Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra informatiky a výpočetní techniky

Diplomová práce

Interpreter výstupních formátů produktu Oracle Forms

Plzeň, 2013 Petr Křepelka

Prohlášení

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

V Plzni dne 14. června 2013 . . . . . .

Petr Křepelka

Abstract

Oracle Forms Output Formats Interpreter

This thesis deals with the Oracle Forms product and a possibility to use its file formats to implement own form interpreter solution. The Oracle Forms pruduct is introduced first, followed by the analyzis of the XML format created and used by Oracle. The next part contains an analyzis of existing software products similar to Oracle Forms. The last part contains the documentation of the design and implementation of the program solution.

Obsah

1 Úvod

Informační systém STAG vyvinutý Centrem informatizace a výpočetní techniky Západočeské univerzity (CIV), jenž je používaný nejen na Západočeské univerzitě a který slouží pro správu studijní agendy univerzity, je postaven nad databází od firmy Oracle. Pro přístup k datům z databáze jsou v současnosti využívány různé prostředky. Existuje přístup přes web a portálovou aplikaci Portál ZČU, který je primárním prostředkem přístupu k databázi. Webový přístup přes Portál je bezpochyby velmi uživatelsky přívětivý a z pohledu uživatele elegantní, neboť k jeho realizaci je potřeba pouze webový prohlížeč. Nicméně existuje i další způsob připojení k databázi STAGu, jenž CIV podporuje, a to je tlustý klient vytvořený pomocí produktu Oracle Forms. Pro tuto aplikaci je používáno označení STAG klient.

Produkt Oracle Forms umožňuje vytvářet grafické formuláře nad jednotlivými databázovými tabulkami i skupinami tabulek a celkově poskytuje velmi široké možnosti pro tvorbu klientských aplikací nad databází Oracle.

Ovšem v současnosti jsou již znatelné určité znaky zastaralosti této technologie, které komplikují plnohodnotné používání STAG klienta. Je to zejména nemožnost ukládání 16 bitový znaků do databáze a pak také neflexibilní uživatelské rozhraní. A navíc i podpora onoho způsobu nasazení, který používá STAG klient (tj. mód tlustý klient), již ze strany Oraclu dávno vypršela, takže hledání nového řešení je na místě.

Nicméně za vytvořením STAG klienta je velké množství práce – je vytvořena asi stovka formulářů –, a tak by přechod na jinou technologii znamenal její zahození. Proto vznikla myšlenka, zda by nebylo možné nějakým způsobem použít existující soubory, ve kterých jsou formuláře definované, a nahradit původní Oracle Forms Runtime vlastní implementací, která by odstraňovala uvedená omezení.

Cílem této práce je prozkoumat tuto možnost (pochopitelně s předpokladem, že to lze) a vytvořit vlastní interpreter příslušných formátů, resp. souborů Oracle Forms.

Že je budoucnost aplikací vytvořených v Oracle Forms otazníkem nejen pro CIV, ale i řadu dalších společností, dokládá např. loňské setkání vývojářů a organizací používajích Oracle Forms aplikace, které se konalo 15. května v nizozemském Nieuwegeinu.[1] Na setkání se řešily mimo jiné také existující alternativy k Oracle Forms.

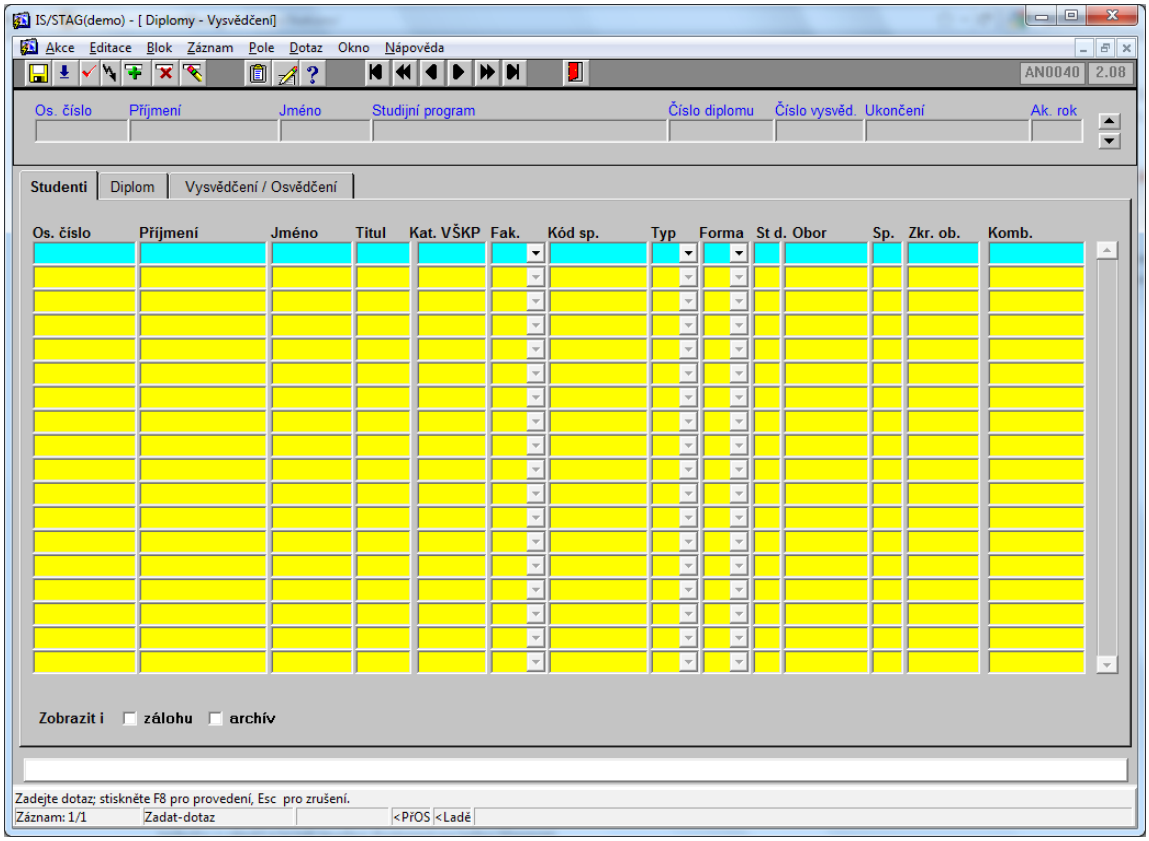
1 http://technology.amis.nl/2012/05/20/the-future-of-forms-presentations-and-other-resources-from-the-may-15th-event-at-amis/

2 Oracle Forms – historie a architektura

Tato kapitola obsahuje obecný úvod k produktu Oracle Forms, přehled jeho historických verzí a vysvětlení architektury.

2.1 Co jsou Oracle Forms

Oracle Forms je prostředek pro tvorbu a spuštění graficky realizovaných formulářů nad databází Oracle. Formuláře pochopitelně slouží pro zobrazování a úpravu dat. Jsou tvořeny grafickým uživatelským rozhraním a aplikační logikou.



Obr. 1: *Náhled na formulář Oracle Forms*

Do formuláře lze umístit všechny běžné grafické prvky jako textová pole, tlačítka, rozbalovací nabídky, atd., které jsou svázané se záznamy v databázi. Na Obr. 1 je náhled formuláře ze STAG klienta.

Pokud jde o aplikační logiku formulářů, je vytvořena v PL/SQL. Podrobně bude vše rozebráno v dalších kapitolách.

2.2 Historie

Produkt Oracle Forms byl Oraclem představen v polovině 80. let jako jeden z prvních průmyslových vývojových nástrojů pro tvorbu databázových aplikací. Oracle Forms tak má za sebou více než pětadvacetiletou – a nutno říci že úspěšnou – dobu existence. Vývojáři databázových aplikací využívali Oracle Forms velmi hojně, což má dopad až do dnešní doby, protože stále existuje velké množství aplikací vytvořených v tomto produktu.

Oracle Forms pochpitelně za tak dlouhou dobu prošly evolucí. Nejdříve pracovaly jen v textovém režimu, ale brzy byl implementován grafický režim, nejdříve pro X Windows, později pro Microsoft Windows (od verze 4). Architektura byla postavena na principu klient-server, přesněji řečeno tlustý klient (runtime, GUI, business logika) a databázový server (více o architektuře v kap. 2.3).

Tento stav trval až do verze 6, kdy byly formuláře přeneseny na webovou technologii. Byla to reakce na rozvoj Internetu a webových technologií. Oracle pozměnil architekturu a mezi klienta a databázový server vložil middleware vrstvu – webový server (Forms Server), do kterého se přesunula business logika. Klient „zeštíhlel“ z tlustého klienta na Java applet, jenž mohl běžet ve webovém prohlížeči. Původní architektura klient-server byla ještě zachována.

Od verze 9 pak existuje jako jediný prostředek nasazení formulářů jen webová platforma a Forms Server. Původní klient-server již není podporován.

2.2.1 Přehled verzí Oracle Forms

Tab. 1 přináší přehled verzí formulářů. Zdrojem pro vypracování bylo [ORAFAQ] a [WIKI].

