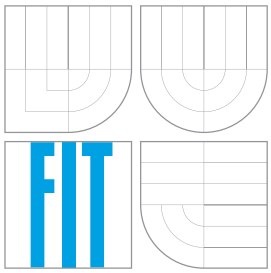
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

AUTOMATICKÁ TVORBA GRAFICKÉHO ROZHRANÍ PRO VZDÁLENÝ PŘÍSTUP

AUTOMATIC USER INTERFACE DESIGN FOR REMOTE ACCESS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JAN KORIŤÁK

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. Dr. Ing. PAVEL ZEMČÍK

BRNO 2011

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2010/2011

Zadání diplomové práce

Řešitel: **Koriťák Jan, Bc.**
Obor: Počítačová grafika a multimédia
Téma: **Automatická tvorba grafického rozhraní pro vzdálený přístup**
Automatic User Interface Design for Remote Access
Kategorie: Uživatelská rozhraní

Pokyny:

1. Prostudujte dostupnou literaturu na téma automatická konstrukce uživatelských rozhraní.
2. Navrhněte po konzultaci vhodný typ rozhraní pro automatickou implemetnaci se zaměřením na zpracování audio a video streamů.
3. Navrhněte postup implementace výše uvedeného systému tvorby rozhraní.
4. Implementaujte rozhraní podle předchozího bodu zadání.
5. Diskutujte dosažené výsledky a možnosti pokračování práce.

Literatura:

- Dle pokynů vedoucího

Při obhajobě semestrální části diplomového projektu je požadováno:

- Dle bodů 1-3 zadání

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Zemčík Pavel, doc. Dr. Ing.**, UPGM FIT VUT
Datum zadání: 20. září 2010
Datum odevzdání: 25. května 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2
L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Tato práce zkoumá možnosti abstrakce grafického uživatelského rozhraní a automatizace procesu jeho návrhu a tvorby za provozu aplikace. Důraz je kladen na možnost ovládání aplikace na dálku přes počítačovou síť a tedy na delegaci procesu tvorby rozhraní na vzdálený stroj obsluhovaný uživatelem. Práce analyzuje současné přístupy a metody pro generování rozhraní. V rámci práce byl navržen a implementován systém pro automatické generování uživatelského rozhraní vzdálené aplikace na základě specifikovaného datového modelu.

Abstract

This thesis explores the possibilities of abstraction of graphical user interface and automation of its design and development process at runtime. Emphasis is placed on possibility of remote control over computer network and accordingly on delegation of the interface creation on the human operated remote machine. The thesis analyzes contemporary approaches to and methods of interface generation. As a goal of the thesis a system for automatic user interface generation for remote applications was designed and implemented. The generation is based on specified data model.

Klíčová slova

GUI, uživatelské rozhraní, generování uživatelského rozhraní, ovládání na dálku, vzdálený přístup, Qt

Keywords

GUI, user interface, user interface generation, remote control, remote access, Qt

Citace

Jan Koriťák: Automatická tvorba grafického rozhraní pro vzdálený přístup, diplomová práce, Brno, FIT VUT v Brně, 2011

Automatická tvorba grafického rozhraní pro vzdálený přístup

Prohlášení

Prohlašuji, že jsem tento projekt vypracoval samostatně pod vedením pana Doc. Dr. Ing. Pavla Zemčíka.

.....
Jan Koriťák
25. května 2011

Poděkování

Rád bych poděkoval svému vedoucímu p. Doc. Dr. Ing. Pavlu Zemčíkovi za vedení práce a p. Luborovi Přikrylovi ze společnosti DISK Multimedia s.r.o. za konzultace a iniciativu vedoucí ke vzniku práce.

Obsah

Kapitola 1

Úvod

Osobní počítače a další zařízení založená na počítačích se staly běžnou součástí našich životů, aniž si to často uvědomujeme. Žijeme v době, kdy téměř každý z nás běžně nosí alespoň jedno takové zařízení ve své kapse. Počítače používá široká veřejnost bez nutnosti zvláštních odborných znalostí.

Nedílnou a důležitou součástí počítačových aplikací i jednoúčelových zařízení je vrstva mezi samotným systémem a jeho uživatelem – uživatelské rozhraní. Je to právě uživatelské rozhraní, které nám umožňuje využívat všeho, co nám počítačové systémy nabízí, aniž bychom museli mít jakékoliv technické znalosti o procesech odehrávajících se uvnitř našeho zařízení.

Uživatelské rozhraní je běžně navrhováno a vytvářeno vývojářem v době přípravy aplikace. Tento proces se může s přibývající složitostí aplikace stát velice pracným. U větších projektů pak může být čas strávený prací na uživatelském rozhraní značně nevyvážen funkcionalitě, která je během této práce do produktu přidávána.

Tato práce zkoumá možnosti abstrakce uživatelského rozhraní a automatizace procesu jeho návrhu a tvorby počítačem za provozu aplikace. Důraz je kladen na možnost vzdáleného ovládání aplikace přes počítačovou síť, při kterém obsluhovaná aplikace běží na jiném stroji, než na kterém se vytváří a zobrazuje uživatelské rozhraní.

Impulzem pro vznik této práce byl projekt společnosti DISK Multimedia, s.r.o. zaměřený na univerzální systém pro číslicové zpracování signálů s možností vzdáleného ovládání přes počítačovou síť.

Kapitola 2 představuje současné pohledy na tvorbu uživatelského rozhraní a motivaci pro alternativní přístupy pro tvorbu rozhraní.

Stávající řešení a přístupy pro automatické generování rozhraní a vzdálené ovládání shrnuje kapitola 3.

V kapitole 4 jsou zhodnoceny představené technologie a současný formát serializace dat navržený společností DISK Multimedia, s.r.o.

Kapitola 5 se zabývá jednotlivými aspekty návrhu systému pro automatické generování rozhraní. Je definován model generovaného rozhraní, způsob jeho reprezentace a distribuce.

Samotná realizace navrženého systému je popsána v kapitole 6.

Kapitola 2

Uživatelské rozhraní

Tato kapitola představuje současné metody tvorby grafického uživatelského rozhraní (dále jen GUI) a popisuje motivaci k novým přístupům. Shrnuje současné poznatky týkající se těchto nových přístupů.

Vývoj uživatelského rozhraní obvykle představuje nezanedbatelnou (někdy zcela zásadní) část práce na softwarovém produktu. Realizace každé funkce produktu, u níž se počítá s uživatelskou interakcí, s sebou nese potřebu navrhnout a implementovat veškeré prvky uživatelského rozhraní, které se této funkce týkají.

Zvláštní pozornost věnovaná uživatelskému rozhraní je většinou opodstatněná, hlavně klademe-li důraz na prezentační úroveň produktu. Můžeme se však dostat do takové situace, kdy se práce na tvorbě a rozšiřování rozhraní stane monotónní a získá formu neustálého opakování stejných úkonů vždy upravených pro daný účel. Při každém požadavku na novou funkčnost aplikační logiky se k našim úkolům řadí i úvaha o podobě rozhraní a jeho implementace.

Dnes máme k dispozici řadu nástrojů (využívajících např. vizuálního návrhu rozhraní), pomocí nichž můžeme implementaci rozhraní zvládnout velice rychle. Stále však určitá režie zůstává.

Vzniká tu tedy určitá motivace ke snaze nějakým způsobem abstrahovat data na vstupech a výstupech funkčních celků, které tvoříme. Pokud by existovala nějaká vrstva, která by dokázala z funkčních částí získat informace o podobě vstupních a výstupních dat a automatizovaně k těmto datům vygenerovat rozhraní, odpadla by nám nutnost zabývat se uživatelským rozhraním. To bychom ocenili hlavně v aplikacích, kdy jsme nuceni často upravovat logiku aplikace podle nových požadavků. Navíc by se nám otevřely možnosti spoluúčasti uživatelů na tvorbě rozhraní, kteří by si mohli způsob generování parametrizovat a přizpůsobit si rozhraní svým potřebám.

Takovéto řešení má pochopitelně svá úskalí (např. určitou ztrátu kontroly nad uživatelským rozhraním) a je potřeba vždy vyhodnotit, zda se nám vyplatí jej aplikovat.

2.1 Model–based generování GUI

Účel technologií vývoje uživatelského rozhraní založeném na modelu (dále jen model–based) je poskytnout prostředí, ve kterém vývojáři mohou navrhovat a implementovat uživatelská rozhraní profesionálně a systematicky, avšak jednodušeji, než při použití konvenčních nástrojů.

Využívá se popisů rozhraní pomocí deklarativních modelů. Modelování může probíhat

na různých úrovních abstrakce. Usnadňuje inkrementální vývoj a znovupoužitelnost specifikací rozhraní a poskytuje infrastrukturu potřebnou k automatizování úloh souvisejících s návrhem a implementací rozhraní.

Nevýhodou je složitost modelů a jejich notací. Je obtížné demonstrovat, že modely popisují relevantní aspekty rozhraní nutné ke generování fungujících uživatelských rozhraní. Lze najít pouze pár příkladů použitelných rozhraní vygenerovaných z deklarativních modelů.

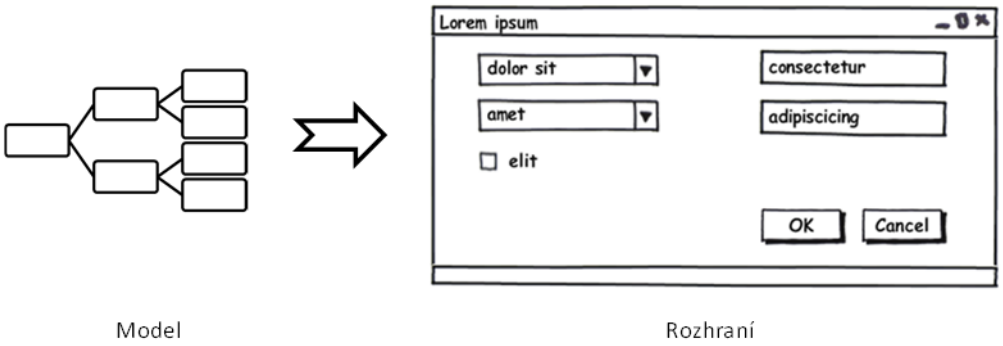
Otázkou stále zůstává nejen to, jaký druh modelů je nejvhodnější, ale i které aspekty uživatelského rozhraní se vyplatí modelovat, a které nikoliv.[9]

Práce [15] shrnuje problematiku takto:

> „Přístup generování uživatelského rozhraní založený na modelech je novým paradigmatem ve vývoji interaktivních systémů. Jako s každým novým paradigmatem s ním přicházejí nové náročné problémy k vyřešení. Současné postupy generování rozhraní založené na modelech byly kritizovány, protože generovaný kód neposkytoval kvalitní grafický výstup, nepodporoval přímou manipulaci a nedovoloval návrhářům realizovat všechny význačné detaily rozhraní. Oponovali jsme tvrzením, že tyto nedostatky nejsou inherentní v model–based přístupu, ale jsou spíše důsledkem aplikace tradičních technik tvorby software v novém paradigmatu.“

Dále je uvedeno stanovisko práce [13]:

> „Přes významné výzkumné úsilí se model–based nástroje nestaly běžně používanými, částečně protože tvorba modelů je abstraktní proces a lepších výsledků často dosáhne lidský vývojář za kratší čas. Ve specifický oblastech lze mluvit o jistém úspěchu, například u návrhu dialogových boxů a dálkových ovladačů. Považujeme model–based techniky za slibné, avšak za předpokladu, že rozsah generovaných rozhraní je udržován na zvladatelné úrovni.“



Obrázek 2.1: Model–based generování rozhraní

2.2 Distribuované GUI

U centralizovaných síťových aplikací se potýkáme s nutností reflektovat jakékoliv změny v software na všech klientských počítačích. Toto může představovat neúnosnou zátěž v systémech s velkým počtem uživatelů a častými změnami aplikace.

Zde nám vzniká motivace pro použití architektury tenkých klientů. Klientský software bude implementovat pouze základní elementy logiky a uživatelského rozhraní, které bez dalších změn budou schopné stále komunikovat s aktualizovanou serverovou aplikací a v plné míře ji využívat.

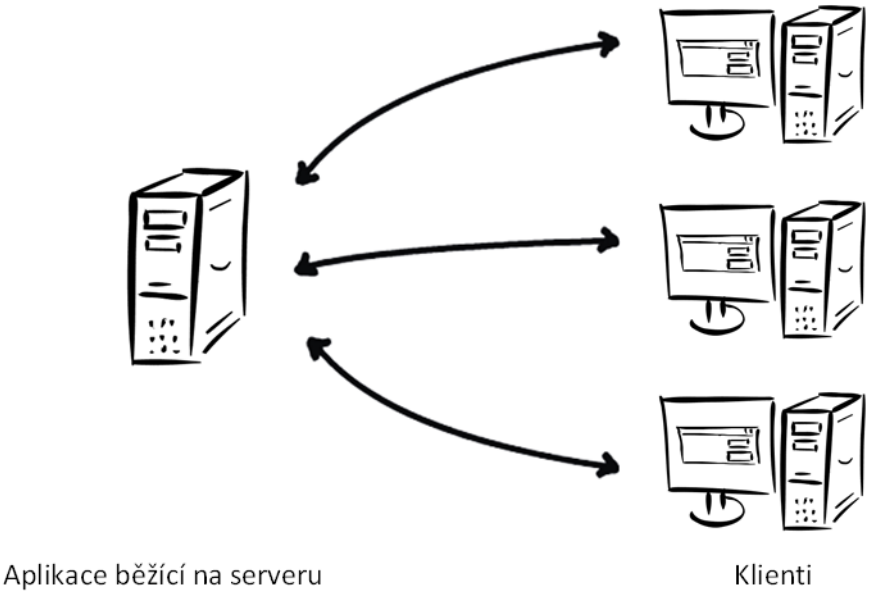
Z pohledu uživatelského rozhraní lze tohoto dosáhnout na různých úrovních.

S přihlédnutím k paradigmatu model–based systémů uživatelského rozhraní může server pouze poskytnout klientovi model a nechat v jeho plné režii vytvoření a obsluhu rozhraní. Klient následně odešle serveru pouze data získaná pomocí tohoto rozhraní.

Jinou možností je detailní specifikování rozhraní serverem a jeho odeslání klientovi v domluveném formátu. Klient je vybaven mechanismy, pomocí nichž může požadované rozhraní jednoznačně vytvořit, obsluhovat a zaslat zpět data z něj získaná.

Nejuniverzálnější přístup, který však představuje největší zátěž na komunikaci mezi klientem a serverem, spočívá v tom, že server zasílá klientovi instrukce k elementárním grafickým úkonům (nakresli čtverec, vypiš text) a naopak klient zasílá serveru informace o nastalých událostech (uživatel klikl myší, stiskl klávesu).

Jaký přístup je v dané situaci nejvhodnější závisí na konkretních požadavcích na složitost mechanismu uživatelského rozhraní a na objem přenesených dat mezi klientem a serverem.



Obrázek 2.2: Distribuované GUI

Kapitola 3

Stávající řešení pro generované a distribuované GUI

Tato kapitola popisuje existující technologie pro dynamické generování uživatelského rozhraní a pro vzdálený přístup. Dále představuje některé současné a vznikající technologie a metody pro automatizaci procesu generování rozhraní.

3.1 HTML

HTML (HyperText Markup Language) a související webové technologie jsou masově rozšířeným řešením distribuovaného GUI, s kterým se setkáváme každý den.

