DEBUG_BONEMAT_GUI
12
13  at line 304:
14       const wxString densityRelationship[] = {"points coordinates","
             intercept slope"};
15
16  at line 305:
17       m_Gui->Label("density relationship");
18
19  at line 306:
20       m_Gui->Combo(ID_DENSITY_RELATIONSHIP_LISTBOX, "", &
             m_DensityRelationshipListbox, 2, densityRelationship);
21
22  at line 309:
23       m_Gui->Double(ID_HU0,"HU0", &m_HU0_d0_el0);
24
25  at line 315:
26       m_Gui->Enable(ID_HU0, false);
27
28  at line 321:
29       m_Gui->Divider(2);
30
31  at line 322:
32     #endif
33
34 Comment at line 414:
35     //   EnableTwoIntervals(true);
36
37 Unknown command at line 421:
38       DisableRhoAshThreeIntervals();
39
40 Comment at line 423:
41     /*
42  at line 424:
43       m_Gui->Double(ID_SECOND_EXPONENTIAL_COEFFICIENTS_VECTOR_V3, "",
             m_Ea1_Eb1_Ec1_V3);
44
45  at line 425:
46       m_Gui->Label("hu threshold");
47
48  at line 426:
49       m_Gui->Double(::ID_HU_THRESHOLD, "", &m_HUThreshold);*/
50
51 Method OnEvent begins at line 483:
52
53 Comment at line 493:
54     //WORKAROUND CODE
```

Fragment kódu E.1: Log soubor uložený ve formě textového souboru.

F Úprava zdrojového kódu MAF3

Následující části zdrojového kódu popisují změny v platformě MAF3 umožňující přístup modulu k objektu uživatelského rozhraní. Řádky označené světle zelenou barvou jsou nově vložené, čísla řádek odpovídají umístění v souboru.

```
26 ...
27 void mafOperationWidget::setOperation(mafCore::mafObjectBase *op)
28 {
29   m_Operation = op;
30   connect(this,SIGNAL(operationUILoaded()),m_Operation, SLOT(on_gui_loaded()
         ));
31 }
32 ...
```

```
65 ...
66     ui->verticalLayoutOperation->addWidget(m_OperationGUI);
67     Q_EMIT operationUILoaded();
68 }
69 ...
```

Fragment kódu F.1: Úpravy v souboru mafOperationWidget.cpp

```
61 ...
62     /// Signal to alert the observet that the operation GUI has been
           dismissed.
63     void operationDismissed();
65   /// Signal to inform mafOperation that it's GUI has been loaded
66     void operationUILoaded();
67 ...
```

Fragment kódu F.2: Úpravy v souboru mafOperationWidget.h

```
95 ...
96
97 void mafOperation::on_gui_loaded(){
98
99 }
```

Fragment kódu F.3: Úpravy v souboru mafOperation.cpp

```
60 ...
61     /// slot called when interactingSignal from connected interactor is
           emitted.
62     virtual void updateFromInteraction();
63   /// slot called when operationUILoaded signal from mafOperationWidget is
       emitted
64     virtual void on_gui_loaded();
65
66 private Q_SLOTS:
67 ...
```

Fragment kódu F.4: Úpravy v souboru mafOperation.h