Název produktu	Verze formulářů	Verze Oracle DB[1]	Poznámky
Forms v1	Forms v1	4 (1984)	první verze
SQL*Forms	Forms v2	5 (1985)	znakový mód
SQL*Forms	Forms v3	6 (1988)	• znakový mód • přidáno PL/SQL pro tvorbu triggerů • běh pod grafickým systémem X
Oracle Forms	Forms v4	6 (1988)	• znakový i GUI mód • běh pod MS Windows i X • uveden binární formát FMB
Oracle Forms	Forms v5	7 (1992)	znakový i GUI mód
Oracle Forms	Forms v6/6i	8/8i (1997/1998)	• Znakový i GUI mód • Architektura rozšířena o webový server (Forms Server) a

1 V závorce rok vydání příslušné verze

			zobrazení formulářů přes webový prohlížeč jako Java applet
Oracle Forms	Forms v9i	9i (2001)	zrušena možnost znakového módu i GUI módu, jediný podporovaný způsob nasazení přes webový server
Oracle Forms	Forms v10g	10g (2003)	zůstává jediný podporovaný způsob nasazení přes webový server
Oracle Forms	Forms v11g	11g (2007)	

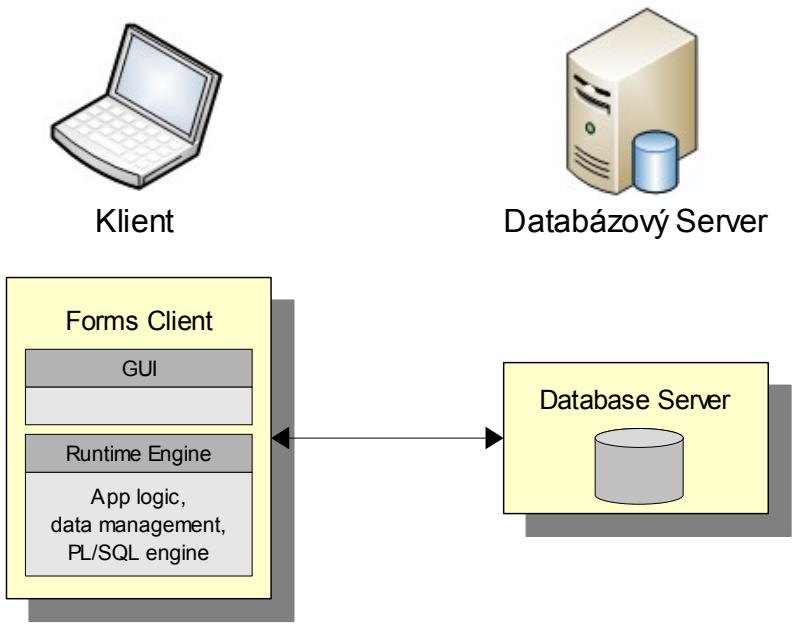
Tab. 1: *Přehled verzí Oracle Forms.*

2.3 Architektura

Pro Oracle Forms byly během jejich vývoje navrženy dvě architektury. Starší architektura klient/server (kap. 2.3.1) byla používána do verze Oracle Forms 9i, novější webová architektura (kap. 2.3.2) byla nasazena ve verzi Oracle Forms 6/6i a používá se do současné verze 11g.

2.3.1 Klient/Server

Na Obr. 2 je znázorněna původní architektura Oracle Forms. Klient obsahuje GUI i aplikační logiku (tzv. tlustý klient). Databázový server poskytuje data a vykonává jen nezbytné operace. Aplikační logika je umístěna v klientu.



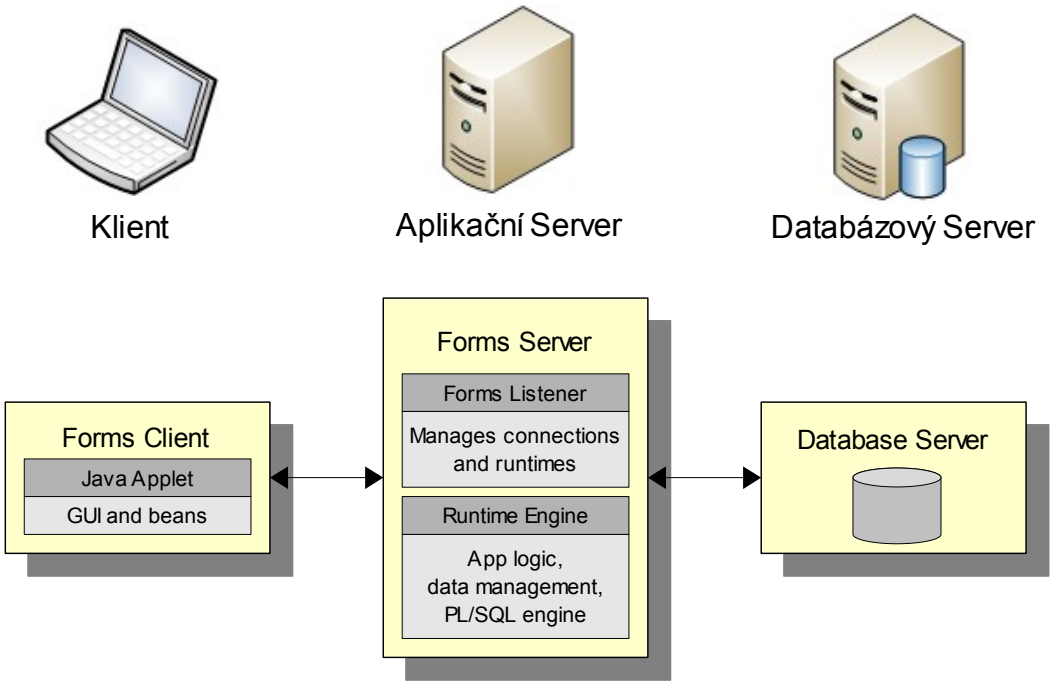
Obr. 2: *Schéma klient/server architektury Oracle Forms*

2.3.2 Webová architektura

Obr. 3 ukazuje současnou architekturu Oracle Forms. Oproti předchozí implementaci přibyla aplikační vrstva v podobě Forms Serveru. Forms Server se skládá ze tří komponent:

- **Java Applet:** Když uživatel zahájí session přes web, tenký, čistě javovský klient se stáhne z aplikačního serveru a spustí se ve webovém prohlížeči. Tento klient poskytuje uživatelské rozhraní pro Runtime Engine na Forms Serveru, a zpracovává interakce s uživatelem jako například navigování mezi položkami nebo zatržení checkboxu. Tento applet je stejný pro spuštění jakéhokoliv formuláře. Není nutné vytvářet žádný Java kód pro každý formulář nebo aplikaci, kterou chceme nasadit.
- **Forms Server Runtime Engine:** Forms Server Runtime Engine vykonává stejné funkce jako Runtime Engine v klient/server architektuře s rozdílem, že veškeré UI akce se přesměrovávají na Java applet. Dále tato komponenta spravuje spojení s databází podle pokynů od klienta. Engine vykonává kód formulářů, menu i knihoven stejným způsobem jako v klient/server módu. Žádná konverze nebo rekompilace pro nasazení formulářů na Forms Server není nutná.
- **Forms Server Listener:** Forms Server Listener je zprostředkovatel, jenž zpracovává požadavky na připojení od klientů a inicializuje instance Forms Server Runtime Enginu. Listener tedy udržuje množinu spojení a bežících instancí runtime enginu.

([ARCH])



Obr. 3: *Schéma webové architektury Oracle Forms*[1]

1 Zdroj: http://documentation.microfocus.com/help/topic/com.microfocus.silkperformer.doc/GUID-E3A72500-A540-4BB1-A292-CD4BAF321CF3-low.png

3 Oracle Forms moduly a objekty

Pro úplné pochopení Oracle Forms je naprosto zásadní porozumět, z čeho se skládají Oracle Forms aplikace, jak se formuláře vytváří, jaké komponenty může obsahovat uživatelské rozhraní, jak funguje aplikační logika formulářů atd. Tato témata se pokusím vysvětlit v této kapitole.

3.1 Moduly a jejich souborové formáty

Oracle poskytuje pro tvorbu formulářů nástroj *Oracle Forms Builder*, který nabízí funkcionalitu pro grafický návrh formuláře a jeho aplikační logiky. S jeho pomocí lze vytvářet formulářové aplikace. Formulářové aplikace jsou tvořeny čtveřicí typů modulů:

- Formuláře (Form Module)
- Menu (Menu Module)
- PL/SQL knihovny (PL/SQL Library)
- Knihovny objektů (Object Library)

Modul formuláře obsahuje definici vzhledu a aplikační logiky (jednoho) formuláře. V modulu menu je definováno menu aplikace (aplikace může mít podle potřeby několik menu modulů). PL/SQL knihovny se skládají z PL/SQL funkcí, procedur a balíků použitelných napříč celou aplikací. A knihovny objektů obsahují definice znovupoužitelných objektů.

Každý modul má vlastní souborové formáty. Obecné má každý modul svůj binární formát, spustitelný formát a textový formát. Mezi sebou lze převádět binární a textové formáty. Spustitelný formát se vytváří z binárního a slouží k vykonávání interpreterem. Převod zpět ze spustitelného formátu na binární není možný. V Tab. 2 je přehled typů souborů pro každý modul. Vedle uvedených typů ještě existuje XML formát, viz kap. 5.

Typ modulu	**Binární formát**	**Spustitelný formát**	**Textový formát**
Formulář	.FMB	.FMX	.FMT
Menu	.MMB	.MMX	.MMT
PL/SQL	.PLL	.PLX	.PLD
Knihovna objektů	.OLB	.OLX	.OLT

Tab. 2: *Přehled souborových formátů Oracle Forms modulů*.

3.2 Formulářové objekty

Formulářový modul je tvořen objekty, které tvoří jak vizuální stránku formuláře, tak jeho logiku. Existují tyto typy objektů:

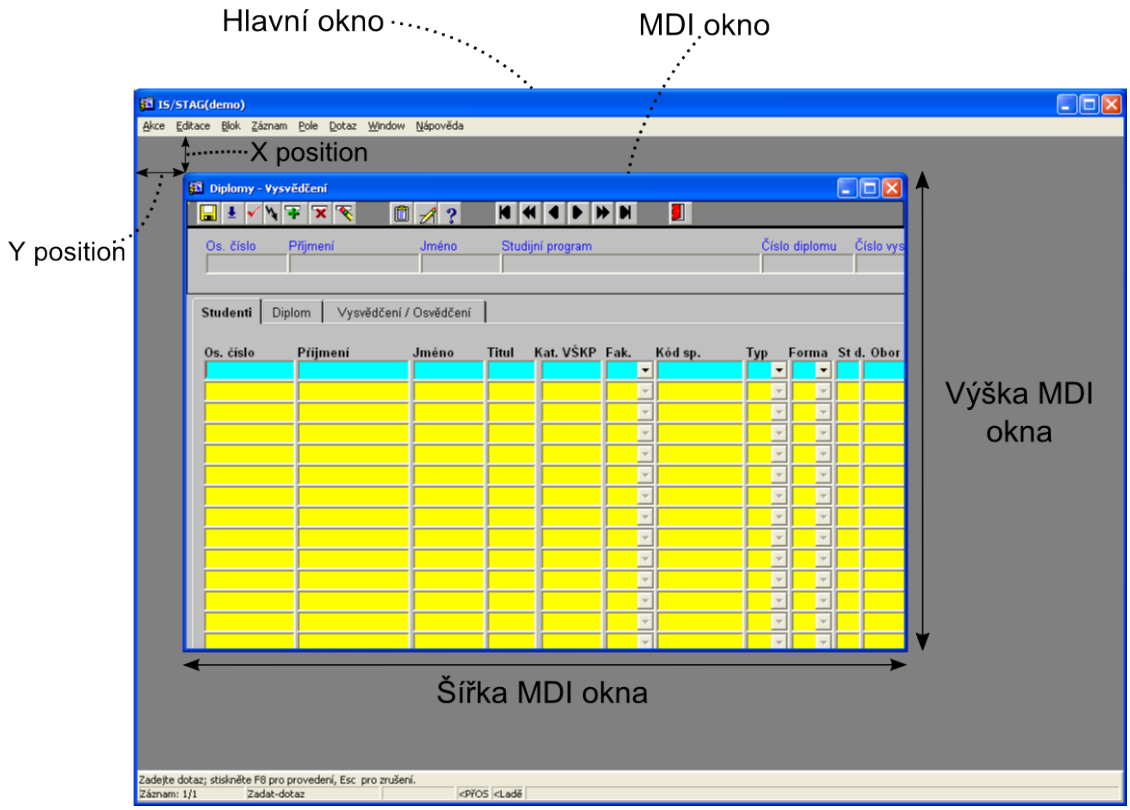
- Alert
- Attached Library
- Block
- Canvas
- Editor
- Item
- LOV
- Parameter
- Popup Menu
- Program Unit
- Property Class
- Record Group
- Trigger
- Visual Attribute
- Window

V následujících podkapitolách proberu funkce každého z uvedených typů. Každý objekt má množinu vlastností, které definují jeho vizuální vlastnosti nebo chování. Důležité vlastnosti budou zpravidla udevedny v tabulkové podobě.