Pomocí předem dohodnutého standardu – HTML – server specifikuje strukturu rozhraní včetně možnosti detailně ovlivnit způsob jeho prezentace pomocí kaskádových stylů CSS (Cascading Style Sheets).

Specifikované rozhraní (kód v jazyce HTML) představuje relativně malý datový soubor, který se jednorázově přenese po síti pro každý uživatelův požadavek o zobrazení stránky.

Software na straně klienta (webový prohlížeč) má za úkol interpretovat specifikaci rozhraní, správně jej vykreslit a zajišťovat interakci s uživatelem.

Server není dále nijak zapojen do procesu interakce uživatele s GUI s výjimkou předem definovaných okamžiků (odeslání vyplněného formuláře, použití hypertextového odkazu). V tento daný okamžik klient odešle serveru data získaná od uživatele prostřednictvím GUI (např. vyplněná formulářová data). Server má možnost tato data zpracovat v rámci logiky celé aplikace a následně odešle klientovi novou specifikaci GUI, se kterým uživatel bude dále pracovat.

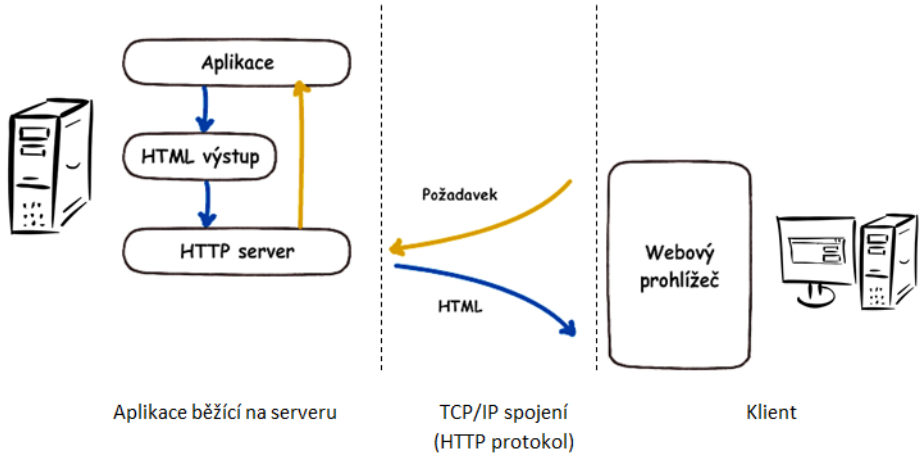
Celé řešení tedy spočívá v přenosech relativně malých datových souborů mezi klientem a serverem a to v předem definovaných okamžicích.

Veškerá logika aplikace zůstává na serveru, přenáší se pouze popis GUI. Tím je tedy zajištěna možnost kdykoliv změnit funkčnost aplikace (a to i ve zcela zásadní míře), bez nutnosti zásahu do software klientských počítačů (kterých je v případě uživatelů internetu obrovské množství).

Specifikace rozhraní pomocí HTML podléhá standardům, které spravuje organizace W3C (World Wide Web Consortium). K prohlížení stránek (tedy používání GUI specifikovaného pomocí HTML) poslouží jakýkoliv prohlížeč implementující tyto standardy (není tedy předepsaný žádný konkrétní výrobce, ani softwarový produkt). Toto však v praxi představuje problém, neboť prohlížeče různých výrobců interpretují normy různým způsobem.

Ačkoliv tento problém způsoboval znatelné potíže tvůrcům webu hlavně v minulosti (např. chyba interpretace box-modelu ve starých verzích prohlížeče Internet Explorer [21]), dodnes se můžeme setkat s drobnými rozdíly interpretace webových standardů mezi majoritními prohlížeči [10].

Řešení pomocí čistého HTML z principu postrádá možnost ovlivňovat chování GUI na klientském počítači v reálném čase a v přímé interakci s uživatelem. Tento nedostatek je řešen použitím skriptovacích jazyků (např. JavaScript), ve kterých je možné implementovat dynamické chování GUI na klientském počítači. Kód ve skriptovacím jazyce se přenáší společně s HTML kódem stránky, podléhá tedy stejným pravidlům, co se týče určeného okamžiku přenosu dat a kódu.



Obrázek 3.1: Typické použití HTML na Webu

3.2 X Window System

Tento systém grafického rozhraní je nasazen v unixových systémech jako výchozí grafické prostředí. První verze vznikla na Massachusetts Institute of Technology v roce 1984. Projekt je nyní spravován organizací X.Org Foundation [20].

Přestože se jedná o síťovou aplikaci typu klient/server, je široce využíván i na izolovaných pracovních stanicích, kde vedle sebe běží klientské i serverové procesy a komunikují spolu v duchu klient/server architektury.

Rozdělení rolí klienta a serveru je u X Window poněkud nekonvenční. Stroj, který chápeme jako aplikační server — běží na něm software, jehož služeb chce vzdálený uživatel využít — je v prostředí X Window klientem. Počítač vzdáleného uživatele je označován jako X server — poskytuje X klientovi služby displeje a vstupních zařízení.

Start aplikace na aplikačním serveru tedy obnáší síťové připojení této aplikace (v roli X klienta) na počítač vzdáleného uživatele (v roli X serveru).

Protokol, kterým spolu klient a server komunikují (X protokol), definuje několik elementárních prvků GUI (např. okna, pixmapy, fonty, události) [19].

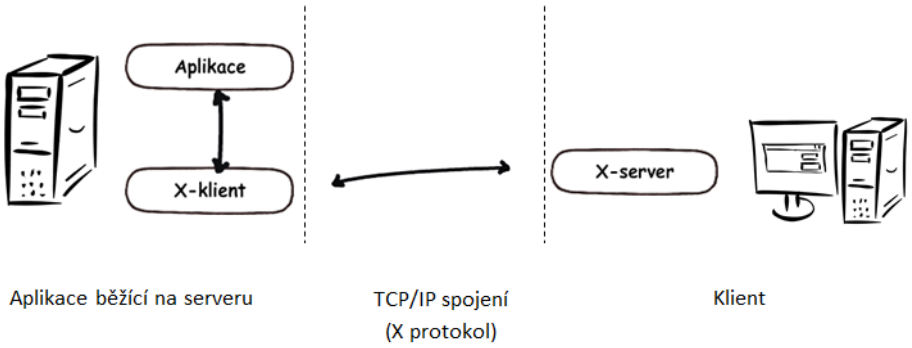
X server se stará o nízkoúrovňové úkony, jako jsou vykreslení oken a pixmap a přeposílání událostí klientovi. Obsluhu událostí a vykreslování na vyšší úrovni už řeší samotná aplikace (X klient). Dochází tedy k objemným přenosům dat mezi klientem a serverem.

Šířka pásma a latence mohou při použití systému přes počítačovou síť způsobovat výrazné snížení použitelnosti některých režimů práce se softwarem. Šířka pásma je klíčovým faktorem při sledování videa a při přenášení textur pro 3D zobrazení. Latence může způsobovat problémy v interaktivních aplikacích (zejména ve hrách), při vyšších latencích se však může např. i základní manipulace s příkazovými nabídkami aplikace stát obtížnou.

X Window System byl vyvinut před rozšířením přehrávání videa a animací na počítačových displejích a nejlépe pracuje se statickou grafikou a textem. Síťový provoz neposkytuje dostatečnou šířku pásma pro přehrávání animace založené na bitmapách o vysokém rozlišení. Například datový proud nekomprimovaného videa o rozlišení 640x480 při barevné hloubce 24 bitů a 30ti snímcích za sekundu spotřebuje 221 184 000 bitů/s (221 Mbit/s), což dalece překonává možnosti 10 Mbit Ethernetu, který byl dostupný v době, kdy rozhraní X bylo novinkou. Stroj s typickým displejem dnešní doby o rozlišení 1920x1200 by potřeboval konektivitu s šířkou pásma 1.65 Gbit/s pro přehrávání videa přes celou obrazovku [18].

Vývojáři tvořící aplikace pro X Window System používají ve svých programech knihovnu XLib, která plní roli X klienta a stará se o vykreslování a obsluhu rozhraní na X serveru. Programování pro XLib je však poměrně nízkoúrovňové a těžkopádné, existuje proto celá řada nadstavbových knihoven (grafických toolkitů), které slouží k zapouzdření XLibu a ke snadné tvorbě pokročilého rozhraní. Příkladem mohou být toolkity Motif [4], GTK+ [2], Qt [6].

Kromě verze X serveru pro unixové systémy (včetně Mac OS X) existuje implementace X serveru např. pro Microsoft Windows (xming [8]). Samotný protokol X Window je nezávislý na platformě, je proto možné vzdáleně přistupovat k aplikaci X Window z jiného operačního systému, než který běží na straně X klienta (vzdálené aplikace).



Obrázek 3.2: X Window system

3.3 OpenXava

OpenXava je framework jazyka Java pro model–based generování uživatelského rozhraní webových aplikací.

Základní myšlenkou je zanedbání specifikace grafické reprezentace dat aplikace. Vývojář pouze pomocí anotací[1] třídy modelující zobrazovanou entitu přidá upřesňující pokyny

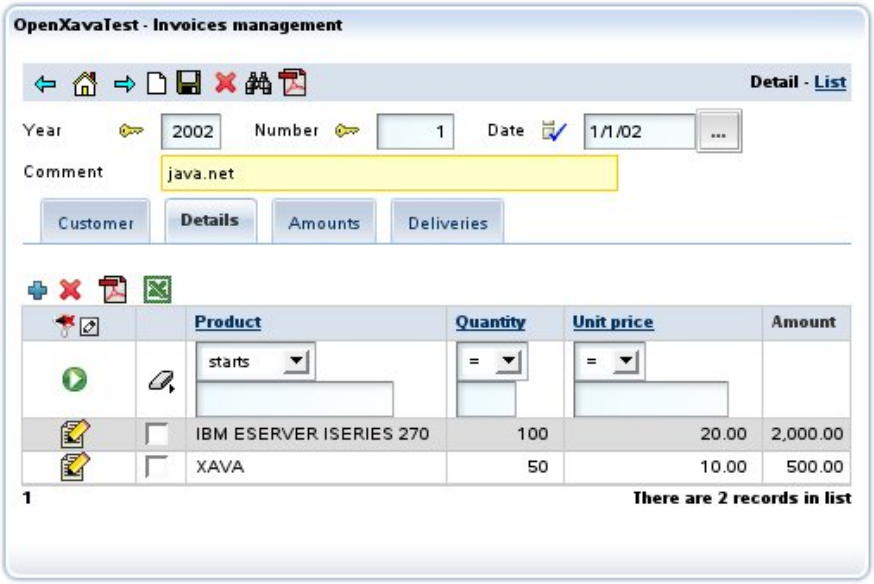
[1] Jedná se o speciální konstrukce jazyka Java nesoucí metadata, které lze vkládat do zdrojového kódu. Anotace lze přidávat k třídám, metodám, proměnným, parametrům a balíčkům. Na rozdíl od běžných anotací dokumentačního charakteru v jiných prostředích, jsou anotace v jazyce Java přístupné za běhu programu pomocí reflexivního programování.[17]

týkající se žádané podoby reprezentace daných atributů. Tyto anotace mohou být velmi elementární a určovat jen konkrétní druh zobrazeného elementu uživatelského rozhraní. Pokud však vývojář není spokojen s automaticky vygenerovaným výsledkem, má možnost doladit podobu rozhraní pomocí přídavných parametrů anotací.

Kromě svých vlastních anotací dokáže framework interpretovat i anotace rozhraní Java Persistence API a sledovat tak vzájemné závislosti některých entit a atributů.

Obrázek 3.3 ukazuje formulář vygenerovaný pomocí OpenXava. Jedná se o formulář pro editaci faktury. Zdrojový kód, z kterého byl formulář vygenerován, lze nalézt v příloze B – anotace jsou ve zdrojovém kódu zapsány jako identifikátory začínající znakem "@". Příklad je převzat z [14].

Domovská stránka projektu viz [5].



Obrázek 3.3: Ukázkový formulář vygenerovaný pomocí OpenXava

3.4 Charakterizace dat

K rozlišení dat vztažených k uživatelskému rozhraní a popisu jejich vazeb byl zaveden proces charakterizace dat. Tato sekce čerpá z práce [22], která zavádí detailní taxonomii pro charakterizaci dat:

Pro charakterizaci dat je použito následujících šest dimenzí: typ *(data type)*, doména (*data domain*), atributy (*data attributes*), relace (*data relations*), role (*data role*) a smysl (*data sense*).

Typ určuje rozdělitelnost informace na menší celky. Rozlišujeme typy atomické (*atomic*) a složené (*composite*). Objekty složeného typu se skládají z jiných objektů složených, nebo atomických typů. Mohou být dále klasifikovány jako množiny, nebo struktury v závislosti na vzájemných vztazích komponent. Znalost strukturních charakteristik složených objektů je klíčová pro konstrukci jejich grafické reprezentace.

Doména slouží ke kategorizaci informace v sémantické taxonomii. Jsou definovány základní tři domény:

- Entita (*entity*)

  Entita určuje nezávisle existující objekt s jedinečnou identitou. Může korespondovat s fyzickým objektem reálného světa, nebo může být jako fyzický objekt zobrazena.

  Příklad: auto

- Koncept (*concept*)

  Jedná se o abstraktní pojem, který buď existuje nezávisle, nebo musí být vztažen k jiným objektům. Koncepty nejsou fyzické objekty.

  Příklad: stáří

- Míra (*measurement*)

  Míra je číselný, nebo nečíselný údaj volitelně doprovázený měrnou jednotkou. Nemůže existovat samostatně, ale musí se vztahovat k nějakému konceptu.

  Příklady: 10 let; středně starý

Atributy jsou specifické pro konkrétní typy dat. Některé atributy jsou však společné pro všechny typy:

- Forma (*form*)

  Týká se tvaru objektu v reálném světě. Objektům mající nějaký tvar se připisuje forma tvaru (*shaped*), objekty, jejichž tvar není jednoznačný, nebo je příliš abstraktní, mají formu beztvarou (*shapeless*). Všechny konceptuální objekty pak nemají formu žádnou (*none*).

- Materiál (*material*)

  Popisuje potenciálně komplexní proměnné, které určují, jakým způsobem objekt reaguje na světlo. Může se jednat o jednoduchý barevný popis pomocí složek základních barev, nebo o sofistikovaný osvětlovací model.

- Umístění (*location*)

  Může popisovat prostorové (*spatial*), ale i časové (*temporal*) umístění datového objektu. K popisu používáme buď kvantitativní souřadnice, (např. Kartézská soustava souřadnic), nebo kvalitativní popis („blízko“, „před“).

- Pomíjivost (*transience*)

  Rozlišuje informace dynamické (*dynamic*), které se mění v čase, a statické (*static*), které zůstávají neměnné.

- Důležitost (*importance*)

  Různá důležitost informací nám umožňuje různými technikami vizuálně sloučit, nebo naopak oddělit jednotlivé prvky informace. Důležitá část informace může být zvýrazněna za účelem upoutání uživatelovy pozornosti.

Další dimenzí charakterizace dat jsou relace. Jsou definovány relace funkční závislosti (*functional dependency*), obsažnosti (*constituency*), atributu (*attribute*) a výčtu (*enumeration*).

Dimenze role charakterizuje funkční úlohu, kterou část informace hraje v kontextu vizuální prezentace. Definované role jsou kategorizace (*categorization*), seskupování (*clustering*), identifikace (*identification*), rozlišení (*distinguishing*), porovnání (*comparison*), asociace (*association*), ohodnocení (*ranking*), korelace (*correlation*) a distribuce (*distribution*).