3.2.1 Window

Objekt *Window* definuje okno, hlavní komponentu formuláře. *Window* je kontajner, který obsahuje další komponenty jako *Canvas*, *Block* a *Item* (viz další kapitoly). Jeden formulářový modul může obsahovat více oken. [DEVE, s. 37]

Oracle Forms Runtime je MDI (Multiple Document Interface) aplikace, takže má jedno hlavní okno, uvnitř kterého se nachází kontajner pro samotná formulářová okna (těch může být více než jedno), viz Obr. 4. Vlastnosti objektu obsahuje Tab. 3.



Obr. 4: *Hlavní okno a MDI okno s formulářem.*

Vlastnost	Význam
Name	Jméno objektu
Title	Titulní text v liště okna
Primary Canvas	Jméno primárního content canvasu, který je spojen s oknem
Horizontal Toolbar Canvas Name	Jméno horizontální nástrojové lišty (pokud je součástí okna)
Vertical Toolbar Canvas Name	Jméno vertikální nástrojové lišty (pokud je součástí okna)
Hide On Exit	Určuje, zda se má okno místo zavření jen skrýt
Maximized Allowed	Určuje, zda je povolena maximalizace okna
Minimized Allowed	Určuje, zda je povolena minimalizace okna
Modal	Určuje, zda je okno je modální (tj. zůstává na něm focus, dokud není zavřeno)
Move Allowed	Určuje, zda se může měnit pozice okna
Resize Allowed	Určuje, zda se může měnit velikost okna

Show Horizontal Scrollbar	Určuje, zda je zobrazen horizontální posuvník na dolním okraji okna
Show Vertical Scrollbar	Určuje, zda je zobrazen vertikální posuvník na pravém okraji okna
Height	Výška okna
Width	Šířka okna
Window Style	Styl okna – *Document* nebo *Dialog*
X Position	Souřadnice X pozice levého horního rohu okna. Počátek soustavy souřadnic je umístěn vlevo nahoře.
Y Position	Souřadnice Y pozice levého horního rohu okna
Back Color	Barva pozadí
Foreground Color	Barva popředí

Tab. 3: *Vybrané vlastnosti objektu Window.*

3.2.2 Canvas

Canvas je plocha (plátno) uvnitř okna, na kterou lze umisťovat vizuální objekty jako *textové itemy*, *list itemy*, *image itemy*, *push buttony*, atd. *Canvas* je vždy zobrazen v okně, ke kterému je přiřazen. [DEVE]

V Oracle Forms existuje několik typů *canvasů*:

- Content
- Stacked
- Tab
- Horizontal Toolbar
- Vertical Toolbar

Content canvas je základní a implicitní typ. Zabírá celou plochu okna a pro každé okno musí být tento *canvas* definován.

Stacked canvas je *canvas*, který je umístěn v *content canvasu*, zpravidla má menší rozměry a nemůže existovat sám o sobě. *Stacked canvas* patří do okna, do kterého patří jeho *content canvas*. V jednom okně muže být víc než jeden *stacked canvas*.

Tab canvas umožňuje uspořádat vizuální prvky do záložek. Počet záložek je 2 a více (*canvas* s jednou záložkou nemá smysl). *Tab canvas* – podobně jako *stacked canvas* – musí být umístěn v *content canvasu*. Rovněž platí, že počet *tab canvasů* v jednom okně není omezen.

Vertical toolbar a *horizontal toolbar* jsou dva poslední typy *canvasů* a jsou v podstatě ekvivalentní. Slouží k umístění často používaných *item*, typicky tlačítek. Toolbary se přiřazují k oknu, nikoliv k *content canvasu*. Vertikální *toolbar* je umístěn u levého okraje

okna a má vertikální orientaci, horizontální *toolbar* je umístěn hned pod menu a má horizontální orientaci.

Přehled vlastností objektu *canvas* ukazuje Tab. 4.

Vlastnost	Význam
Name	Jméno objektu
Canvas Type	Typ canvasu, přípustné hodnoty: *Content*, *Stacked*, *Vertical Toolbar*, *Horizintal Toolbar*, *Tab*
Show Horizontal Scrollbar	Určuje, zda se zobrazí horizontální posuvník
Show Vertical Scrollbar	Určuje, zda se zobrazí vertikální posuvník
Visible	Určuje, zda má být canvas zobrazen
Height	Výška canvasu
Width	Šířka canvasu
Tab Action Style	Určuje, jaký styl písma má mít popisek záložky, když je záložka aktivní. Možné hodnoty jsou *Normal* a *Bold* (normálně a tučně).
Tab Attachement Edge	Poloha připojení štítků záložek k *tab canvasu*. Hodnoty jsou *Top*, *Bottom*, *Left*, *Right*, *Start*, *End*.
Tab Style	Styl záložek: *Chamfered* (zkosený), *Square* (hranatý), *Rounded* (zaoblený)
Tab Width Style	Určuje, zda se má šířka štítku záložky měnit podle délky popisku.
Back Color	Barva pozadí
Foreground Color	Barva popředí
Popup Menu Name	Jméno *popup menu* připojeného ke *canvasu*
Window Name	Jméno okna, ve které vlastní canvas

Tab. 4: *Vybrané vlastnosti objektu Canvas.*

3.2.3 Block

Na rozdíl od *window* nebo *canvasu* není *block* vizuální, ale logický objekt. Je to kontejner pro seskupení *itemů* do funkční jednotky pro ukládání, zobrazení a manipulaci s databázovými záznamy. Dva typy *blocků* jsou *Base Table Block* a *Control Block*. *Base Table Block* je spojen s konkrétní tabulkou nebo pohledem v databázi. *Control Block* není spojen s tabulkou ani pohledem. Obsahuje *itemy*, které neumožňují vstup, jako jsou *push button* (tlačítko), *image item* (obrázek) nebo *sound item*. [DEVE, s. 38]

Pokud je v jednom *canvasu* více *blocků*, *blocky* jsou sekvenčně uspořádány do seznamu. Ten slouží k navigování mezi *blocky* za běhu (operace *next* a *previous*). Přehled vlastností objektu *block* ukazuje Tab. 5.

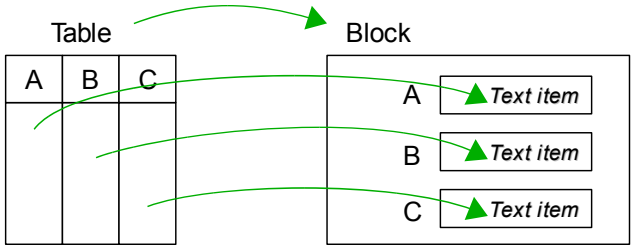
Název	Význam
Name	Jméno objektu
Navigation Style	Má význam pro navigování přes záznamy. Možné tři hodnoty: Same Record (uživatel nemůže procházet záznamy), Change Record (může procházet záznamy) a Change Data Block (pokud víc než jeden block je spojen s jedním canvasem)
Next Navigation Block Name	Jméno následujícího blocku v hierarchii
Previous Navigation Block Name	Jméno předchozího blocku v hierarchii
Records Buffered Count	Minimální počet záznamů načtených v zásobníku
Records Display Count	Počet zobrazených záznamů
Records Fetched Count	Maximální počet záznamů, které se mají načíst z databáze na jeden dotaz
Query All Records	Určuje, zda se mají při načítání záznamů načíst všechny záznamy najednou
Database Block	Určuje, zda je daný block datový *block*
Query Allowed	Určuje, zda je v blocku povoleno dotazování
Query Data Source Type	Typ zdroje dat: *None*, *Table*, *Procedure*, *Transactional Trigger*, *From Clause Query*
Query Data Source Name	Zdroj dat
Order By Clause	Order by klauzule pro specifikování řazení záznamů
Where Clause	Where klauzule pro filtrování záznamů
Insert Allowed	Určuje, zda je povoleno do *blocku* vkládat záznamy
Update Allowed	Určuje, zda je povoleno editovat záznamy v bloku u těch *itemů*, které mají nastavenou vlastnost *Update Allowed* na true (viz Tab. 6)

Delete Allowed	Určuje, zda je v *blocku* povoleno mazat záznamy
Lock Mode	Mód zamykání záznamů: *Immediate* (okamžitý), *Delayed* (zpožděný), *Automatic* (automatický)
Update Changed Columns	Určuje, zda se aktualizují jen sloupce, u nichž se hodnoty skutečně změnily
Maximum Query Time	Maximální doba dotazování dat
Maximum Records Fetched	Maximální počet záznamů, které se mohou celkově načíst v daném *blocku*

Tab. 5: *Vybrané vlastnosti objektu Block.*

3.2.4 Item

Objekt *item* reprezentuje grafické komponenty (textové pole, tlačítko atd.), ale současně v sobě může obsahovat i vazbu na data, např. na konkrétní databázový sloupec. Jak byl uvedeno výše, *itemy* jsou vždy seskupeny v objektu *block*. V *blocku* je taktéž definován zdroj dat, např. databázová tabulka, jedná-li se o *databázový block*. *Item* je namapován na konkrétní sloupec zdroje dat *blocku*, ve kterém se nachází. Na Obr. 6 je znázorněno mapování dat.