Poslední dimenzí je smysl, který určuje preferovaný způsob vizuální prezentace dat. Může být ovládaný uživatelem, nastavený předem, nebo může být odvozen od ostatních charakteristik. Může nabývat následujících hodnot: text (*label*), seznam (*list*), graf (*plot*), symbol (*symbol*) a portrét (*portrait*).

3.5 Charakterizace kódu

Práce [11] pro účely automatického generování uživatelského rozhraní kromě využití zmíněné taxonomie pro charakterizaci dat zavádí charakterizaci programového kódu. Obsah této sekce čerpá z této práce.

Využitím sémantických informací získaných charakterizací kódu můžeme odvodit mnoho rysů uživatelského rozhraní. V kombinaci s charakterizací dat získáváme kromě popisu dat a jejich vztahů popis funkcí a akcí, které je možné nad daty provádět.

Charakterizace kódu je realizována pomocí reflexivního programování, které je podporováno soudobými programovacími jazyky (např. Java a .NET framework). Reflexivní programování nám umožňuje za běhu programu získávat informace o programových součástech (třídách, metodách, atributech). Proces je konkrétně předveden na příkladech v jazyce C# s využitím jazykové konstrukce atributů, která slouží k zápisu metadat.

Zavedená taxonomie zavádí pět kategorií anotačních značek, které popisují zvláštní aspekty metod a funkcí s ohledem na uživatelské rozhraní.

Atributy metod (*method attributes*) vyjadřují různé informace o metodách, které daný objekt implementuje. Ačkoliv se může zdát, že metody jsou obecně příliš různorodé na to, abychom je dokázali popsat předem definovanými atributy taxonomie, mají dost společných atributů důležitých pro správné vytvoření uživatelského rozhraní. Jsou definované následující atributy metod:

- Jméno (*name*)

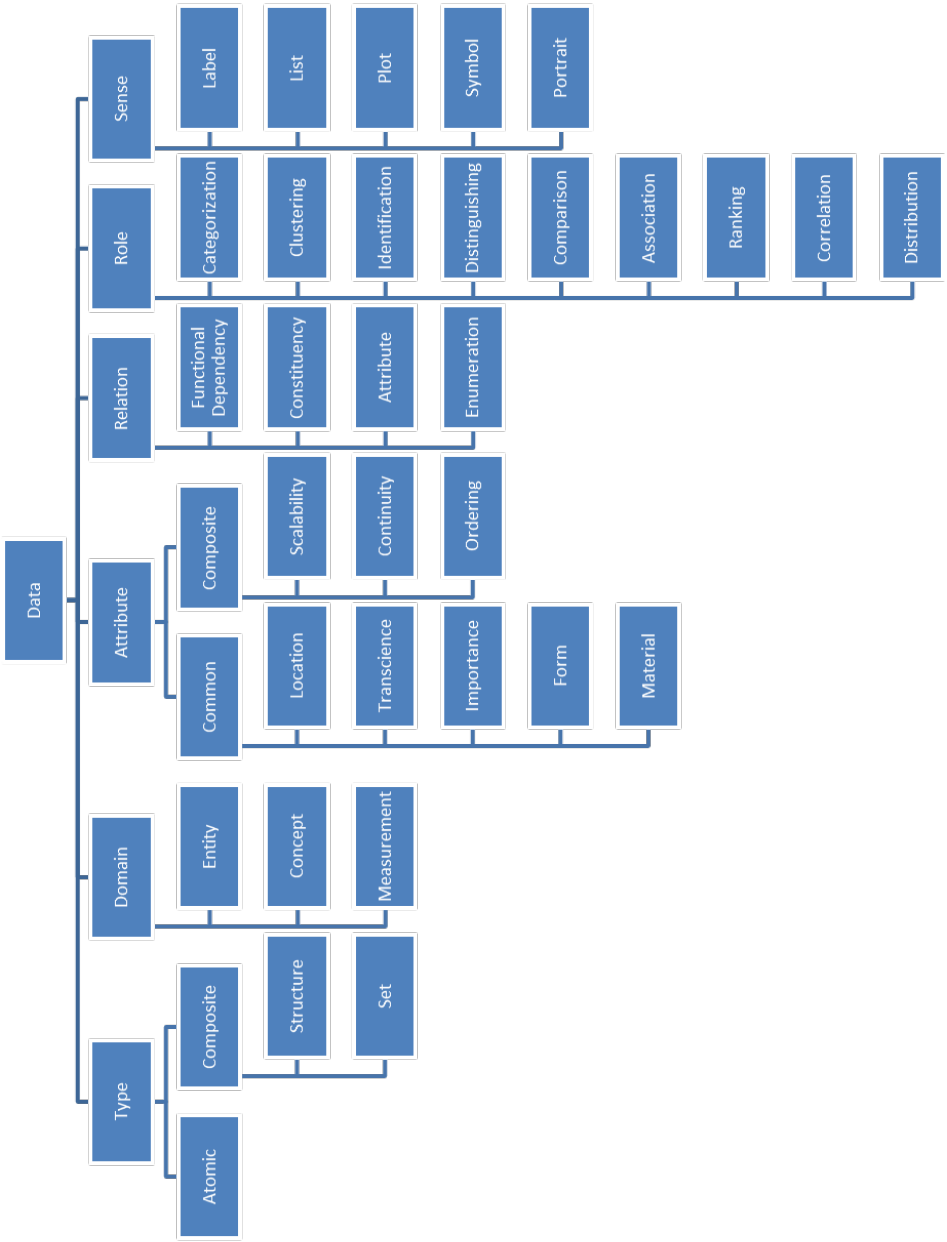
  Implementuje-li objekt metodu, která se má podílet na funkci uživatelského rozhraní, měl by u ní být definován tento základní atribut. Tímto způsobem pojmenováváme metodu tak, jak bude prezentována uživateli. Metody, které spadají do některé předdefinované kategorie, atribut mít nemusí, neboť jejich pojmenování je již definováno pomocí této kategorie.

- Popis (*description*)

  Tímto volitelným atributem můžeme uživateli poskytnout detailnější informace o funkci dané metody, například formou kontextové nápovědy.

- Důležitost (*importance*)

  Pomocí tohoto atributu můžeme rozlišovat metody podle důležitosti. Častěji prováděné operace by měly být snadněji dostupné v rámci efektivnosti ovládání. Tyto operace mohou být poté umisťovány do snadněji přístupných nabídek a panelů nástrojů. Není snadné jednoznačně uspořádat metody podle důležitosti, protože četnost jejich použití závisí na řadě faktorů. Je možné aktualizovat tento atribut za běhu a reflektovat tím skutečnou četnost použití konkrétním uživatelem.



Obrázek 3.4: Charakterizace dat

- Reprezentace (*representation*)

Tímto atributem definujeme symbol, který bude reprezentovat danou metodu. Může se jednat o ikonu, nebo obrázek. U metod spadajících do jedné z předdefinovaných kategorií je většinou tento atribut definován samotnou kategorií.

- Závislost (*dependency*)

  Metody mohou záviset na některých předpokladech, na jejichž splnění závisí provedení programového kódu. Zohlednění tohoto faktu většinou kontroluje samotný vývojář, při automatickém generování rozhraní však musí vrstva uživatelského rozhraní vědět, kdy je možné danou akci provést a kdy ne. Atribut závislosti tedy určuje podmínky, které musí být splněny, aby daná metoda byla zpřístupněna v uživatelském rozhraní. Jeho použití je velice důležité, neboť bez kontroly předpokladů by uživatelé měli přístup k nedostupným příkazům. Atribut může být použít vícekrát, aby bylo možné reflektovat více pravidel.

- Modální metoda (*modal*)

  Některé operace v rámci zachování správného sledu akcí vyžadují omezení uživatelské interakce výlučně na danou operaci po dobu jejího provádění. K označení metody implementující takovou operaci slouží tento atribut. Může být realizován například modálními okny, které omezují přístup do ostatních oken aplikace.

- Dialog (*dialog*)

  Slouží k realizaci upozornění a potvrzovacích dotazů předcházejících a ovlivňujících spouštění dané metody.

Pro zavolání metody je většinou třeba určit hodnoty jejích parametrů. Je-li metoda předmětem generování uživatelského rozhraní, měly by i její parametry být součástí uživatelského rozhraní. Aby bylo možné určit podobu tohoto rozhraní musí být parametry také předmětem charakterizace. Jsou tedy zavedeny atributy parametrů (*parameter attributes*):

- Jméno (*name*)

  Slouží k podobnému účelu jako shodný atribut metody. Není bezpodmínečně nutné jej použít, je ale velmi důležitý pro rozhraní, která nejsou založena na vizuálním výstupu.

- Popis (*description*)

  Má stejný význam jako shodný atribut metody.

- Výchozí hodnota (*default value*)

  Určuje předvyplněnou hodnotu parametru.

- Maximální a minimální hodnota (*maximum / minimum value*)

  Pro číselné parametry lze stanovit rozsah, v jakém se smí pohybovat uživatelský vstup. Uživatelské rozhraní poté automaticky kontroluje zadanou hodnotu.

Další kategorií je smysl (*sense*), kterým rozlišujeme způsob, jakým uživatel vyvolá provedení metody:

- Příkaz (*command*)

  Uživatel zvolí příkaz a je uživatelským rozhraním dotázán na hodnoty parametrů. Metoda je poté spuštěna s určenými parametry. Tento přístup je výchozí.

- Nástroj (*tool*)

  Metoda je volána opakovaně, dokud je určitý příkaz v daném kontextu vybrán jako aktivní.

- Výchozí (*default*)

  U některých objektů definujeme také výchozí metodu, která u běžných vizuálních rozhraní může odpovídat například dvojitému kliknutí myší.

Součástí taxonomie je struktura často používaných kategorií (*category*) operací z běžných aplikací. Metoda označená takovou kategorií automaticky získává rozhraní, na které je uživatel zvyklý a které je pro danou operaci efektivní. Příkladem můžou být operace ovládání přehrávání multimediálních dat.

Metoda je v kódu označena příslušnou kategorií a také konkrétní operací v této kategorii, kterou metoda implementuje. Definované kategorie jsou následující:

- Kolekce (*collection*)

  Zahrnuje operace pracující nad daty uspořádanými v kolekci – přidávání a odebírání prvků, výběr.

- Úložiště (*storage*)

  Zahrnuje operace nad obecným úložištěm dat – vytváření, otevírání a ukládání souborů.

- Navigace (*navigation*)

  Zahrnuje operace pro navigaci v kolekci dat – přechod na první, poslední, další a předchozí prvek.

- Přehrávač (*player*)

  Zahrnuje operace specifické pro přehrávání záznamů – spuštění, zastavení, pozastavení

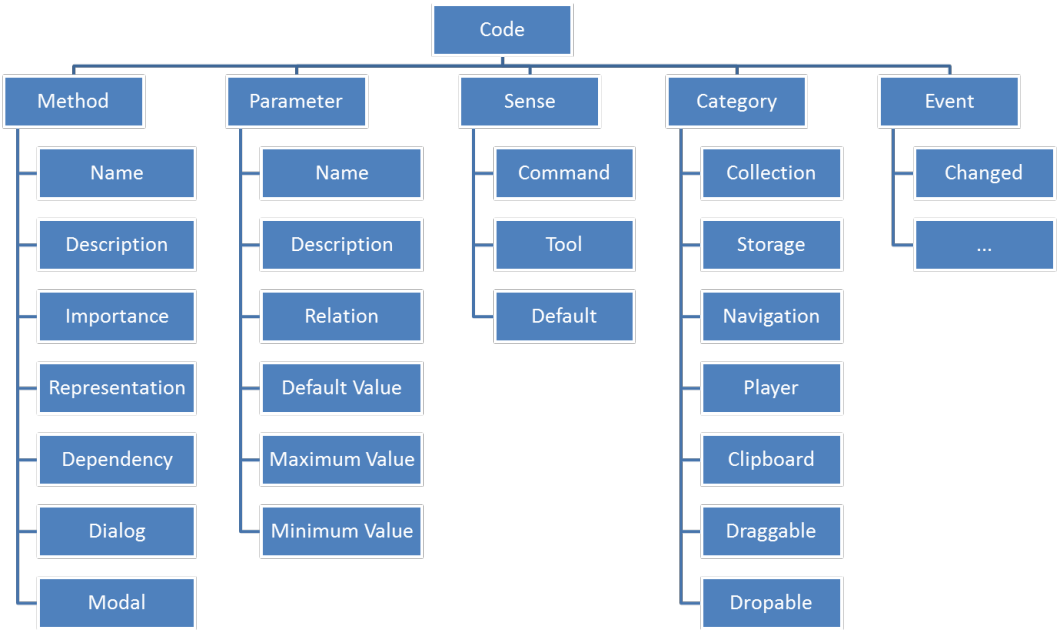
- Schránka (*clipboard*)

  Zahrnuje operace pro práci se schránkou – kopírování, vložení

- Operace „táhni a pusť“ (*draggable*, *droppable*)

  Zahrnuje operace táhnutí objektu, a jeho upuštění nad jiným objektem.

Poslední kategorií v taxonomii jsou události (*event*). Základní a v návrhu jedinou uvažovanou událostí je událost změny (*changed*), která nastává při jakékoliv změně obsahu daného objektu.



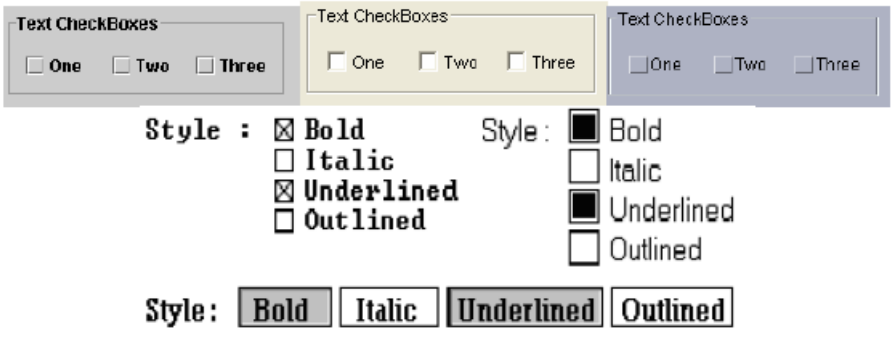
Obrázek 3.5: Charakterizace kódu

Kromě popsané taxonomie je v odkazované práci [11] navržen způsob abstrakce jednotlivých objektů uživatelského rozhraní, který vychází z publikace [16]. Konkrétně se jedná o koncept objektů CIO (*concrete interaction object* – konkrétní objekt interakce) a AIO (*abstract interaction object* – abstraktní objekt interakce).

Jako CIO lze označit jakoukoliv entitu uživatelského rozhraní vnímanou, či ovládanou uživatelem. Může se jednat o určitý ovládací prvek poskytnutý použitou knihovnou pro implementaci uživatelského rozhraní.

Lze pozorovat značnou podobnost (vizuální i behaviorální) mezi ovládacími prvky v prostředí rozličných systémů a knihoven uživatelských rozhraní (viz obrázek 3.6).

Proto byly zavedeny objekty AIO, které jsou definovány jako abstrakce množiny objektů CIO s ohledem na společné vlastnosti. Musí být definována knihovna objektů AIO, kterým jsou přiřazeny objekty CIO (konkrétní implementace), které mají očekávanou funkcionalitu.