Obr. 6: *Mapování databázové tabulky na block*

Objekt typu *item* se dělí na dvě skupiny: *input itemy* (vstupní) a *non-input itemy* (nevstupní). Hlavní rozdíl mezi *input* a *non-input itemy* je, že *input itemy* slouží k dotazování, vkládaní, změnám a mazání databázových záznamů. *Input itemy* dovolují na rozdíl od *non-input* uživateli zadávat vstup.

V Oracle Forms jsou čtyři typy *input itemů*:

- Text Item
- Check Box
- Radio Group
- List Item

Text item je základní typ *itemu*. Je to klasické textové pole zobrazující data jako řetězec. U *text itemu* lze definovat typ dat (řetězec, číslo, datum atd.), maximální počet znaků, formát.

Check box grafický objekt, jenž má dva stavy – *vybrán*, *nevybrán*. Specifikuje se, jaká hodnota odpovídá stavu *vybrán*, jaká hodnota stavu *nevybrán* a jak se *check box* chová pro ostatní hodnoty.

Radio group je skupina tlačítek (*radio button*), kde každé tlačítko reprezentuje hodnotu a vybrat lze jen jedno tlačítko.

List item je rozbalovací nabídka množiny hodnot, z nichž lze jednu vybrat. U *list itemu* je možné oddělit data ve zdroji dat a hodnoty, které vidí uživatel v nabídce.

Non-input itemy se od *input itemů* liší tím, že neumožňují editaci dat, pouze je zobrazují. Do skupiny *non-input itemů* patří následující typy *itemů*:

- Display Item
- Image
- Push Button
- Calculated Item
- Hierarchical Tree
- Bean Area

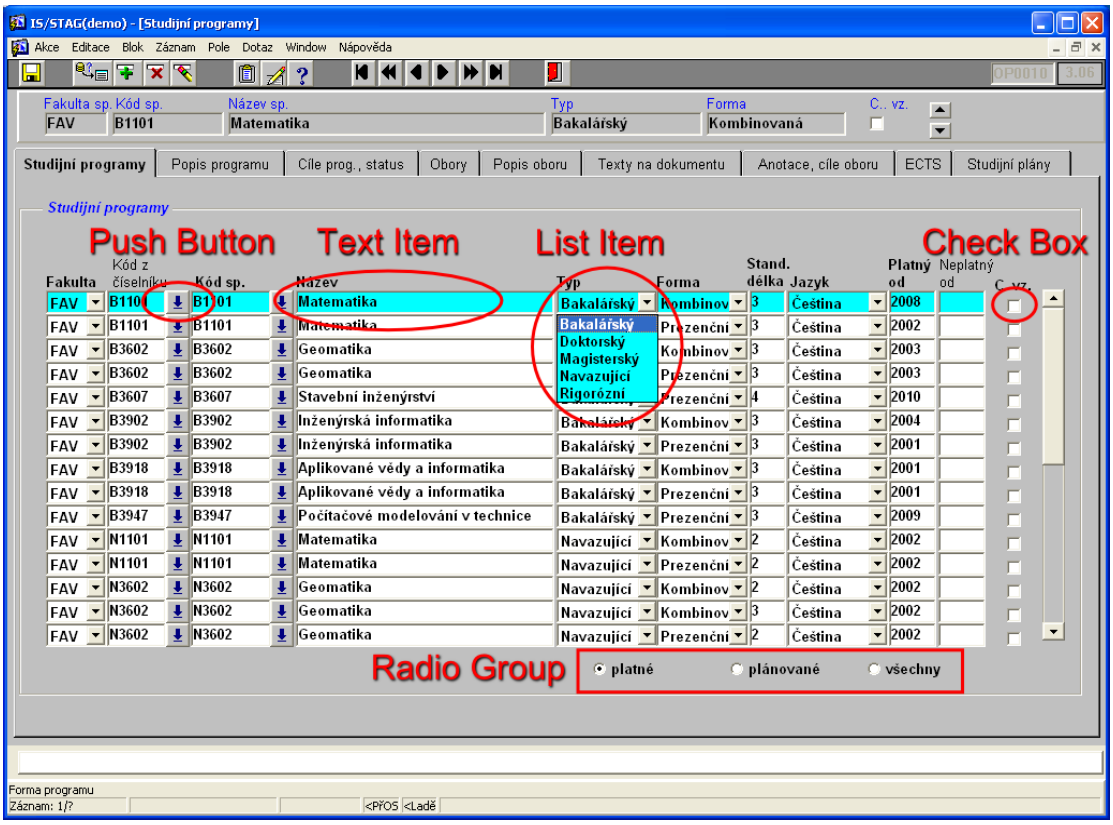
Display item je prostý text umístěný na formulář (popisek). *Image* umožňuje na formulář umístit obrázek, a to buď ze souboru, nebo z databáze pro záznam typu BLOB.

Push button reprezentuje tlačítko a to pochopitelně neslouží pro zobrazení dat, ale pro spouštění akcí typu provedení dotazu do databáze, otevření dalšího okna nebo dialogu, vykonání výpočtu apod.

Calculated item je podobný *display itemu*, ale jeho obsah je vypočítán za běhu. Jsou dva druhy výpočtu: horizontální výpočet používá hodnoty jen z jednoho záznamu a provádí se pro každý záznam (výstupem je *n* výsledků, kde *n* je počet záznamů), vertikální výpočet se provádí přes všechny záznamy a výstupem je jediný výsledek (např. suma).

Hierarchical tree slouží pro vytvoření grafické stromové struktury podobné stromovému zobrazení adresářu a souborů.

Bean area umožňuje umístit do formuláře Java objekt.



Obr. 7: *Typy itemů ve formuláři Studijní programy.*

Název	Význam
Name	Jméno itemu (ekvivalentní se jménem sloupce u databázového itemu)
Item Type	Typ itemu
Height	Výška
Width	Šířka
X Position	Souřadnice X polohy itemu v daném canvasu
Y Position	Souřadnice Y polohy itemu v daném canvasu
Canvas Name	Jméno canvasu, kterému item náleží
Tab Page Name	Jméno záložky, které item náleží
Visible	Určuje, zde se item se zobrazí
Prompt	Popisek u itemu
Prompt Attachment Edge	Poloha popisku vůči *itemu*; přípustné hodnoty jsou *Start* (před), *End* (za), *Top* (nad), *Bottom* (pod)
Label	U některých typů itemů se pro popisek nepoužívá

	Prompt, ale Label (např. Push Button nebo Check Box)
Database Item	Item je databázový, tzn. je spojen s nějakým sloupcem v DB tabulce
Primary Key	Sloupec DB tabulky je primárním klíčem
Insert Allowed	Určuje, zda lze při vkládání záznamu vložit hodnotu
Query Allowed	Určuje, zda lze při dotazování použít hodnotu z toho *itemu* pro filtr vysledků
Update Allowed	Určuje, zda je *item* editovatelný
LOV Name	LOV, který se připojí k textovému itemu

Tab. 6: *Vybrané vlastnosti objektu Item.*

3.2.5 Record Group

Objekt *Record Group* obsahuje definovanou skupinu záznamů. Záznamy, které se načtou do *Record Group* objektu, mohou být definovány dvěma způsoby: *staticky* nebo na základě *dotazu* (dynamicky). Tab. 7 ukazuje vlastnosti objektu *Record Group*.

Název	Význam
Name	Jméno skupiny záznamů
Record Group Type	Typ skupiny záznamů: *static* nebo *query*
Record Group Query	Text dotazu pro typ *query*

Tab. 7: *Vlastnosti objektu Record Group.*

Record Group je podobný dvojrozměrnému poli. Je tvořen sloupci a uvnitř každého sloupce jsou hodnoty. Sloupec je reprezentován objektem typu *Record Group Column*. Sloupce může obsahovat data typu řetězec, číslo nebo datum. V Tab. 8 je přehled vlastností objektu *Record Group Column*.

Název	Význam
Name	Jméno sloupce skupiny záznamů
Column Data Type	Typ hodnot (znak/řetězec, číslo, datum)
Maximum Length	Maximální délka pro znakové hodnoty

Tab. 8: *Vlastnosti objektu Record Group Column.*

Statická skupina záznamů má pevně definované jak sloupce, tak i hodnoty zcela nezávisle na databázových záznamech.

Skupina záznamů typu *dotaz* (query) je definována SQL dotazem. Objekty sloupců odpovídají sloupcům dotazu. Sloupce jsou tedy vytvořeny v době návrhu. Načtení vlastních hodnot z databáze se ovšem provádí až za běhu formuláře. K provedení načtení hodnot slouží build-in funkce *populate_group*, kterou je třeba zavolat v požadované fázi.