Obrázek 3.6: Ovládací prvky z různých grafických prostředí (Java Swing, Microsoft Windows, OSF/Motif, Garnet) [16]

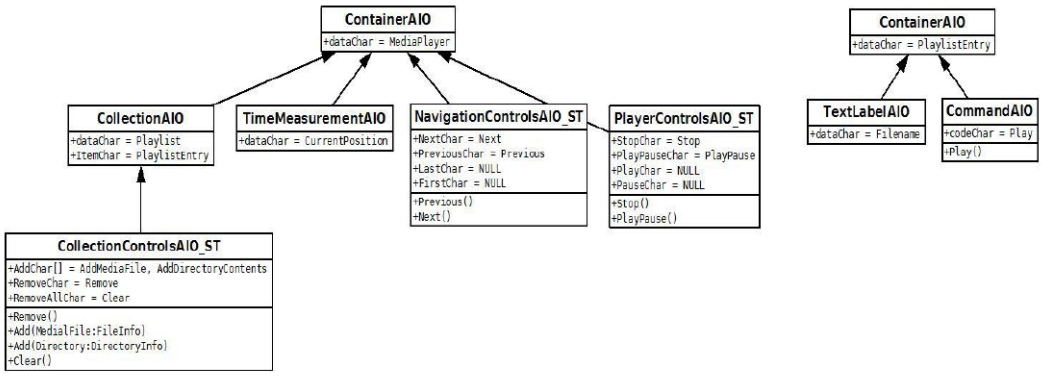
Charakterizovaný kód a data musí být namapovány na odpovídající objekty AIO ne-

soucí informace o komponentách uživatelského rozhraní, které budou použity k interakci s uživatelem. Volba vhodného objektu AIO je podřízena komplexnímu a stabilnímu systému pravidel. V odkazované práci [11] je prezentovaný algoritmus pro převod charakterizovaného kódu a dat (tzv. charakterizačního stromu) na sadu objektů AIO.

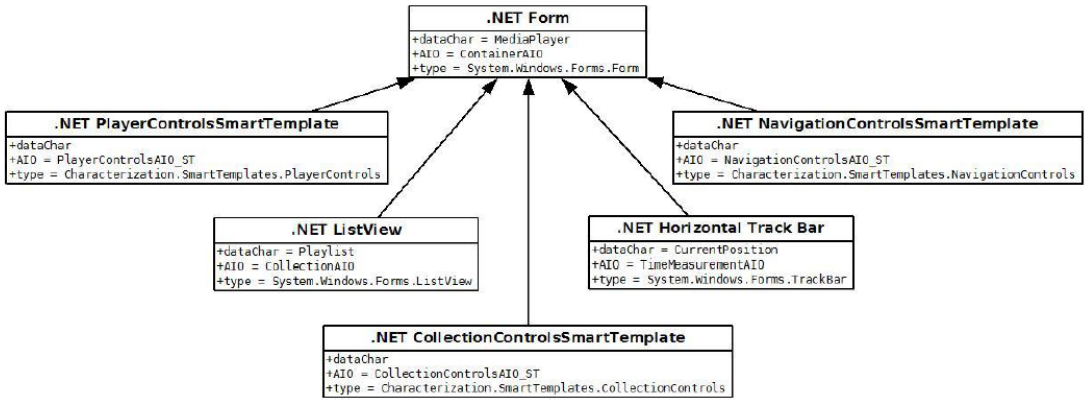
Po získání objektů AIO jsou jim přiřazeny odpovídající objekty CIO, a to s ohledem na použitý systém i na požadovanou podobu rozhraní (koncept počítá např. s možností generování rozhraní pro ovládání hlasem stejným způsobem).

Jednou z ukázkových aplikací konceptu je jednoduchý mediální přehrávač. Pomocí objektů AIO (viz obr. 3.7) a jim přiřazených objektů CIO (viz obr. 3.8) je vytvořené uživatelské rozhraní zobrazené na obrázku 3.9.

Příslušný charakterizovaný kód je uveden v příloze C.



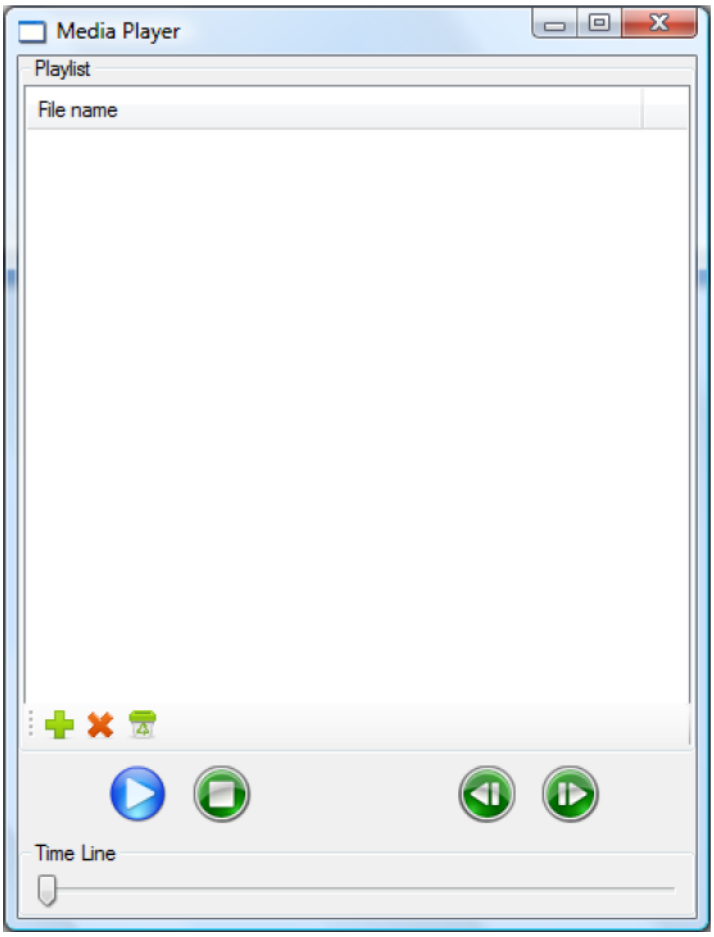
Obrázek 3.7: Objekty AIO ukázkové aplikace [11]



Obrázek 3.8: Objekty CIO ukázkové aplikace [11]

3.6 Stávající řešení společnosti DISK

Společnost DISK Multimedia, s.r.o. se zabývá mimo jiné vývojem aplikací pro zpracování signálů v reálném čase. Příkladem takového produktu může být virtuální modulární mixážní



Obrázek 3.9: Ukázková aplikace (mediální přehrávač) generovaná pomocí charakterizovaného kódu [11]

pult zobrazený na obrázku 3.10. Do jednotlivých zvukových kanálů uživatel vkládá moduly pro úpravu signálu a nastavuje jejich parametry.

Produkty společnosti DISK Multimedia, s.r.o. pro serializaci dat používají formát IXML (Incremental XML), který byl navržen za účelem přenosu hodnot atributů mezi jednotlivými aplikacemi, případně pro ukládání nastavení a potenciálně také ke generování uživatelského rozhraní.

Základem formátu je značkovací jazyk XML se stanoveným typem dokumentu Property List. Typ dokumentu Property List je vyvinut společností Apple a běžně využíván v jejích produktech [3]. Princip formátu spočívá v hierarchickém uspořádání záznamů do XML stromu.

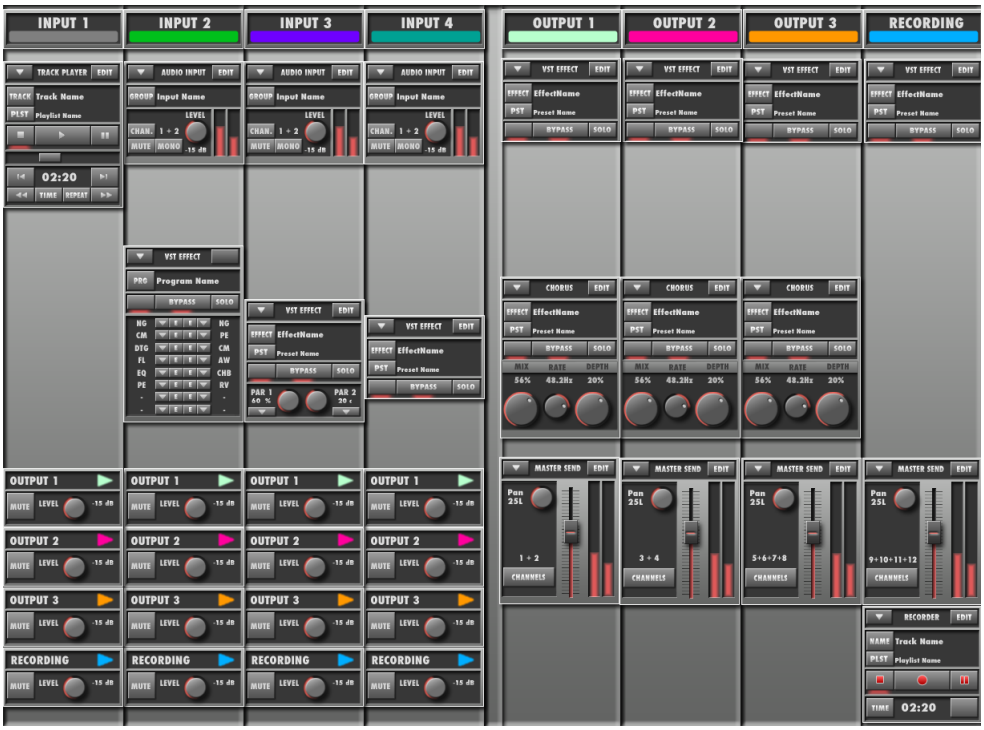
Každý záznam se skládá z klíče (key) a svého obsahu, což může být hodnota (textový řetězec, celé číslo, reálné číslo), pole těchto hodnot, specifická blíže neurčená data, nebo seznam vnořených záznamů.

Zvláštnost tohoto formátu spočívá v nezvyklém uspořádáním značek v XML souboru. Intuitivně bychom reprezentovali záznam pomocí zvláštního elementu, kterému bychom přiřadili klíč a obsah — a to buď pomocí atributů, nebo vnořených elementů. Ve formátu Property List však záznam nemá vlastní element a je vždy reprezentován dvojicí sousedních elementů (klíč a obsah). Chceme-li tedy získat obsah záznamu pod určitým klíčem, musíme

přečíst obsah elementu následujícího po elementu s nalezeným klíčem.

Příklad dat uložených v IXML znázorňuje kód A.1 v příloze A.

Inkrementální funkčnost IXML je realizována speciálním záznamem — zprávou o změně stavu. Každá zpráva o změně stavu se skládá z pěti částí: adresa modulu, adresa klíče, typ zprávy, příkaz a datový blok, který se na dané umístění má uložit. Principiální strukturu takovéto zprávy znázorňuje kód A.2 v příloze A.



Obrázek 3.10: Software společnosti DISK: Modulární mixážní pult

Kapitola 4

Zhodnocení stávajících možností

Bylo prezentováno několik současně dostupných možností pro generování GUI a vzdálené GUI. Při návrhu nového produktu stojí určitě za zvážení použití existujících technologií, jako jsou vysokoúrovňové model–based systémy, jako je například zmíněná OpenXava, nebo systémy pro popis GUI, jako je HTML.

Konkrétně HTML je velmi dobrý adept na implementaci rozhraní jak vzdálených počítačových aplikací, tak i vestavěných systémů. Webové prohlížeče jsou masově rozšířené a používané v osobních počítačích i mobilních zařízeních, není třeba programovat žádné klientské aplikace. Ovládání vestavěného systému jako webové služby může pro uživatele znamenat snadnou orientaci v dobře známém prostředí. Například použití webového rozhraní v administračních nástrojích aktivních síťových prvků je osvědčená praxe.

Implementace webového serveru do vestavěného systému však může představovat příliš velkou režii, navíc paradigma webových služeb nemusí plně vyhovovat účelu uživatelského rozhraní. Pro systémy pracující v reálném čase se nejedná o vhodné řešení.

Systémy vzdáleného GUI založené na vytváření rozhraní na straně aplikace a jeho přenos počítačovou sítí na úrovni primitiv (např. X Window) představují velký objem přenášených dat a nižší svižnost, nejsou tedy rovněž vhodné pro systémy pracující v reálném čase.

Žádný ze zkoumaných systémů nebral zvláštní ohled na druh zobrazujícího zařízení (velikost displeje) a kontext užití aplikace.

Žádný rovněž neumožňoval uživateli přizpůsobit si ve větší míře rozhraní podle svých představ.

4.1 Zhodnocení stávajícího formátu

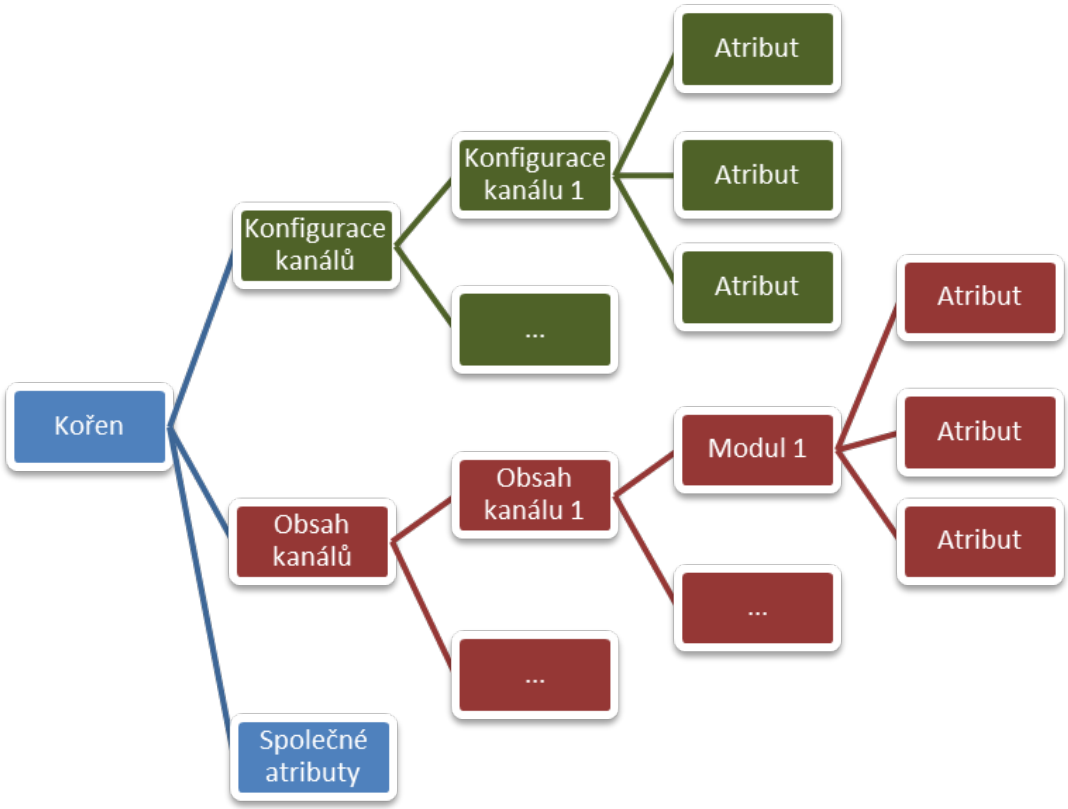
Vzhledem k úmyslu aplikovat systém generování rozhraní na produkty společnosti DISK Multimedia se nabízí možnost využít přímo interní datový formát zmíněných produktů k specifikaci datových modelů pro generování rozhraní.

Přestože formát IXML byl navržen za účelem abstraktního popisu atributů v aplikaci a jejich změn, což vystihuje princip zmíněných datových modelů, je jeho struktura podřízena specifikům aplikace a neobsahuje kompletní informace nutné k univerzálnímu automatickému generování rozhraní.

Jedním z požadavků, které klademe na model, je hierarchické uspořádání atributů do skupin tak, abychom podle nich mohli vizuálně seskupovat ovládací prvky logicky spadající do jednoho celku. Formát IXML sice hierarchickou strukturu má, k seskupování atributů ji však využívá pouze v omezené míře, a navíc spoléhá na apriorní znalosti struktury aplikace.

Příkladem může být soubor obsahující popis uspořádání modulů pro zpracování signálů ve virtuální mixážní konzoli a hodnoty atributů. Kód A.3 v příloze A je výtahem z tohoto souboru, celý soubor lze nalézt na přiloženém CD umís `models/ixmlmix.xml`.

Pro snazší pochopení je jeho zjednodušená struktura znázorněna na obrázku 4.1. Soubor principiálně odpovídá rozhraní na obrázku 3.10, liší se v konkrétních modulech a jejich nastaveních.

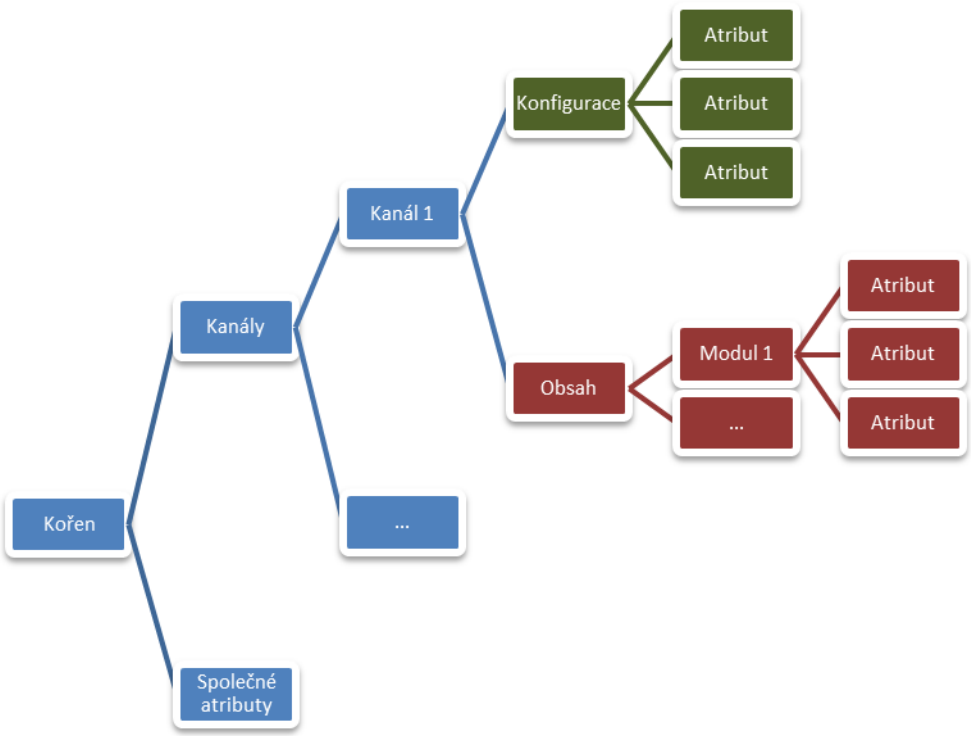


Obrázek 4.1: Struktura IXML popisu mixážní konzole

Z obrázku můžeme pozorovat některé atributy logicky patřící do společného celku (např. veškeré atributy týkající se audio kanálu 1) rozmístěné do dvou různých částí hierarchie — části pro nastavení základních atributů kanálu a části pro rozmístění modulů a nastavení jejich atributů. Toto není pro naše účely žádoucí, neboť potřebujeme všechny atributy týkající se jednoho audio kanálu sloučit do jedné skupiny (a dále je případně slučovat do skupin podle modulů). Vhodnější uspořádání znázorňuje obrázek 4.2.

Dalším problémem použitého formátu jeho nedostatečná obecnost. Formát sice zpočátku vycházel z univerzálně navrhnutého formátu Property List, způsob strukturování dat v něm

je však podřízen samotné aplikaci. Příkladem může být vzájemné provázání kolekcí `Chain Configs`, a `Chain Contents` v příkladu A.3 jejichž obsahy spolu vzájemně souvisí, tento vztah je však implicitně předpokládán a není ze souboru jinak patrný.



Obrázek 4.2: Vhodnější struktura modelu mixážní konzole

K účelům univerzálního automatického generátoru rozhraní je tedy vhodnější namísto přímého využití stávajícího formátu navrhnout formát vlastní. Tento návrh popisuje kapitola 5.2.

Kapitola 5

Návrh platformy pro generování distribuovaného GUI

5.1 Požadavky na systém

Na systém pro automatické generovaní rozhraní přes počítačovou síť klademe následující požadavky:

- Tenký klient

  Pracovní stanice budou vybaveny unifikovaným klientem, který bude vykreslovat a obsluhovat GUI. Klient bude nezávislý na konkrétní aplikaci a bude sestavovat rozhraní podle dané specifikace, kterou mu dodá server.

- Multiplatformnost

  Aplikace musí být ovladatelné z prostředí všech majoritních operačních systémů (Microsoft Windows, Mac OS X, Unix/Linux)

- Asynchronní aktualizace rozhraní

  Klient bude moci kdykoliv odeslat serveru události vzniklé v rozhraní. Stejně tak i server bude moci v kterýkoliv okamžik aktualizovat objekty rozhraní podle nových skutečností (např. při manipulaci s aplikací jinými uživateli).

- Přizpůsobitelnost

  Uživatel by měl mít možnost přizpůsobit si vygenerované rozhraní (přemístit nebo skrýt některé prvky).

- Transparentnost

  Uživatel by neměl prakticky pocítit nasazení systému hlavně z pohledu nutnosti složitější konfigurace, nebo podobných úkonů.

5.2 Model rozhraní

Rozhraní bude generované na základě zjednodušeného modelu. Tento model je složen z atributů, jejichž smyslem je popis jednotlivých proměnných aplikace zamýšlených k reprezentaci pomocí vygenerovaného rozhraní.

Atributy jsou v modelu hierarchicky uspořádány do skupin podle vzájemného logického vztahu. Ve výsledku se to projeví rozmístěním odpovídajících ovládacích prvků do vnořených kontejnerů.

Atributy mohou být parametrizovány dalšími vlastnostmi, jejichž hodnoty ovlivní volbu příslušného ovládacího prvku a jeho nastavení:

- Minimální a maximální hodnota

  Omezuje přípustné hodnoty číselného atributu. Vygenerované rozhraní zajistí dodržení rozsahu.

- Krok

  Je–li u číselného atributu zadán krok, bude číselný vstup omezen pouze na diskrétní hodnoty v daném rozsahu. To prakticky umožňuje využít ovládacích prvků posuvník, nebo přírůstkové pole.

- Jen pro čtení

  Atribut s touto vlastností bude v uživatelském rozhraní prezentován jako pevná, neměnitelná hodnota.

- Datový typ

  Výchozím datovým typem je desetinné číslo, lze ale specifikovat celočíselný, nebo řetězcový datový typ.

- Identifikátor

  Atribut musí mít jednoznačný identifikátor, aby při obsluze uživatelských akcí aplikace mohla správně interpretovat změnu v modelu.

- Popisek

  Určuje textový popis atributu zobrazitelný uživateli ve vygenerovaném rozhraní

- Jednotka

  U číselných atributů lze určit jednotku veličiny, kterou atribut představuje. Tato jednotka se pak zobrazí na odpovídajícím místě za vstupním ovládacím prvkem.

- Znepřístupnění

  Atribut můžeme dočasně uzamknout proti zápisu. Rozdílem oproti vlastnosti „Pouze pro čtení“ je dočasná povaha tohoto příznaku. Typ vygenerovaného ovládacího prvku bude umožňovat úpravu hodnoty, tato možnost však bude dočasně znepřístupněna.

- Viditelnost

  Ovládací prvek příslušného atributu můžeme schovávat a zobrazovat podle potřeby.

Kontejner s atributy (nebo vnořenými kontejnery) má vždy buď vertikální, nebo horizontální rozložení obsažených prvků. Výchozí rozložení kontejneru je vždy komplementární k rozložení nadřazeného kontejneru – tedy kontejnery vnořené do kontejneru s vertikálním rozložením mají rozložení horizontální a naopak. Toto výchozí chování lze dodatečnou

anotací atributu v modelu změnit, a to buď explicitní specifikací druhu rozložení, nebo vynucením zdědění rozložení z nadřazeného kontejneru.

Další složkou modelu jsou signály. Nepředstavují žádnou uživatelem nastavitelnou hodnotu, nýbrž pokyn k akci, který může uživatel udělit. Typickým ovládacím prvkem vygenerovaným pro signál je tlačítko. Kliknutím na tlačítko v uživatelském rozhraní uživatel způsobí odeslání zprávy aplikaci, která na tuto skutečnost může zareagovat. O zprávách dále pojednává následující sekce.

5.3 Zprávy

Při inicializaci pohledu je načten model a na jeho základě je vygenerováno rozhraní. Během provozu aplikace ale dochází k událostem vyžadujícím aktualizaci tohoto modelu. Uživatelské akce způsobující změnu zobrazených informací je nutné reflektovat zpět do modelu a skrz něj zpět do logiky aplikace. Na druhou stranu v aplikaci může dojít k vnitřním událostem, v jejichž důsledku dojde k aktualizaci dat, což je opět nutné pomocí aktualizace modelu zohlednit v uživatelském rozhraní.

Při odděleném běhu aplikace a uživatelského rozhraní je potřeba zajistit synchronizaci modelu mezi oběma účastníky. K tomuto účelu slouží aktualizační zprávy mezi uživatelským rozhraním a samotnou aplikací.

Tyto zprávy obsahují následující informace:

- Identifikátor atributu

  Určuje atribut, jehož hodnota se změnila

- Nová hodnota atributu

Aplikace kromě aktualizačních zpráv dále může zasílat uživatelskému rozhraní další druhy zpráv:

- Ukončení aplikace

- Zrušení rozhraní pro daný model

  Tímto požadavkem aplikace signalizuje pokyn k ukončení uživatelského rozhraní odpovídajícího danému modelu. V praktickém smyslu to znamená uzavření okna, aplikace však zůstává stále spuštěna.

- Inicializace rozhraní podle nového modelu

  Tímto způsobem lze vygenerovat nové uživatelské rozhraní podle nového kompletního modelu, který je součástí zprávy. Nové rozhraní může být vytvořeno odděleně např. v novém okně, nebo nahradit stávající rozhraní, které tímto přestane existovat.

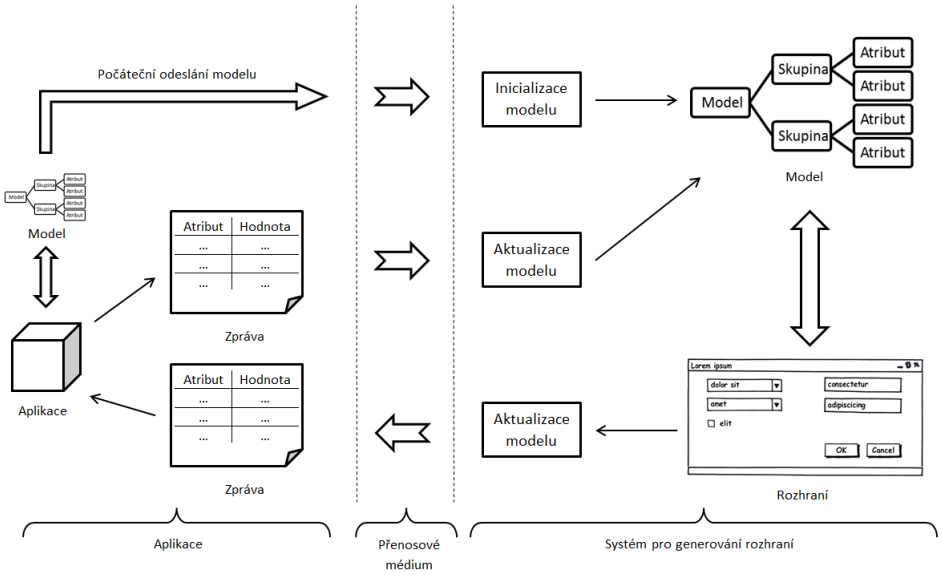
Uživatelské rozhraní pak aplikaci může zasílat tyto druhy zpráv:

- Zrušení rozhraní pro daný model

  Odešle se v případě zavření okna uživatelem.

- Signál

  Odešle se v případě aktivace tlačítka, nebo jiného prvku reprezentujícího signál.



Obrázek 5.1: Architektura systému generování rozhraní

5.4 Reprezentace modelu a zpráv

Modely a zprávy mají formu XML dokumentů. Důvodem je snadná implementovatelnost prototypu systému i případných aplikací s využitím standardních knihoven pro práci s XML a také názornost usnadňující porozumění systému vyplývající ze snadné čitelnosti formátu XML. Dalším důvodem je použití formátu XML pro reprezentaci dat v produktech společnosti DISK Multimedia, s.r.o. a tedy možnost vzájemných transformací datových souborů. V rámci zvýšení efektivity a výkonu při přenosech přes počítačovou síť by bylo vhodné navrhnout úsporný binární formát s principiálně shodnou strukturou. Toto je však mimo rámec této práce a bylo by předmětem dalšího vývoje.

Navrženou strukturu XML reprezentace modelu popisuje tabulka 5.1. Tabulka 5.2 popisuje navrženou strukturu XML reprezentace zpráv.

5.5 Klient a server

Serverovou částí bude aplikace třetí strany schopná generovat rozhraní v navrženém formátu a distribuovat je připojeným klientům.

Klientskou část bude představovat univerzální aplikace sloužící k připojení k serveru, ke stažení modelu pro generování GUI a k jeho vykreslení a obsluze. Pro reálné použití by zde měl být kladen velký důraz na uživatelskou přívětivost a snadnou konfiguraci.

Prototyp implementovaný v rámci práce prostředky k usnadnění použití neobsahuje. V rámci návrhu jsou však uvedeny příklady takovýchto prostředků:

S připojením k serveru (konkrétní aplikaci – službě) bude pomáhat grafický průvodce, jehož posledním krokem bude vytvoření spustitelného odkazu (dle použitého operačního systému, např. zástupce na ploše, symbolický odkaz, položka nabídky, zástupce ve složce s aplikacemi apod.). Daný odkaz potom vždy spustí klientskou aplikaci a provede připojení k vybrané službě.

Z pohledu uživatele se tedy nainstaluje na jejich počítač nová aplikace, která přímo před-

Značka	Atribut	Význam
`model`		Kořenový element
	`id`	Identifikátor modelu umožňující pozdější referenci na model
`group`		Skupina, ve které mohou být vnořeny atributy, nebo další skupiny
	`title`	Popisek skupiny zobrazený v uživatelském rozhraní
	`layout`	Rozložení vygenerovaného kontejneru pro tuto skupinu. Může nabývat následujících hodnot: • `auto` – výchozí rozložení, komplementární k rozložení nadřazeného kontejneru • `inherit` – zdědění rozložení z nadřazeného kontejneru • `horizontal` – horizontální rozložení • `vertical` – vertikální rozložení
`attribute`		
	`id`	Identifikátor atributu
	`title`	Popisek zobrazený v uživatelském rozhraní
	`value`	Hodnota atributu
	`unit`	Jednotka veličiny reprezentované atributem
	`type`	Datový typ atributu • `float` – číslo s plovoucí desetinnou čárkou (výchozí) • `integer` – celé číslo • `string` – řetězec
	`min`	Minimální hodnota číselného atributu
	`max`	Maximální hodnota číselného atributu
	`step`	Krok změny číselného atributu
`signal`		Signál
	`id`	Identifikátor signálu

Tabulka 5.1: Značky použité v XML reprezentaci modelu

stavuje službu, jenž chtějí využívat. Jediný nutný uživatelský úkon související se samotným systémem vzdáleného GUI je tedy spuštění zmíněného průvodce a vytvoření odkazu.