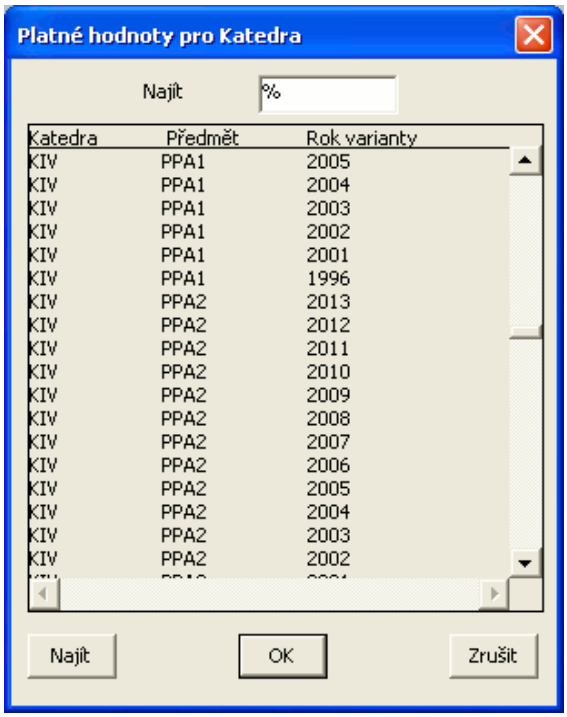
Existuje ještě další způsob vytvoření *Record Group* – zcela dynamicky, jen pomocí PL/SQL kódu, viz Kód 1.

```
DECLARE
  rg_name VARCHAR2(40) := 'Salary_Range';
  rg_id RecordGroup;
  gc_id GroupColumn;
  errcode NUMBER;
BEGIN
  /* Ověření, že skupina záznamů neexistuje */
  rg_id := Find_Group(rg_name);
  /* Pokud neexistuje, vytvoř ji a přidej dva sloupce */
  IF Id_Null(rg_id) THEN
    rg_id := Create_Group(rg_name);
    gc_id := Add_Group_Column(rg_id, 'Base_Sal_Range',
                              NUMBER_COLUMN);
    gc_id := Add_Group_Column(rg_id, 'Emps_In_Range',
                              NUMBER_COLUMN);
  END IF;
  /* Naplň skupinu daty podle dotazu */
  errcode := Populate_Group_With_Query( rg_id,
                'SELECT SAL-MOD(SAL,1000),COUNT(EMPNO) '
              ||'FROM EMP '
              ||'GROUP BY SAL-MOD(SAL,1000) '
              ||'ORDER BY 1');
END;
```

Kód 1: *Vytvoření Record Group v PL/SQL kódu.[REFE, s. 56]*

3.2.6 LOV

LOV je zkratka pro spojení *List of Values* (seznam hodnot). Vizuálně je *LOV* dialogové okno, které zobrazuje seznam hodnot, ze kterých si uživatel může jednu vybrat. *LOV* má podobnou úlohu jako *list item*, ale je mnohem komplexnější. Hodí se pro situaci, kdy existuje velké množství položek, ze kterých je možné vybírat. Pro snadné nalezení konkrétní hodnoty má v sobě implementované vyhledávání. Další odlišnost oproti *list itemu* je, že *LOV* neobsahuje hodnoty „jednorozměrně“, ale „dvojrozměrně“.



Obr. 8: *Náhled LOV dialogu.*

Více napovídá Obr. 8. Tento *LOV* obsahuje tři sloupce. Vybráním řádku tak lze najednou vybrat tři hodnoty, které spolu nějakým způsobem souvisí (nikoliv jen jednu hodnotu). Textové pole v horní části a tlačítko *Najít* slouží k filtrování hodnot (filtruje se jen první sloupec).

Zásadní otázka je, jak se získávají data pro LOV. Každý *LOV* objekt musí mít svůj *Record Group* objekt, který obsahuje data. Dalo by se říci, že *Record Group* je datový model pro *LOV*. A z toho plyne, že *Record Group* musí být naplněný daty ještě před vyvoláním *LOVu*.

Nyní ještě zbývá vysvětlit, co se stane po vybrání řádku a stisku tlačítka OK. *LOV* umožňuje pro každý sloupec definovat *Return Item* (návratová položka), tj. *item* nebo *parameter*, do kterého se vybraná hodnota následně přiřadí. *Return Item* a další vlastností sloupců, které obsahuje *LOV*, jsou definovány v objektech *LOV Column Mapping*. Přehled vlastností objektů *LOV* a *LOV Column Mapping* obsahují Tab. 9 a Tab. 10.

Název	Význam
Name	Jméno *LOV* objektu
Title	Text v liště okna dialogu
Height	Výška dialogového okna
Width	Šířka dialogového okna
Record Group Name	Jméno přidruženého *Record Group* objektu
Auto Column Width	Určuje, zda se má šířka sloupců *LOVu* zvětšit,

	pokud je délka textu v záhlaví sloupce delší než nastavená šířka sloupce
Auto Display	Určuje, zda se má *LOV* zobrazit automaticky při editaci *itemu*, ke kterému je *LOV* připojen (viz vlastnost *LOV Name* v Tab. 6)
Auto Position	Určuje, zda se má pozice dialogu určit s ohledem na pozici *itemu*, ke kterému je *LOV* připojen
Auto Refresh	Určuje, zda se má přidružený *Record Group* objekt načíst pokaždé, když je vyvolán *LOV*, nebo jen při prvním vyvolání *LOVu*
Auto Select	Určuje, zda se má *LOV* automaticky zavřít a vybrat hodnotu, pokud *Record Group* obsahuje jen jeden záznam.
Filter Before Display	Určuje, zda má být před zobrazením *LOVu* zobrazen dialog, do kterého uživatel vloží filtrovací kritéria pro filtrování hodnot z *Record Group*

Tab. 9: *Vlastnosti objektu LOV.*

Název	Význam
Name	Jméno sloupce
Display Width	Šířka sloupce
Return Item	*Item* nebo *parameter*, do kterého se uloží vybraná hodnota z tohoto sloupce
Title	Text v záhlaví sloupce

Tab. 10: *Vlastnosti objektu LOV Column Mapping.*

3.2.7 Editor

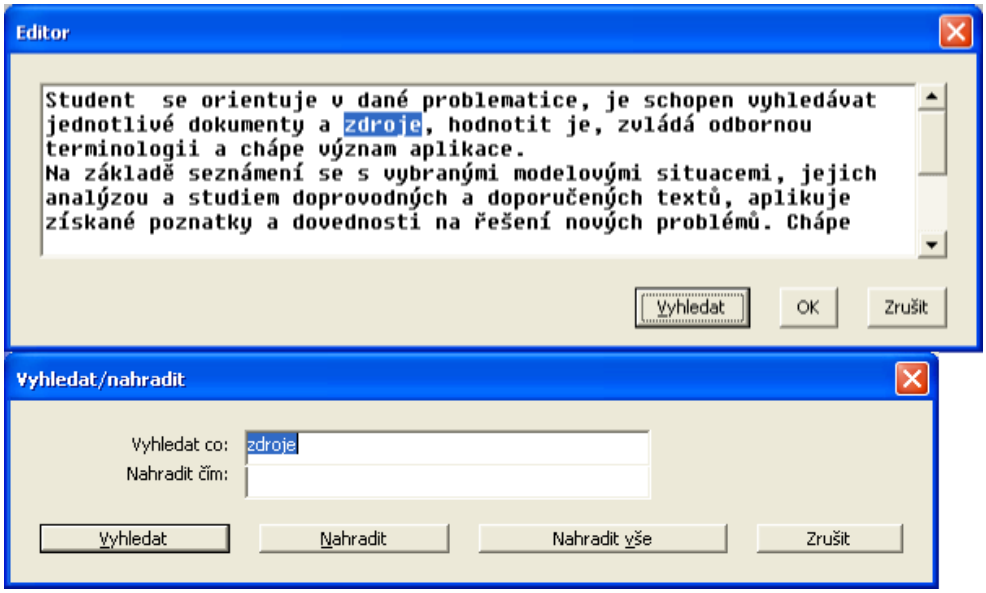
Objekt *Editor* reprezentuje dialog pro úpravu textových dat (*text item*) ve formuláři. Hodí se na editaci dlouhých textů a textů s více řádky. Další funkcionalita *editoru* je vyhledávání a nahrazování textu.

Jak uvádí [DEVE, s. 132], existují tři typy editorů:

- Standardní editor
- Uživatelský editor
- Systémový editor

Standardní editor je zobrazen na Obr. 9 a plní základní editovací a vyhledávací funkci. Je součástí každého formuláře a je ho možné vyvolat z každého *text itemu*. Uživatelský editor má stejnou funkci jako standardní editor, ale může mít jinak nastevené

vlastnosti jako výška, šířka, pozice okna nebo text v liště. Systémový editor je specifický pro operační systém, na kterém běží Oracle Forms Runtime. Např. pro systém Windows může být systémový editor nástroj *notepad*.



Obr. 9: *Okno standardního editoru s vyhledávacím oknem.*

3.2.8 Parameter

Parametry jsou formulářové proměnné definované při návrhu formuláře. Jsou to v podstatě uspořádané dvojice klíč–hodnota. Používají se v situaci, kdy formulář vytváří nový formulář. *Parametry* se přiřazují k objektu *Parameter List*, který se používá v situaci, kdy formulář vytváří nový formulář. *Parameter List* se předává jako parametr při volání build-in procedur *open_form*, *call_form*, *new_form* a *run_product*.

Existují dva typy *parametrů*:

- Textové parametry
- Datové parametry

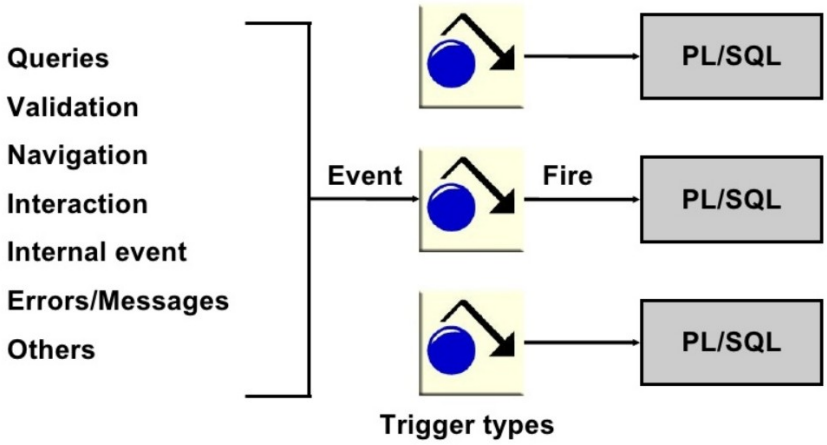
Textový parametr má určitou hodnotu a tato hodnota nemá žádný další význam kromě toho, že je přiřazena k danému parametru. Oproti tomu hodnota datového parametru musí být vždy jméno *Record Group* objektu, který existuje v daném fomuláři. Datové parametry jsou používány pro proceduru *run_product*, nemohou být předány jinému formuláři. [DEVE, s. 350]

Název	Význam
Name	Jméno parametru
Parameter Initialize Value	Počáteční hodnota parametru
Parametr Data Type	Typ hodnoty: Char (znak), Number (číslo), Date (datum)

Tab. 11: *Vlastnosti objektu Parameter.*

3.2.9 Trigger

Trigger je objekt aplikační logiky formuláře, jenž obsahuje PL/SQL kód. Vykonání kódu je spuštěno vždy určitou událostí (načtení formuláře, stisk tlačítka, zavření okna), pro kterou je *trigger* definován.



Obr. 10: *Ilustrace ke triggerům. [PRE13]*

Typy událostí, které spouští *trigger* lze rozdělit podle různých hledisek. Jedno dělení ukazuje Obr. 10. Události dělí na události vzniklé při dotazování dat, validaci dat, navigaci kurozoru ve formuláři, interakci uživatele, interním zpracování, při vzniku chyby a poslední skupina jsou všechny ostatní události.