I tento úkon bude možné eliminovat vytvořením speciálního instalačního balíčku (např. poskytovatelem služby), který kromě systému vzdáleného GUI automaticky nainstaluje příslušný odkaz a odstíní uživatele od jakékoliv počáteční konfigurace.

5.6 Komunikace

Po ustanovení spojení server odešle klientovi první model pro vygenerování uživatelského rozhraní.

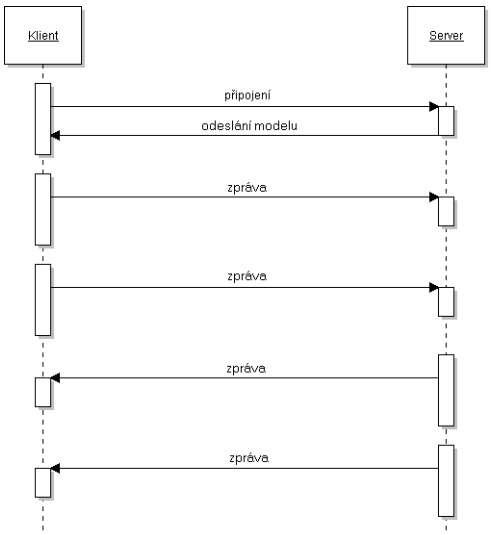
Dále bude komunikace asynchronní – obě strany budou moci kdykoliv odeslat data

Značka	Atribut	Význam
`message`		Kořenový element
`update`		Pokyn k aktualizaci hodnoty nebo vlastnosti atributu
	`attr`	Identifikátor atributu k aktualizaci
	`...`	*(Další atributy jsou shodné s atributy značky* `attribute`*)*
`signal`		Vyvolání signálu
	`id`	Identifikátor signálu
`quit`		Ukončení aplikace
`close`		Zrušení uživatelského rozhraní pro daný model
	`id`	Identifikátor modelu
`load`		Načtení nového modelu V této značce je vnořen XML popis nového modelu
	`replace`	Identifikátor modelu, jehož uživatelské rozhraní bude nahrazeno rozhraním nového modelu

Tabulka 5.2: Značky použité v XML reprezentaci zprávy

popisující aktualizaci rozhraní.

Principiální časový průběh komunikace demonstruje sekvenční diagram na obrázku 5.2.



Obrázek 5.2: Časový průběh komunikace

5.7 Cílová aplikace

Funkčnost navrženého systému bude demonstrována na cílové aplikaci. Bude se jednat o vzdálenou obsluhu produktu společnosti DISK Multimedia, s.r.o. – virtuálního mixážního pultu – pomocí vygenerovaného uživatelského rozhraní.

Pult bude možné obsluhovat více uživateli najednou – tito pak v reálném čase budou moci sledovat změny prováděné ostatními uživateli.

5.8 Navržené úpravy stávajícího formátu

Z důvodů popsaných v kapitole 4.1 byl navržen a implementován nový formát pro reprezentaci modelu a jeho aktualizací (viz. kapitola 5.4).

I stávající formát IXML by však bylo možné použít k účelům automatického generování uživatelského rozhraní po následujících úpravách:

- Úprava struktury souboru tak, aby atributy logicky patřící do společné skupiny, nebyly rozdělené do nezávislých skupin (eliminovat implicitní spojení skupin atributů na základě znalosti struktury aplikace).
- Hierarchické seskupování atributů – seskupování logicky souvisejících atributů v rámci skupiny do dalších podskupin. Přestože z pohledu aplikace není důvod tyto atributy strukturovat, ve výsledku to může pozitivně ovlivnit kvalitu vygenerovaného rozhraní. Příkladem mohou být atributy formující nastavení ekvalizéru. Tato konkrétní situace je dále popsána v realizační části v kapitole 6.4 a ilustruje ji obrázek 6.7.
- Přiřazení jednoznačných identifikátorů všem atributům a skupinám atributů. Současné řešení při aktualizaci modelu pro přístup k danému atributu používá celou cestu v dokumentu, což přináší zbytečně větší výpočetní složitost aktualizace.
- Zavedení značek pro ovlivnění vizuální podoby atributů a skupin v automaticky vygenerovaném rozhraní. Korespondujícím příkladem z navrženého formátu modelu může být atribut skupiny `layout`, který umožňuje ovlivnit rozvržení ovládacích prvků pro atributy v dané skupině (viz kapitola 5.4, tabulka 5.1).

Kapitola 6

Realizace

6.1 Prostředí Qt

K implementaci bylo použito prostředí frameworku Qt z důvodu snadné přenositelnosti a knihovnám umožňujícím realizaci všech aspektů aplikace bez použití dalších externích knihoven.

Prostředí bylo původně vyvinuto společností Trolltech, nyní jej vlastní a dále vyvíjí společnost Nokia.[6]

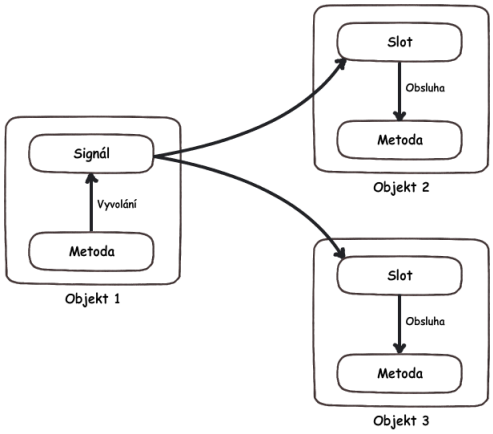
Hlavní součástí frameworku je pokročilý toolkit pro tvorbu aplikací s grafickým uživatelským rozhraním. Dále obsahuje množství knihoven k různým účelům (síťová komunikace, zpracování XML dokumentů, atd.) i vlastní obaly nad standardními strukturami (řetězcový datový typ, kolekce).

K vývoji ve frameworku Qt se standardně používá jazyk C++, pro řadu jiných programovacích jazyků (C#, Java, Python, Ruby, a další) však existují knihovny k provázání s Qt. Tyto knihovny se vzájemně liší (kromě podporovaného programovacího jazyka) kvalitou a mírou jejich pokrytí součástí frameworku.

Specifikem frameworku Qt při použití C++ je systém meta–objektů. Tento systém spočívá ve faktickém rozšíření jazyka C++ o mechanizmus signálů a slotů (viz níže) a další součásti. Před samotným překladem zdrojového souboru, je tento soubor předzpracován dodaným nástrojem `moc` (Meta–Object Compiler). Výstupem tohoto zpracování je standardní zdrojový kód jazyka C++ implementující veškeré aspekty meta–objektu. Tento soubor lze poté přeložit běžným překladačem. Postup překladu ilustruje obrázek 6.1.

Mechanizmus signálů a slotů v Qt umožňuje snadno využívat návrhový vzor „pozorovatel“ (observer) bez nutnosti psát podpůrný kód [1]. Každý objekt Qt může definovat signály a sloty. Různé signály lze kdykoliv napojovat na různé sloty různých objektů, stejně tak je možné existující napojení rušit. Ve chvíli, kdy objekt vyvolá svůj signál, jsou vyvolány všechny metody odpovídající slotům připojeným na tento signál. V terminologii návrhového vzoru „pozorovatel“ lze připojení signálu na slot označit za registraci pozorovatele a vyvolání signálu za upozornění pozorovatelů na změnu. Mechanizmus ilustruje obrázek 6.2.

Obrázek 6.1: Překlad zdrojových souborů v Qt



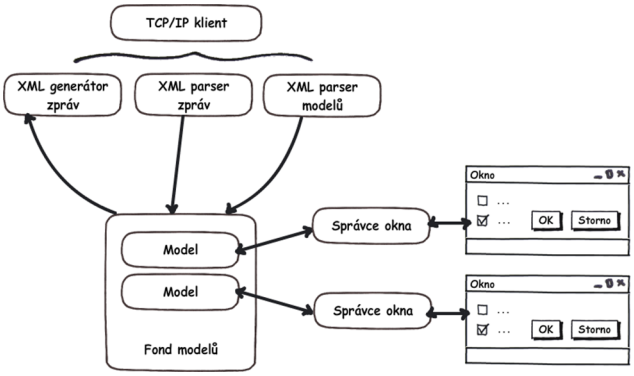
Obrázek 6.2: Mechanizmus signálů a slotů

6.2 Systém pro generování rozhraní

Implementovaný systém se po spuštění dotáže na adresu a TCP port serveru. Se serverem je poté vytvořeno a udržováno asynchronní připojení. V okamžiku přijetí a zpracování modelu XML je vytvořen nový správce okna, který sestaví uživatelské rozhraní na základě přijatého modelu.

Jakékoliv přijaté zprávy (ve smyslu, v jakém jsou popisovány v kapitole 5.3) jsou zpracovány a předány příslušným modelům, které provedou svou aktualizaci a dále pomocí přidruženého správce okna tuto aktualizaci přenesou do uživatelského rozhraní.

Interakci uživatele s prvky uživatelského rozhraní obsluhuje správce okna. Ten aktualizuje příslušný model, který aktualizaci serializuje do zprávy a odešle ji serveru.



Obrázek 6.3: Architektura systému

Fond modelů

Každý serializovaný model přijatý od serveru je zpracován a jemu odpovídající objekt modelu je uložen do fondu modelů. Modely ve fondu jsou adresovatelné svými identifikátory. Případné přijaté zprávy jsou přeposlány odpovídajícímu modelu podle identifikátoru, který je součástí zprávy. Aktualizace modelu ze strany správce okna jsou následně převedeny na zprávy a odeslány na server.

Správce okna

Správce okna na základě modelu sestavuje uživatelské rozhraní z konkrétních komponent frameworku Qt. Vykreslovací logika pomocí implementovaných pravidel každému atributu přiřadí vhodný ovládací prvek, který jej bude reprezentovat. Skupiny atributů jsou realizovány pomocí existujících komponent Qt pro horizontální a vertikální rozložení ovládacích prvků (`QHBoxLayout` a `QVBoxLayout`). Každou uživatelskou akci správce obsluhuje aktualizací modelu.

Síťový protokol

Pro účely demonstrace síťové komunikace s aplikací byl navržen a implementován primitivní aplikační protokol. Protokol je bezstavový, jakákoliv ze stran může kdykoliv odesílat data. Nejsou definované žádné příkazy, účastníci vždy odešlou rovnou XML dokument popisující model či zprávu. Konec přenosu dokumentu je signalizován sekvencí znaků `[CR].[CR]`.

Protokol pro jednoduchost prototypu neřeší případné výskyty zmíněné sekvence v datech, které by způsobily předčasné ukončení přenosu a porušení integrity dokumentu. Tato situace by při standardním použití neměla nastat.

6.3 Server

Pro demonstrativní účely byl implementován jednoduchý server simulující vzdálenou aplikaci. Jeho funkce je následující:

- Každému klientu je ihned po připojení odeslán model načtený ze souboru, jehož název je serveru předán při startu prostřednictvím příkazové řádky
- Případné zprávy přijaté od klientů server nijak nezpracovává, rovnou je odesílá všem ostatním připojeným klientům. Tím je simulováno ovládání aplikace více uživateli najednou.
- Každou sekundu je klientům odeslána pevně definovaná zpráva způsobující aktualizaci atributu s identifikátorem `__time` na řetězec odpovídající aktuálnímu času.
- Jakýkoliv signál zahrnutý v přijaté zprávě vyvolá odeslání alternativního modelu klientovi, který signál vyslal. Tato funkce je aktivní, byl-li parametrem příkazové řádky specifikován název souboru s alternativním modelem.

  Server byl implementován v jazyce Python.

6.4 Ukázková aplikace

Funkčnost implementovaného systému byla předvedena na vygenerování rozhraní pro aplikaci od společnosti DISK Multimedia, s.r.o.. Jedná se o grafický modulární systém pro zpracování signálů. Zmíněný produkt obsahuje mechanismus pro serializaci stavů všech funkčních komponent a jejich inkrementální aktualizaci (viz kapitoly 3.6 a 4.1). Tuto serializovanou strukturu můžeme použít jako základ modelu použitého pro automatické generování grafického rozhraní.

Implementovaný systém má vlastní formát pro serializaci modelů navržený v kapitole 5.4. Existuje několik způsobů k zajištění kompatibility systému s aplikací:

- Podpora pro formát aplikace přímo v systému

Tento způsob není příliš žádoucí, neboť systém by měl být univerzální a nikoliv závislý na konkrétních aplikacích.

V kapitole 4.1 již bylo zmíněno, že tento formát není univerzálně použitelný, protože předpokládá apriorní znalost aplikace. Pro univerzální použitelnost by formát musel být přepracován.

V úvahu by připadalo navrhnout systém zásuvných modulů pro konkrétní produkty, které by disponovaly potřebnými informacemi o struktuře aplikací a jejich datových souborech. Možnosti generování rozhraní by takto byly limitované specifikací aplikace, což však může představovat i výhodu (pevnou specifikací některých komponent rozhraní můžeme získat kvalitnější výstup, než při plně automatickém generování).

- Podpora formátu systému v aplikaci

  Toto řešení by bylo patrně nejvhodnější a systémové. Systém by pak pracoval s aplikacemi přizpůsobenými pro výstup do navrženého formátu pro datové modely.

  V případě reálného využití by zde byl prostor pro další spolupráci se společností a doimplementování výstupu do navrženého formátu do jejích produktů.

- Vzájemná transformace použitých formátů

  Za účelem realizace prototypového příkladu bez nutnosti úpravy aplikace a současně při zachování univerzálnosti systému bylo zvoleno toto kompromisní řešení.

  Mezi systém a vlastní aplikaci se vloží další vrstva, která bude vzájemně převádět data z jednoho formátu do druhého. Bude při tom využívat znalostí o konkrétní aplikaci.

  Vzhledem k použití standardu XML v obou formátech je výhodným řešením použití šablon XSLT pro transformace dokumentů XML.

Transformace datových souborů

Pomocí XSLT transformací můžeme vzájemně převádět XML dokumenty různých formátů. K tomuto účelu musíme sestavit XSLT šablonu, která detailně a formálně popisuje způsob, jakým z jednoho dokumentu získáme dokument druhý. Podrobné informace o XSLT lze nalézt na webu [12].