Každý trigger má 3 klíčové vlastnosti:

- Typ
- Rozsah
- Kód

Typ triggeru je určen událostí, která ho spouští [DEVE, s. 206]. Přehled typů triggerů ukazuje Tab. 12.

Typ triggeru	Význam
When-	Zpracovává takové události, které jsou vyvolány v okamžicích, kdy může být implicitní zpracování události **rozšířeno** o další funkcionalitu. Např. *when-validate-item* trigger se spustí okamžitě po tom, co je vykonána implicitní validace Oracle Forms Runtime.
On-	Zpracovává takové události, které jsou vyvolány v okamžicích, kdy může být implicitní zpracování události **nahrazeno** požadovanou funkcionalitou.
Pre-	Je spuštěn před vykonáním určité akce, např. *pre-query* trigger
Post-	Je spuštěn po vykonáním určité akce, např. *post-query* trigger
Key-	Je spuštěn po stisku definované funkční klávesy uživatelem

Tab. 12: *Přeled typů objektu Trigger.*

Rozsah *triggeru* je úroveň, na které je *trigger* definován ve struktuře formulářového modulu. *Triggery* mohou být definovány na třech urovních:

- Form
- Block
- Item

Trigger s rozsahem *form* je spuštěn vždy při vzniku události, se kterou je spojen. *Trigger* s rozsahem *block* je spuštěn jen tehdy, vznikne-li příslušná událost a *block*, ve kterém je *trigger* definován, je aktuální *block*. Analogicky *trigger* s rozsahem *item* je spuštěn pouze tehdy, vznikne-li příslušná událost a *item*, ve kterém je *trigger* definován, je aktuální *item*.

Kód *triggeru* je PL/SQL kód rozšířený o tři Oracle Forms specifika:

- Build-in (vestavěné) procedury nebo funkce
- Property
- Formulářové proměnné

Build-in procedury a funkce je možné volat v PL/SQL bloku standardním způsobem. Existuje asi 300 těchto procedur a funkcí. Plná specifikace každé z nich se nachází v [REFE].

Property reprezentují vlastnost objektu. Jsou to pojmenované číselné konstanty, jež slouží ke změně nebo získání vlastnosti objektu za běhu. Vlastnosti se mění build-in procedurami se jménem *set_<objekt>_property* (*set_block_property*, *set_canvas_property*, *set_item_property* atd.).

Formulářové proměnné se používají stejně jako jakékoliv jiné proměnné v PL/SQL bloku, ovšem formulářové proměnné není nutné deklarovat. Formulářové proměnné začínají znakem „:“, čímž se odlišují od běžných proměnných. Typy proměnných shrnuje Tab. 13.

Typ proměnné	Účel	Syntaxe
Item	Interakce	:<jméno bloku>.<jméno itemu>
Globální proměnná	Proměnná dostupná po celou session	:GLOBAL.<jméno proměnné>
Systémová proměnná	Stav formuláře	:SYSTEM.<jméno proměnné>
Parametr	Předávání parametru z/do modulu	:PARAMETER.<jméno parametru>

Tab. 13: *Přehled formulářových proměnných.* [LES14]

Kód 2 demostruje uvedená specifika formulářového PL/SQL kódu. *Find_Form* a *Set_Form_Property* jsou build-in procedury, *:System.Current_Form* je formulářová proměnná a *CURSOR_MODE*, *CLOSE_AT_COMMIT*, *VALIDATION_UNIT* a *BLOCK_SCOPE* jsou property.

```
DECLARE
  fm_id FormModule;
BEGIN
  fm_id := Find_Form(:System.Current_Form);
  Set_Form_Property(fm_id,CURSOR_MODE,CLOSE_AT_COMMIT);
  Set_Form_Property(fm_id,VALIDATION_UNIT,BLOCK_SCOPE);
END;
```

Kód 2: *Specifika PL/SQL v Oracle Forms.*

3.2.10 Program Unit

Program unit (programová jednotka) je objekt aplikační logiky formuláře, který obsahuje definici procedury, funkce nebo balíku. PL/SQL kód je zapsán (syntakticky) stejně, jako by byla procedura (resp. funkce nebo balík) vytvořena v databázi. V kódu se zpravidla využívají formulářové speciality jako v *triggerech*. *Program unit* je možné volat z kódu libovolného *triggeru*, lze tak docílit znovupoužitelnosti daného kódu. V Tab. 14 přehled vlastností objektu.

Název	Význam
Name	Jméno objektu
Program Unit Text	PL/SQL kód
Program Unit Type	Typ programové jednotky: *Procedure* (procedura), *Function* (funkce), *Package Spec* (definice balíku), *Package Body* (tělo balíku)

Tab. 14: *Vlastnosti objektu Program Unit.*

3.2.11 Visual Attribute

Objekt *visual atribute* (vizuální atribut) slouží k úpravě vizuálních vlastností objektů. *Visual attribute* lze přiřadit k ostatním objektům přes vlastnost *Visual Attribute Name*. Přehled nastavitelných vlastností zobrazuje Tab. 15.

Název	Význam
Name	Jméno objektu
Visual Attribute Type	Typ vizuálního atributu
Back Color	Barva pozadí
Foreground Color	Barva popředí
Font Name	Jméno písma
Font Size	Velikost písma
Font Spacing	Určuje velikost mezery mezi znaky písma
Font Style	Styl písma
Font Weight	Tloušťka písma

Tab. 15: *Vlastnosti objektu Visual Attribute.*

3.2.12 Alert

Alert reprezentuje klasický třítlačítkový modální dialog pro interakci s uživatelem. Dialog lze přizpůsobit podle požadavků. Kromě zprávy, která je zobrazena uživateli, jsou nastavitelné popisky tlačítek. Tlačítka jsou umístěna zleva (tlačítko 1 je vlevo atd.). V Tab. 16 jsou vybrané vlastnosti objektu.

Název	Význam
Name	Jméno objektu
Title	Text v liště dialogu
Alert Message	Text zprávy
Button 1 Label	Popisek tlačítka 1
Button 2 Label	Popisek tlačítka 2
Button 3 Label	Popisek tlačítka 3
Default Alert Button	Tlačítko, které má být vybráno implicitně při vyvolání dialogu. Možné hodnoty jsou *Button 1*, *Button 2*, *Button 3*.
Alert Style	Styl dialogu. Existují tři možnosti: *Stop*, *Caution*

	(varování), *Note* (poznámka).

Tab. 16: *Vybrané vlastnosti objektu Alert.*

Pro vyolání dialogu slouží build-in funkce *show_alert*, jejíž přesnou specifikaci je možné nalézt v [REFE].

3.2.13 Popup Menu

Popup menu objekt reprezentuje kontextové menu. Lze definovat položky menu a akce při kliknutí na položku. Akce je definována PL/SQL kódem.

3.2.14 Attached Library

Objekt *attached library* slouží k připojení PL/SQL knihovny k formuláři. Knihovna je samostatný modul, který má svůj formát a je definována v samostatném souboru, viz kapitola 3.1. Nejdůležitější vlastnost toho objektu je *Library Location*, která obsahuje cestu k souboru knihovny.

3.2.15 Property Class

Property class (třída vlastností) je objekt, který obsahuje seznam vlastností, které mají ostatní objekty. Je to prostředek pro realizaci dědičnosti. První krok je vytvoření objektu *property class* a nastavení požadovaných vlastností. Druhý krok je přiřazení *property class* k požadovaným objektům.

Existuje několik pravidel a bodů k takto realizované dědičnosti ([TECH]):

- Objekt s připojenou *property class* automaticky dědí její vlastnosti.
- Zděděné vlastnosti mohou být v objektu překryty, pokud je to třeba.
- *Triggery* mohou být součástí *property class*. Objekt, který má definovaný *trigger* a má připojenou *property class* s týmž *triggerem*, *trigger* z *property class* je ignorován.
- *Property class* může být připojen k jinému objektu typu *property class*.
- *Property class* objekty mohou být kopírovány mezi formulářovými moduly.

3.3 Master–detail v Oracle Forms

Zobrazení dat podle vzoru master–detail se hodí v případech, kdy mezi tabulkami existuje vazba 1:N, tzn. jednomu záznamu z hlavní (master) tabulky odpovídá několik záznamů z podřízené (detail) tabulky. Oracle Forms tento způsob zobrazení dat podporuje. Je realizován vytvořením *blocku* pro master záznamy (*master block*) a *blocku* pro detailní

záznamy (*detail block*) a definováním **vztahu** mezi těmito *blocky*. Změna aktuálního záznamu v master *blocku* vyvolá načtení odpovídajících dat do detail *blocku*.

Vztah mezi *blocky* je implementován objektem *relation*, který se definuje v *master blocku*. V relation je definována spojovací podmínka záznamů, což je v podstatě ekvivalent k *where* klauzuli při SQL dotazu nad více tabulkami. V podmínce se mohou používat formulářové proměnné, např.:

```
blok1.item1=blok2.item1 AND blok1.item2=blok2.item2
```

V Tab. 17 se nachází přehled vlastností objektu relation.

Název	Význam
Name	Jméno objektu
Detail Block	Jméno *detail blocku*
Relation Type	Typ spojení, možnosti *Join* a *Ref*
Join Condition	Podmínka spojení záznamů
Delete Record	Určuje chování při mazání master záznamu. Hodnota *Non Isolated* zakazuje mazání master záznamu, pokud existuje detail. *Isolated* dovoluje mazat master záznamy nezávisle na detail záznamech a naopak. A *Cascading* znamená, že se při mazání master záznamu smažou i detailní záznamy.
Prevent Masterless Operation	Zajišťuje, že v *detail blocku* nelze vybírat ani vkládat záznamy, pokud není zobrazen master záznam.
Deferred	Určuje, zda se mají záznamy v *detail blocku* načíst, až když je daný block vybrán (odložené načtení)
Auto Query	Pokud je nastaveno odložené načtení, *Auto Query* určuje, zda se mají detailní záznamy načíst rovnou (automaticky), nebo má být ještě před načtením zadáno filtrovací kritérium.