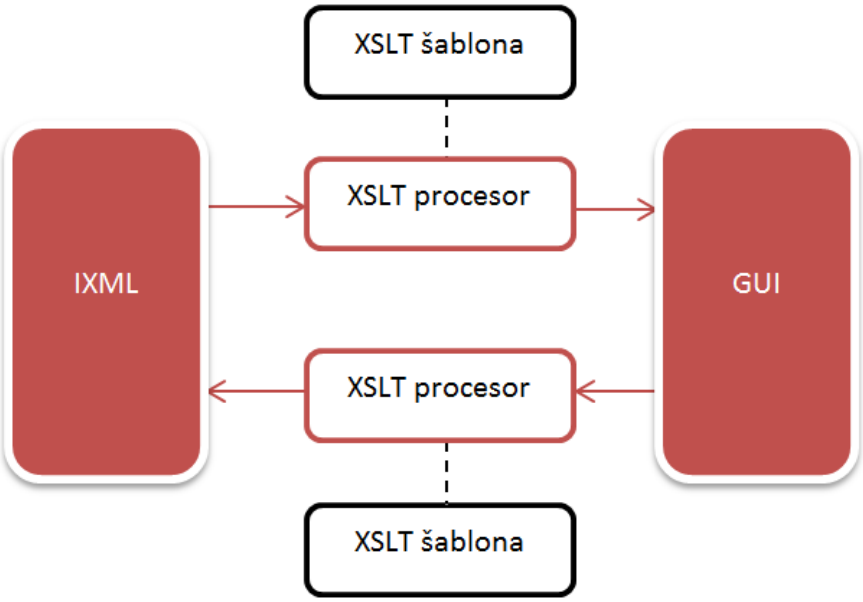
XSLT šablona se skládá z fragmentů cílového dokumentu doplněného o speciální XSLT značky. Pomocí těchto značek můžeme vyhledat a načíst data ve zdrojovém dokumentu, vytvářet cykly a podmínky.

Data jsou ve zdrojovém dokumentu vyhledávána pomocí selektorů XPath umožňujících snadnou navigaci v XML dokumentu.

Kód implementované XSLT šablony pro převod IXML modelu do navrženého formátu je uveden v příloze D.

Samotnou transformaci zajišťuje XSLT procesor, který ze zdrojového dokumentu a transformační šablony vygeneruje dokument cílový. Existuje několik hotových implementací XSLT procesorů, včetně knihoven použitelných v různých programovacích jazycích.

K vývoji a testování byl použit procesor SAXON [7].



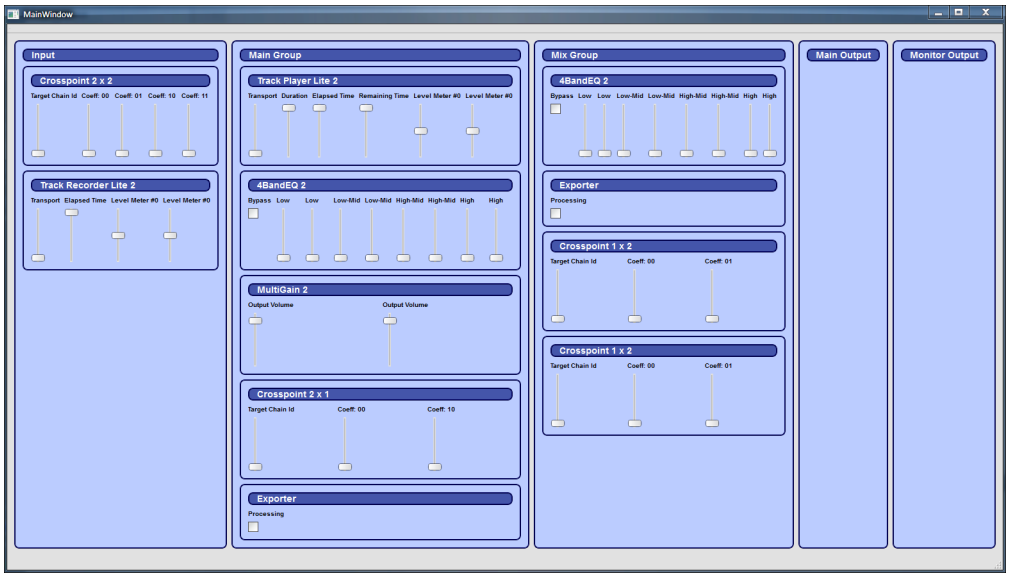
Obrázek 6.4: Transformace datových souborů

Vygenerované rozhraní

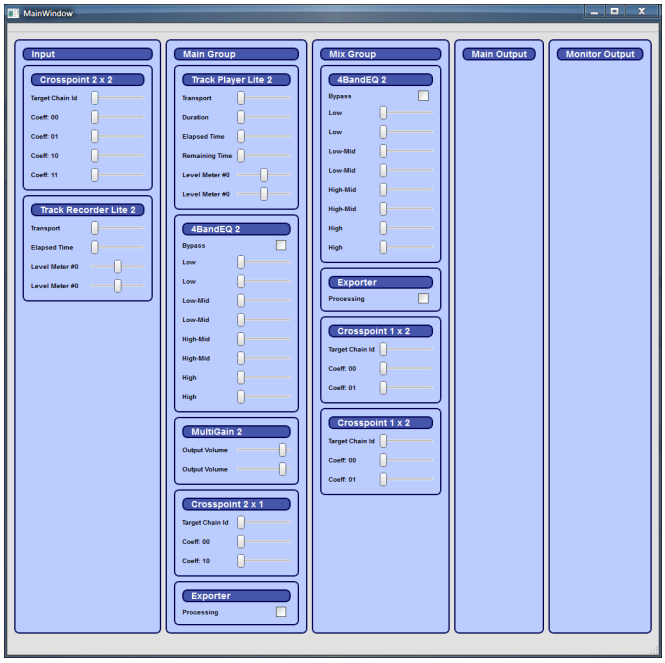
Model rozhraní byl získán transformací pomocí XSLT šablony, která z datového souboru v původním formátu IXML extrahuje atributy a seskupí je implicitním způsobem podle struktury naznačené v samotném souboru. Na jeho základě bylo pak vygenerováno rozhraní zobrazené na obrázku 6.5.

Lepšího výsledku, který by byl více na míru konkrétnímu produktu, bychom dosáhli úpravou rozložení v jednotlivých audio modulech. Vzhledem k vazbě XSLT šablony na konkrétní produkt lze jednoduše skupinám atributů pro audio moduly nastavit rozložení pomocí XML atributu `layout` (viz tabulka 5.1). Výsledek takové úpravy ilustruje obrázek 6.6.

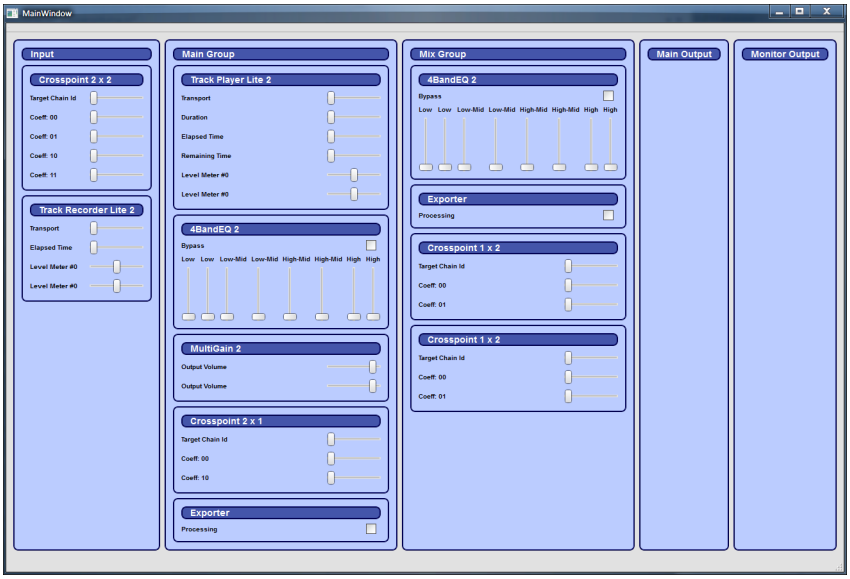
V podobném duchu lze pokračovat dále, například můžeme na základě určitých příznaků detekovat atributy odpovídající ekvalizéru. Tyto atributy pak explicitně sdružíme do nové skupiny a dosáhneme jejich vizuálního oddělení od nesouvisejících prvků. Rozhraní vzniklé touto úpravou je zobrazeno na obrázku 6.7.



Obrázek 6.5: Vygenerované rozhraní



Obrázek 6.6: Rozhraní po úpravě rozložení modulů



Obrázek 6.7: Rozhraní po sdružení atributů pro ekvalizér

Kapitola 7

Závěr

V práci bylo představeno a vyhodnoceno několik přístupů pro generování GUI a pro vzdálené GUI včetně některých existujících implementací.

Po zpracování návrhů firmy DISK Multimedia, s.r.o. byly specifikovány požadavky na vzdálené uživatelské rozhraní pro použití v reálném čase, nezávislé na platformě a přizpůsobitelné druhu zařízení.

Stávající formát pro serializaci dat aplikací vyvinutý zmíněnou společností nebyl ve své současné podobě shledán použitelný pro plně automatické generování uživatelského rozhraní. Z důvodu požadavku obecného použití systému byl navržen a implementován vlastní formát pro reprezentaci datových modelů. Zároveň však byly předloženy návrhy na úpravu stávajícího formátu za účelem přiblížení jeho struktury požadavkům pro generování rozhraní.

Na základě specifikovaných požadavků byl navržen vlastní systém pro vzdálené generování GUI založený na strukturovaném popisu rozhraní a asynchronních zpráv pro jeho aktualizaci.

Implementovaný systém byl ověřen v demonstračním prostředí založeném na datovém modelu skutečné aplikace společnosti DISK Multimedia, s.r.o.

7.1 Směr dalšího vývoje

Jedním z klíčových úkolů dalšího vývoje směřujícího k praktické využitelnosti systému by byla intenzivní práce na usnadnění použití klientské aplikace laické veřejnosti. Cílem je stav, kdy koncový uživatel nepocítí rozdíl mezi používáním běžné aplikace a vzdálené aplikace s automaticky vygenerovaným rozhraním. Konkrétní návrhy k postupu v této oblasti byly představeny v kapitole 5.8.

Dalším úkolem by bylo zajistit dobrou podporu aplikacím. Vhodné by bylo implementovat knihovnu pro použití v konkrétních aplikacích. Tato knihovna by realizovala veškerou funkcionalitu související se vzdáleným uživatelským rozhraním. Implementovala by infrastrukturu TCP/IP serveru a obsluhu připojených klientů. Zajišťovala by serializaci modelu do navrženého formátu a jeho distribuci klientům, interpretaci zpráv od klientů a jejich předání aplikaci například pomocí systému událostí. Ve výsledku by pak úlohou koncového vývojáře bylo pouze doplnit svou aplikaci o kompatibilní datový model.

Vhodné by bylo doplnit klientskou aplikaci o systém zásuvných modulů umožňující rozšířit paletu ovládacích prvků o specifické ovládací prvky pro danou aplikaci nebo rodinu aplikací. Tímto způsobem by se dala realizovat i podpora grafických motivů.

V rámci zvýšení efektivity a výkonu při přenosech přes počítačovou síť by bylo vhodné nahradit navržený formát modelu úsporným binárním formátem s principiálně shodnou strukturou.

Literatura

[1] Bogotobogo: Design Patterns - Observer Pattern. [Online; ověřeno: 25. května 2011].
URL http://www.bogotobogo.com/DesignPatterns/observer.html

[2] The GTK+ Project. [Online; ověřeno: 25. května 2011].
URL http://www.gtk.org/

[3] Introduction to Property List Programming Topics for Core Foundation. [Online; ověřeno: 25. května 2011].
URL http://developer.apple.com/library/mac/#documentation/CoreFoundation/Conceptual/CFPropertyLists/CFPropertyLists.html

[4] Motif. [Online; ověřeno: 25. května 2011].
URL http://www.opengroup.org/motif/

[5] OpenXava - Web application development. [Online; ověřeno: 25. května 2011].
URL http://www.openxava.org/web/guest/home

[6] Qt - Cross-platform application and UI framework. [Online; ověřeno: 25. května 2011].
URL http://qt.nokia.com/

[7] SAXON: The XSLT and XQuery Processor. [Online; ověřeno: 25. května 2011].
URL http://saxon.sourceforge.net/

[8] Xming - PC X Server. [Online; ověřeno: 25. května 2011].
URL http://www.straightrunning.com/XmingNotes/

[9] Da Silva, P. P.: User interface declarative models and development environments: a survey. In *Proceedings of the 7th international conference on Design, specification, and verification of interactive systems*, DSV-IS'00, Berlin, Heidelberg: Springer-Verlag, 2001, ISBN 3-540-41663-3, s. 207–226.

[10] Edskes, H.: IE8 overflow and expanding box bugs. 2010, [Online; ověřeno: 25. května 2011].
URL http://www.edskes.net/ie/ie8overflowandexpandingboxbugs.htm

[11] Kadlec, J.: *Code Characterization for Automated User Interface Generation*. Dizertační práce, 2011.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=9453

[12] Kosek, J.: XSLT v příkladech. [Online; ověřeno: 25. května 2011].
URL http://www.kosek.cz/xml/xslt/

[13] Nichols, J.; Faulring, A.: Automatic Interface Generation and Future User Interface Tools. In *Tools ACM CHI 2005 Workshop on The Future of User Interface Design Tools*, 2005.

[14] Paniza, J.: Model-Driven GUI Generation with OpenXava. [Online; ověřeno: 25. května 2011].
URL http://java.dzone.com/articles/automatic-user-interface

[15] Stirewalt, R. E. K.: Automatic Generation of Interactive Systems from Declarative Models. Technická zpráva, Georgia Institute of Technology, 1997.

[16] Vanderdonckt, J.; Chieu, C. K.; Bouillon, L.; aj.: Model-based design, generation, and evaluation of virtual user interfaces. In *Proceedings of the ninth international conference on 3D Web technology*, Web3D '04, New York, NY, USA: ACM, 2004, ISBN 1-58113-845-8, s. 51–60, doi:http://doi.acm.org/10.1145/985040.985048.
URL http://doi.acm.org/10.1145/985040.985048

[17] Wikipedia: Java annotation. [Online; ověřeno: 25. května 2011].
URL http://en.wikipedia.org/wiki/Java_annotation

[18] Wikipedia: X Window System. [Online; ověřeno: 25. května 2011].
URL http://en.wikipedia.org/wiki/X_Window_System

[19] Wikipedia: X Window System core protocol. [Online; ověřeno: 25. května 2011].
URL http://en.wikipedia.org/wiki/X_Window_System_core_protocol

[20] Wikipedia: X.Org Foundation. [Online; ověřeno: 25. května 2011].
URL http://www.x.org

[21] Wikipedia: Internet Explorer box model bug. 2010, [Online; ověřeno: 25. května 2011].
URL http://en.wikipedia.org/wiki/Internet_Explorer_box_model_bug

[22] Zhou, M. X.; Feiner, S. K.: Data characterization for automatically visualizing heterogeneous information. In *Proceedings of the 1996 IEEE Symposium on Information Visualization (INFOVIS '96)*, INFOVIS '96, Washington, DC, USA: IEEE Computer Society, 1996, ISBN 0-8186-7668-X, s. 13–.