Tab. 17: *Vlastnosti objektu Relation.*

4 Existující produkty pro konverzi

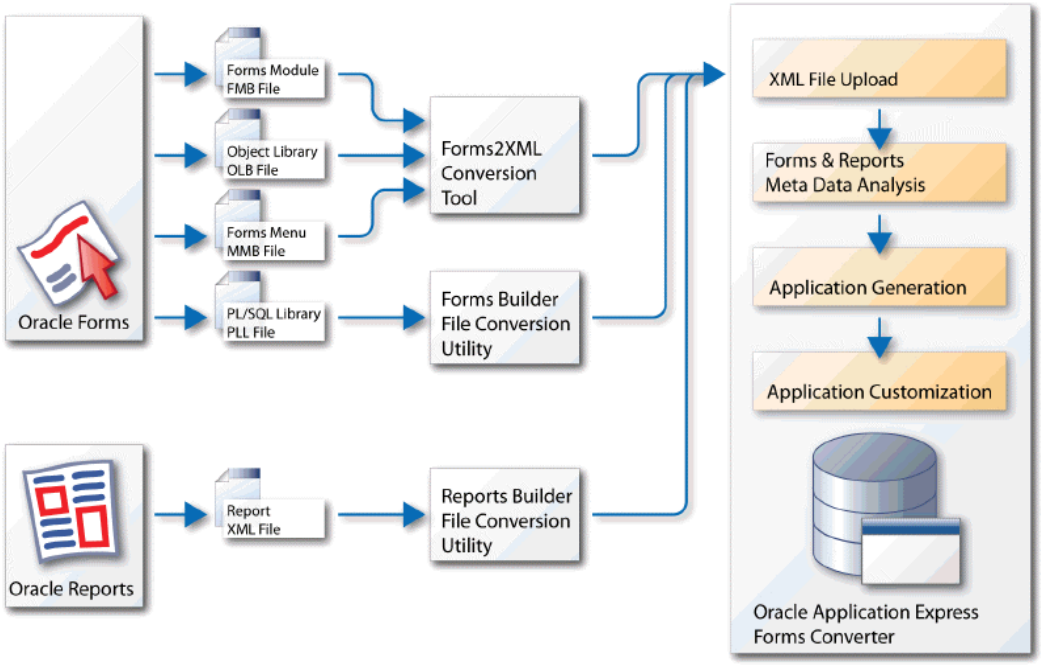
V této kapitole stručně představím nalezené produkty, které mohou sloužit jako alternativa k produktu Oracle Forms. U každého produktu je uveden přehled vlastností a možnost migrace existujících formulářů na danou platformu.

4.1 APEX

APEX je zkratka pro Oracle Application Express. Jedná se o vývojový nástroj pro webové aplikace pro Oracle databázi. Přes grafické dialogy (wizardy) a přímá nastavení lze vytvořit HTML rozhraní nad databází. Výsledná aplikace je provázaná množina HTML stránek obsahující záložky, tlačítka a hyperlinky.

Vývoj aplikací v APEX je do velké míry zjednodušen na grafické programování a vývojáři stačí omezené programátorské zkušenosti.

APEX i Oracle Forms jsou produkty od Oraclu a Oracle pochopitelně nezapomněl na podporu migrace z Oracle Forms na APEX. Existuje manuál, který popisuje, jak se migrace realizuje, viz [APEXM].



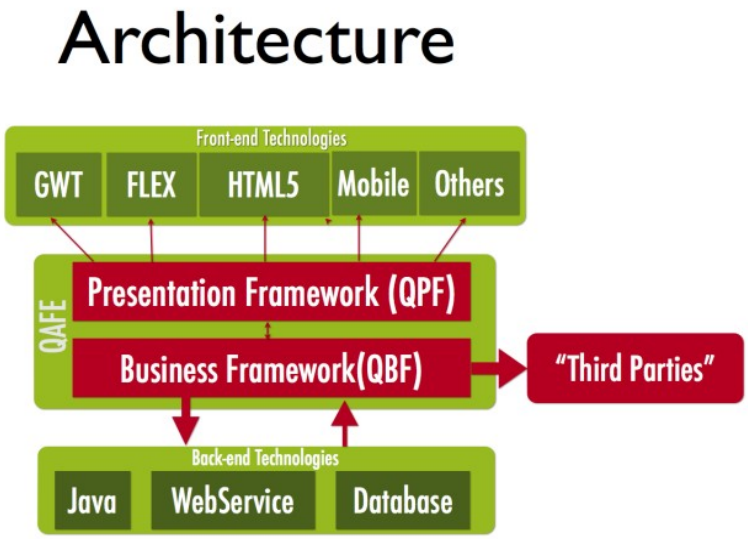
Obr. 11: *Diagram procesu převodu Oracle Forms na APEX.[APEXM, s. 4-2]*

(Zdroj [APEX])

4.2 QAFE

QAFE (zkratka pro Qualogy's Application platform For Enterprises) je platforma, která umožňuje převod Oracle Forms na Web 2.0 aplikaci. Další vlastnosti uvedené výrobcem jsou následující:

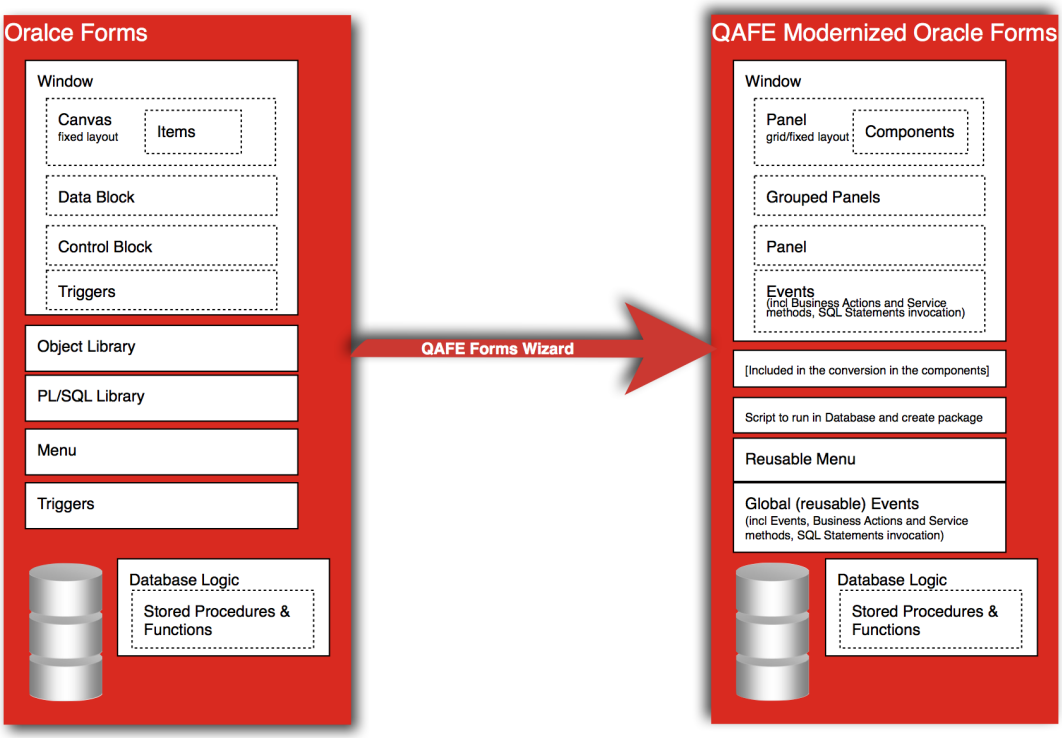
- Plně oddělená prezentační vrstva od business logiky, což podle výrobce zajišťuje použitelnost i pro budoucí webové technologie pro tvorbu uživatelských rozhraní („future proof“)
- Používá SOA (Service Oriented Architecture)
- Prezentační vrstva může být tvořena HTML5, Google Web Toolkit (GWT) nebo Adobe Flex
- Pro vývoj poskytuje vlastní GUI Designer nástroj
- Vývoj v QAML (speciální značkovací jazyk – zkratka pro QAFE's Extensible Markup Language) – není třeba, aby vývojáři uměli Javu nebo nějaký jiný vyšší programovací jazyk, stačí znalost XML a PL/SQL
- Pro převod z FMB na QAFE formulář nástroj QAFE Forms Wizard



Obr. 12: *Architektura platformy QAFE.*[1]

Za pozornost stojí dva obrázky z webu výrobce: na Obr. 12 je znázorněna architektura platformy a na Obr. 13 převod Oracle formuláře uloženém ve FMB na QAFE formulář.

1 Zdroj obrázku: http://www.qafe.com/wp-content/uploads/2011/11/architecture-1024x576.png



Obr. 13: *Převod Oracle Forms na QAFE.*[1]

(Zdroj: www.qafe.com, 20.4.2013)

4.3 Forms2Net

Forms2Net je doplněk (tzv. add-in) do Microsoft Visual Studia od firmy ATX Technologies, který umí zmigrovat formuláře Oracle Forms na Microsoft .NET platformu. Tvoří ho tři části:

- Forms2Net Analyzer – volně dostupný nástroj pro odhad složitosti a náročnosti migrace formulářové alikace
- Forms2Net Converter – konvertuje Oracle formulář z FMB do ekvivalentní entity v HTML5, WPF, Silverlight, ASP.NET, nebo Windows Forms
- Reports2Net Converter – nástroj pro automatickou migraci Oracle Reports na Microsoft Reporting Services

Podporované verze formuláru a reportů jsou od 4.5 do 10g. Oracle Forms formát je transformován podle návrhového vzoru Model–View–Controller a výrobce uvádí, že nástroj umí převést 100% kódu patřícího do business logiky (model), 100% kódu pro uživatelské rozhraní (view) a až 100% kódu controller logiky. Tento nástroj tedy neumí zmigrovat kompletní funkcionalitu formulářů a je na vývojářích, aby chybějící funkce doprogramovali. Je tak zřejmé, že vývojáři na rozdíl např. od QAFE musí disponovat znalostmi .NET technologií.

1 Zdroj obrázku: http://www.qafe.com/developer/docs/images/OracleFormsToQAFE.png

A jelikož je tento produkt postaven na platformě od firmy Microsoft, zmigrovaná aplikace pochopitelně podporuje nejen databázi Oracle, ale i Microsoft SQL Server a další databáze, což může být pro některé projekty zajímavé.