Dodatek A

Ukázkové soubory ve formátu IXML

Kód A.1: Ukázka kódování dat pomocí struktury IXML

```
<key>Effect 00</key>
<dict>
  <key>#Effect Name</key>
  <string>Noise Gate</string>
  <key>Effect Bypass</key>
  <integer>0</integer>
  <key>Effect Parameters</key>
  <array>
    <real>0.0</real>
    <real>0.5</real>
    <real>0.025</real>
  </array>
  <key>VST Chunk</key>
  <data>
  </data>
  <key>VST Chunk Size</key>
  <integer>0</integer>
  <key>Window Position X</key>
  <integer>0</integer>
  <key>Window Position Y</key>
  <integer>0</integer>
</dict>
```

Kód A.2: Struktura zprávy protokolu IXML

```
<dict>
  <key>Address</key>
  <string>Module Address String</string>
  <key>Key Address</key>
  <string>Key Address String</string>
  <key>Message Type</key>
  <string>Message Block, Configuration, Parameter, Event</string>
  <key>Command</key>
  <string>Set, Get, Add, Change, Delete, Reset</string>
  <key>Data</key>
  <dict>
    ...
```

```
    XML Deskriptor modulu
    ...
  </dict>
</dict>
```

Kód A.3: IXML popis mixážní konzole (zkráceno)

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN" "http://www
    .apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
    <dict>
        <key>Program Name</key>
        <string>2011_02_19 - 23_44_55</string>
        <key>Chain Configs</key>
        <dict>
            <key>Chain Config 000</key>
            <dict>
                <key>Chain Name</key>
                <string>Input</string>
                <key>Chain Type</key>
                <string>Input</string>
                <key>Content ID</key>
                <integer>0</integer>
                <key>Input Channels Setup</key>
                <array>
                    <integer>0</integer>
                    <integer>1</integer>
                </array>
                <key>Output Channels Setup</key>
                <array>
                    <integer>0</integer>
                    <integer>1</integer>
                </array>
                <key>Input Levels</key>
                <array>
                    <real>1</real>
                    <real>1</real>
                </array>
                <key>Output Levels</key>
                <array>
                    <real>1</real>
                    <real>1</real>
                </array>
                <key>Channel Names</key>
                <array>
                    <string>Channel 1</string>
                    <string>Channel 2</string>
                </array>
                <key>Channels Mute</key>
                <array>
                    <false />
                    <false />
                </array>
            </dict>
            <key>Chain Config 001</key>
            <dict>
                 ......
```

```
    </dict>
    <key>Chain Config 002</key>
    <dict>
         ......
    </dict>
    <key>Chain Config 003</key>
    <dict>
         ......
    </dict>
    <key>Chain Config 004</key>
    <dict>
         ......
    </dict>
</dict>
<key>Chain Contents</key>
<dict>
    <key>Chain Content 000</key>
    <dict>
        <key>Effect Module 000</key>
        <dict>
            <key>Name</key>
            <string>Crosspoint 2 x 2</string>
            <key>Parameters</key>
            <array>
                <dict>
                    <key>Default Value</key>
                    <real>0</real>
                    <key>Value</key>
                    <real>-1</real>
                    <key>Min Value</key>
                    <real>-1</real>
                    <key>Max Value</key>
                    <real>256</real>
                    <key>Step</key>
                    <real>1</real>
                    <key>Short Name</key>
                    <string>Target Chain Id</string>
                    <key>Units</key>
                    <array>
                        <string></string>
                    </array>
                    <key>Name</key>
                    <string>Target Chain Id</string>
                </dict>
                <dict>
                    <key>Default Value</key>
                    <real>1</real>
                    <key>Value</key>
                    <real>1</real>
                    <key>Min Value</key>
                    <real>0</real>
                    <key>Max Value</key>
                    <real>1</real>
                    <key>Step</key>
                    <real>9.9999997473787516e-005</real>
                    <key>Short Name</key>
                    <string>Coeff: 00</string>
                    <key>Units</key>
                    <array>
```

```
            <string>-</string>
        </array>
        <key>Name</key>
        <string>Coeff: 00</string>
    </dict>
    <dict>
        <key>Default Value</key>
        <real>0</real>
        <key>Value</key>
        <real>0</real>
        <key>Min Value</key>
        <real>0</real>
        <key>Max Value</key>
        <real>1</real>
        <key>Step</key>
        <real>9.9999997473787516e-005</real>
        <key>Short Name</key>
        <string>Coeff: 01</string>
        <key>Units</key>
        <array>
            <string>-</string>
        </array>
        <key>Name</key>
        <string>Coeff: 01</string>
    </dict>
    <dict>
        <key>Default Value</key>
        <real>0</real>
        <key>Value</key>
        <real>0</real>
        <key>Min Value</key>
        <real>0</real>
        <key>Max Value</key>
        <real>1</real>
        <key>Step</key>
        <real>9.9999997473787516e-005</real>
        <key>Short Name</key>
        <string>Coeff: 10</string>
        <key>Units</key>
        <array>
            <string>-</string>
        </array>
        <key>Name</key>
        <string>Coeff: 10</string>
    </dict>
    <dict>
        <key>Default Value</key>
        <real>1</real>
        <key>Value</key>
        <real>1</real>
        <key>Min Value</key>
        <real>0</real>
        <key>Max Value</key>
        <real>1</real>
        <key>Step</key>
        <real>9.9999997473787516e-005</real>
        <key>Short Name</key>
        <string>Coeff: 11</string>
        <key>Units</key>
```

```
                            <array>
                                <string>-</string>
                            </array>
                            <key>Name</key>
                            <string>Coeff: 11</string>
                        </dict>
                    </array>
                    <key>Loaded</key>
                    <true />
                </dict>
                <key>Effect Module 001</key>
                <dict>
                    .........
                </dict>
            </dict>
            <key>Chain Content 001</key>
            <dict>
                .........
            </dict>
            <key>Chain Content 002</key>
            <dict>
                .........
            </dict>
            <key>Chain Content 003</key>
            <dict />
            <key>Chain Content 004</key>
            <dict />
        </dict>
        <key>ID Application</key>
        <string>onStage-100-Win</string>
        <key>ID Data Format</key>
        <string>AFFX-200</string>
        <key>MIDI Routes</key>
        <dict>
            <key>Control Routes</key>
            <dict />
            <key>Note Routes</key>
            <dict />
        </dict>
        <key>Program Tempo</key>
        <integer>120</integer>
        <key>Program Locked</key>
        <false />
        <key>Stored By</key>
        <string>onStage-100-Win</string>
    </dict>
</plist>
```

Dodatek B

Zdrojový kód příkladu OpenXava

Následuje zdrojový kód třídy entity podle které se generuje formulář na obrázku 3.3:

Kód B.1: Příklad popisu rozhraní

```
@Entity
@View(
 members=
  "year, number, date;" +
  "comment;" +
  "customer { customer }" +
  "details { details }" +
  "amounts { amountsSum; vatPercentage; vat }" +
  "deliveries { deliveries }"
)
public class Invoice {

  @Column(length=4)
  private int year;

  @Column(length=6)
  private int number;

  private Date date;

  @Column(length=80)
  private String comment;

  @ManyToOne
  private Customer customer;

  @OneToMany(mappedBy="invoice")
  @ListProperties(
    "serviceType, product.description," +
    "product.unitPriceInPesetas, quantity," +
    "unitPrice, amount")
  private Collection<InvoiceDetail> details;

  @OneToMany(mappedBy="invoice")
  private Collection<Delivery> deliveries;

  // Getters and setters
  ...
```

```
    // Calculated properties
    @Digits(integerDigits=12, fractionalDigits=2)
    public BigDecimal getAmountsSum() { ... }

    @Digits(integerDigits=12, fractionalDigits=2)
    public BigDecimal getVat() { ... }

    @Digits(integerDigits=12, fractionalDigits=2)
    public BigDecimal getTotal() { ... }

}
```

Dodatek C

Zdrojový kód příkladu charakterizovaného kódu

Následuje zdrojový kód generující rozhraní mediálního přehrávače na obrázku 3.9 (příklad převzat z práce [11]):

Kód C.1: Charakterizovaný zdrojový kód třídy pro přehrávač

```
[CodeCharacterization.FirstObjectSelected]
[DataCharacterization.DataType(DataCharacterization.DataTypes.Atomic)]
[DataCharacterization.DataDomain(DataCharacterization.DataDomains.Entity)
    ]
[DataCharacterization.DataForm(DataCharacterization.DataShapes.Shaped)]
[CodeCharacterization.CodeName("Media Player")]
public class MediaPlayer
{
  [CodeCharacterization.CodeName("Playlist")]
  public Playlist PlayList;

  [CodeCharacterization.OnChanged]
  public event EventHandler Changed { ... }

  [CodeCharacterization.CodeCategory("Player", "PlayPause")]
  [CodeCharacterization.CodeImportance(0)]
  [CodeCharacterization.CodeSense(CodeCharacterization.CodeSenseTypes.
      Command)]
  [CodeCharacterization.CodeDependence("IsMediafileOpened")]
  public void PlayPause() { ... }

  [CodeCharacterization.CodeCategory("Player", "Stop")]
  [CodeCharacterization.CodeImportance(0)]
  [CodeCharacterization.CodeSense(CodeCharacterization.CodeSenseTypes.
      Command)]
  [CodeCharacterization.CodeDependence("IsMediafileOpened")]
  public void Stop() { ... }

  [CodeCharacterization.CodeCategory("Navigation", "Next")]
  [CodeCharacterization.CodeSense(CodeCharacterization.CodeSenseTypes.
      Command)]
  [CodeCharacterization.CodeImportance(0)]
  [CodeCharacterization.CodeDependence("ExistsNextInPlaylist")]
  public void Next() { ... }
```

```
  [CodeCharacterization.CodeCategory("Navigation", "Previous")]
  [CodeCharacterization.CodeSense(CodeCharacterization.CodeSenseTypes.
      Command)]
  [CodeCharacterization.CodeImportance(0)]
  [CodeCharacterization.CodeDependence("ExistsPreviousInPlaylist")]
  public void Previous() { ... }

  [DataCharacterization.DataType(DataCharacterization.DataTypes.Atomic)]
  [DataCharacterization.DataDomain(DataCharacterization.DataDomains.
      Measurement, SpecificName="TimeLine")]
  [DataCharacterization.DataContinuity(DataCharacterization.
      ContinuityTypes.Continuous)]
  [DataCharacterization.DataTransience(DataCharacterization.
      DataTransienceType.Dynamic)]
  [DataCharacterization.DataImportance(0)]
  [CodeCharacterization.CodeName("Time Line")]
  [CodeCharacterization.CodeDependence("IsMediafileOpened")]
  [CodeCharacterization.ParameterRange(0,0)]
  public ulong CurrentPosition { ... }
}
```

Kód C.2: Charakterizovaný zdrojový kód třídy pro položku v seznamu skladeb

```
[DataCharacterization.DataType(DataCharacterization.DataTypes.Structure)]
[DataCharacterization.DataDomain(DataCharacterization.DataDomains.Entity)
    ]
[DataCharacterization.DataTransience(DataCharacterization.
    DataTransienceType.Static)]
public class PlaylistEntry
{
  [DataCharacterization.DataType(DataCharacterization.DataTypes.Atomic)]
  [DataCharacterization.DataDomain(DataCharacterization.DataDomains.
      Concept)]
  [CodeCharacterization.CodeName("File name")]
  public string Filename;

  [CodeCharacterization.CodeName("Play")]
  [CodeCharacterization.CodeDescription("Starts playing of this media
      file.")]
  [CodeCharacterization.CodeSense(CodeCharacterization.CodeSenseTypes.
      Command, Default=true)]
  public void Play(){ ... }
}
```

Kód C.3: Charakterizovaný zdrojový kód třídy pro seznam skladeb

```
[DataCharacterization.DataType(DataCharacterization.DataTypes.Atomic)]
[DataCharacterization.DataDomain(DataCharacterization.DataDomains.Entity)
    ]
[DataCharacterization.DataForm(DataCharacterization.DataShapes.Shaped)]
[DataCharacterization.DataContinuity(DataCharacterization.ContinuityTypes
    .Discreete)]
[DataCharacterization.DataOrdering(DataCharacterization.DataOrderingTypes
    .Quantitative)]
[DataCharacterization.DataRole(DataCharacterization.DataRoleTypes.
    Identification)]
```

```
[DataCharacterization.DataSense(DataCharacterization.DataSenseTypes.List)
   ]
[DataCharacterization.DataImportance(0)]
public class Playlist : IList<PlaylistEntry>
{
  [CodeCharacterization.OnChanged]
  public event EventHandler Changed { ... }

  [CodeCharacterization.CodeName("Add media file")]
  [CodeCharacterization.CodeDescription("Adds media file to the playlist
     ")]
  [CodeCharacterization.CodeCategory("Collection","Add")]
  [CodeCharacterization.CodeImportance(1)]
  public void AddMediaFile(
    [CodeCharacterization.ParameterName("Media File")]
    FileInfo file)
    { ... }

  [CodeCharacterization.CodeName("Add media file")]
  [CodeCharacterization.CodeDescription("Adds media files to the playlist
     ")]
  [CodeCharacterization.CodeCategory("Collection", "Add")]
  [CodeCharacterization.CodeImportance(1)]
  public void AddDirectoryContents(
    [CodeCharacterization.ParameterName("Directory with media files.")]
    DirectoryInfo directory)
    { ... }

  [CodeCharacterization.CodeName("Remove")]
  [CodeCharacterization.CodeDescription("Removes selected item in
     playlist.")]
  [CodeCharacterization.CodeCategory("Collection", "Remove")]
  public void RemoveAt(int index) { ... }

  [CodeCharacterization.CodeCategory("Collection", "Remove All")]
  [CodeCharacterization.CodeDependence("IsNotEmpty")]
  public void Clear() { ... }
}
```

Dodatek D

XSLT šablona

Kód D.1: XSLT šablona pro převod IXML modelu do navrženého formátu

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="2.0">

  <xsl:output method="xml" indent="yes"/>

  <xsl:template match="/">
    <ui>
      <!-- Each Chain -->
      <xsl:for-each select="
        /plist/dict
        /key[text()='Chain Configs']/following-sibling::dict[1]
        /key[starts-with(text(), 'Chain Config')]
        ">
        <xsl:variable name="chain_id" select="substring-after(text(), '
            Chain Config ')"/>
        <!-- Select Config -->
        <xsl:for-each select="following-sibling::dict[1]">
          <group title="{key[text()='Chain Name']/following-sibling::
              string[1]}">
            <xsl:for-each select="
              /plist/dict
              /key[text()='Chain Contents']/following-sibling::dict[1]
              /key[text()=concat('Chain Content ', $chain_id)]/following-
                  sibling::dict[1]
              /key[starts-with(text(), 'Effect Module')]/following-
                  sibling::dict[1]
              ">
              <group title="{key[text()='Name']/following-sibling::string
                  [1]}">

                <xsl:for-each select="
                  key[text()='Parameters']/following-sibling::array
                      [1]/dict
                  ">

                  <attribute name="{key[text()='Short Name']/
                      following-sibling::string[1]}"
                    units="{key[text()='Units']/following-sibling::
                        array[1]/string}"
```

```
                        min="{key[text()='Min Value']/following-sibling::
                            real[1]}"
                        max="{key[text()='Max Value']/following-sibling::
                            real[1]}"
                        step="{key[text()='Step']/following-sibling::real
                            [1]}"
                        value="{key[text()='Value']/following-sibling::
                            real[1]}"
                        />

                    </xsl:for-each>

                </group>
            </xsl:for-each>
          </group>
        </xsl:for-each>
      </xsl:for-each>
    </ui>
  </xsl:template>

</xsl:transform>
```

Dodatek E

Obsah CD

Na přiloženém disku se nacházejí následující složky:

- `thesis` – zdrojové soubory textu práce pro sazbu systémem LaTeX
- `onlineui` – zdrojové soubory implementovaného systému, včetně návodu k použití a k sestavení v souboru `README`.
- `server` – demonstrační server pro testování vzdáleného rozhraní, včetně návodu k použití v souboru `README`.
- `models` – ukázkové soubory s modely v obou formátech
- `transform` – XSLT šablona pro převod formátu modelu, kopie XSLT procesoru SAXON a návod k použití v souboru `README`.