(Zdroj: http://www.forms2net.com, 21.4.2013)

4.4 Formspider

Další produkt, který umí vytvářet formuláře, je Formspider. Jedná se o framework pro vývojáře PL/SQL, který umožňuje vytvořit formulářovou webovou aplikaci. Přednosti produktu jsou podle webu výrobce tyto:

- 100% PL/SQL – vývojář nepotřebuje znát žádný další programovací jazyk, vše se píše čistě v PL/SQL
- AJAX (Asynchronous Javascript and XML)
- Vysoký bezpečnostní standard – Formspider implementuje 9 z 10 OWASP (Open Web Application Security Project) bezpečnostních hrozeb na úrovni frameworku
- Optimalizován pro hlavní webové prohlížeče
- Cloud – Formspider byl od začátku vytvářen s podporou pro Cloud
- Nezávislost na platformě klienta
- Reporty – podpora Oracle Reports a Jasper Reports
- Jednoduchá tvorba formulářů ve vztahu master–detail
- Vlastní webové IDE pro tvorbu formulářů

Vyzdvihl bych ještě jako pozitivum, že Formspider má na svém webu poměrně velkou sekci s návody a tutoriály, jak s frameworkem pracovat.

Další vlastnost – už ne tak úplně přednost – je, že Formspider umí pracovat jen s databází Oracle, a to pouze od verze 10.2.0.1.

Bohužel Formspider neobsahuje žádný nástroj pro migraci Oracle Forms na své formuláře, takže převod formulářů STAG klienta by byl zřejmě manuální záležitostí. Tvorba převodního nástroje byla dříve výrobcem zamýšlena, ale z kapacitních důvodů nebyla do současné verze realizována.

(Zdroj: http://theformspider.com, 21.4.2013)

4.5 Yo!Forms

YoForms nabízí Java platformu pro nasazení Oracle Forms formulářů. Jedná se o webovou aplikaci. Formuláře jsou zobrazeny pouze za použití HTML a Javascriptu. Podle výrobce je to vhodné řešení pro zobrazování formulářů i na mobilních zařízeních.

Celá platforma (pojmenovaná YoServer) se skládá ze čtyř podproduktů:

- YoForms – běhové prostředí na straně serveru pro formuláře a PL/SQL aplikace,

které převádí formuláře na HTML a Javascript

- YoReports – integruje reporty do formulářové aplikace
- YoDeveloper – vývojové prostředí pro tvorbu formulářů a reportů, obsahuje mimo jiné i debugger
- Yo/SQL – kompilátor a běhové prostředí pro vykonávání PL/SQL kódu

Tento produkt je bezesporu velice zajímavý a výjimečný jednou věcí – na rozdíl od předchozích produktů nepoužívá vlastní formát, ale používá přímo Oracle Forms FMB soubory nebo jejich XML variantu. Neboli formuláře se vezmou tak, jak jsou, nahrají se na YoServer a vše by podle tvrzení výrobce mělo fungovat jako s Oracle Forms Runtime (samozřejmě s rozdílem, že ovládání formulářů se provádí přes webový prohlížeč). A totéž platí pro Oracle Reports soubory i menu soubory (MMB).

Navíc výrobce inzeruje další funkce, například opravy chyb v původním Oracle Forms Runtime, ale současně také možnost tyto opravy vypnout a chyby zachovat, aby ty formuláře, které jsou pro ně přizpůsobené, správně fungovaly dál.

Podporované verze formulářů jsou v současnosti 6i až 11g.

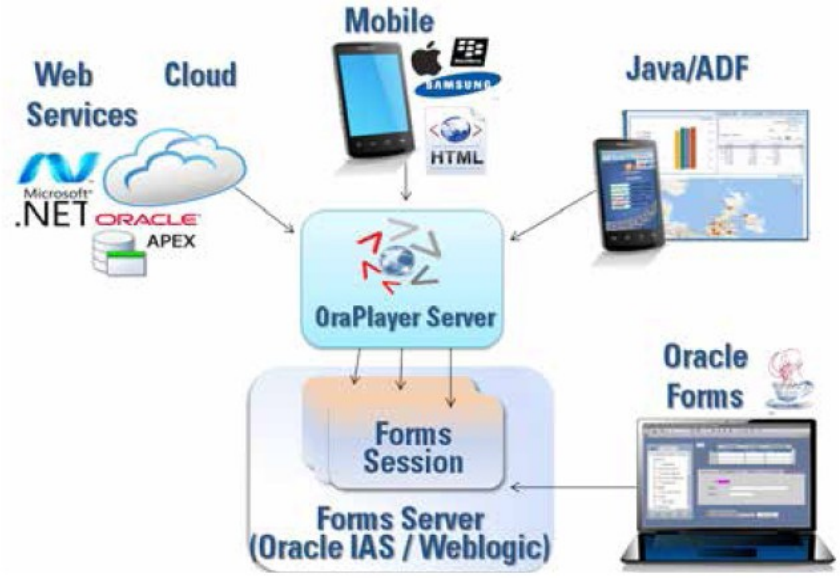
(Zdroj: http://www.quintessencesystems.com/yoforms.php, 22.4.2013)

4.6 OraPlayer

Pod hlavičkou OraPlayer se ve skutečnosti skrývá skupina produktů se stejným cílem – přenést Oracle Forms na jinou platformu. Existují tři varianty:

- produkt Forms to Mobile / PDA
    - umožňuje běh formulářů na mobilních zařízeních (mobilní telefony, iPad, PDA) včetně aplikační logiky formulářů
- produkt Forms to Java / ADF / .NET
    - umožňuje běh formulářů na platformě Java, ADF nebo .NET
- produkt Forms to Cloud / SOA
    - umožňuje integrovat formuláře do SOA aplikace běžící v Cloudu

Podle výrobce všechny produkty fungují bez jakýchkoliv úprav formulářů a dalšího vývoje. Jak je vidět na Obr. 14, OraPlayer rozšiřuje webovou architekturu Oracle Forms o další UI technologie. Aplikační logika pravděpodobně zůstává starostí Forms Serveru.



Obr. 14: *OraPlayer architektura.* [ORAPL]

4.7 Shrnutí

Na závěr přináším souhrnný přehled (Tab. 18) produktů. Tabulka u každého produktu ukazuje použitou technologii, typ nasazení, jak se vytváří formuláře a podporu migrace.

Produkt	Technologie	Nasazení	Tvorba formulářů	Podpora migrace z Oracle Forms
APEX	HTML	Web	Grafické programování	Ano
QAFE	HTML, GWT, Adobe Flex, SOA	Web	QAML	Ano
Forms2Net	HTML, .NET (WPF, ASP, WinForms, Silverlight)	Web, Desktopová aplikace	.NET	Ano
Formspider	AJAX	Web	PL/SQL	Ne
Yo!Forms	HTML, Javascript	Web	Forms Builder	Používá přímo Oracle Forms formáty
OraPlayer	Java, ADF, .NET, SOA, Cloud	Web	Forms Builder	Používá přímo Oracle Forms formáty

Tab. 18*: Souhrnný přehled produktů.*

5 XML formát formulářových modulů

V této kapitole je popsán způsob převodu binárního formátu na XML formát. Dále je vypracována anlýza XML formátu pro jednotlivé typy modulů.

5.1 Převod binárního formátu na XML

Pro vytvoření XML souboru z binárního formátu modulů (viz kap. 3.1) existuje konverzní utilita *frmf2xml*, která je součástí Oracle Forms Builder. Jedná se o skript, který jen předává argumenty volaným Java knihovnám Oraclu, jež dělají konverzi, a v Javovské aplikaci je tedy možné použít přímo tyto knihovny a spouštět konverzi programově.

Skript převádí formáty FMB, MMB a OLB, nepřevádí PLL. Spuštění skriptu viz následující řádek. Užitečné parametry ukazuje Tab. 19.

```
frmf2xml [parametry] soubor1 [soubor2...]
```

Parametr	Hodnoty	Význam
OVERWRITE	YES NO	Implicitní hodnota je NO a znamená, že utilita nevytvoří výstupní XML soubor, pokud již existuje. Hodnota YES znamená, výstupní soubor bude vytvořen vždy znovu.
DUMP	ALL OVERRIDEN	Implicitní hodnota je OVERRIDDEN a znamená, že ve výstupním XML budou u objektů uvedeny jen ty vlastnosti, které mají hodnoty odlišné od implicitních hodnot. Hodnota ALL způsobí, že v objektech budou všechny vlastnosti.

Tab. 19: *Parametry převodního nástroje frmf2xml.*

PL/SQL modul nemá svůj XML formát a ani není potřeba, textový PLD obsahuje PL/SQL kód jako čistý text. Převod z binárního PLL na textový PLD provádí utilita *frmcmp*. PLD se vytvoří následujícím příkazem.

```
frmcmp module=<cesta k PLL> module_type=LIBRARY scipt=YES
```

5.2 XSD schéma

XML vytvořené v předchozí podkapitole má pochopitelně pevnou strukturu a pro zpracování je klíčové ji znát. Validní struktura XML je popsána v XSD (XML Schema

Definition) souboru, který je nutné vygenerovat skriptem *frmxmlsg* z Oracle Forms Builderu. Spouští se bez parametrů a výstupní soubor se jmenuje *forms.xsd*.

5.3 Analýza XML formátu formuláře

XSD ani XML neobsahují žádné zvláštnosti, které by komplikovaly programové zpracování XML. Formulářové objekty mají reprezentovány XML elementy a vlastnosti objektů jsou uloženy v atributech.

Pro názornost přikládám fragment formulářového XML (Kód 3), který obsahuje většinu formulářových objektů (viz kap. 3.2). Plný obsahu souboru jsem zkrátil, aby ukázka nezabrala příliš mnoho místa, ale bylo vidět vše podstatné.

```
<?xml version="1.0" encoding="UTF-8"?>
<Module version="101020002" xmlns="http://xmlns.oracle.com/Forms">
  <FormModule Name="CI0060" ConsoleWindow="ROOT_WINDOW" RuntimeComp="5.0"
              MenuModule="st_menu" FirstNavigationBlockName="B_CIOB"
              ValidationUnit="Item" Title="ČÍSELNÍK OBORŮ STŘEDNÍCH ŠKOL."
              InitializeMenu="">
    <Coordinate CharacterCellWidth="10" CoordinateSystem="Real"
                CharacterCellHeight="20" RealUnit="Pixel"
                DefaultFontScaling="false"/>
    <Block Name="TOOLBAR" SubclassObjectGroup="true" ...>
      ...
    </Block>
    <Block Name="B_INFO" SubclassObjectGroup="true" ...>
      <Item Name="TEXT" SubclassSubObject="true"/>
    </Block>
    <Block Name="B_CIOB" ScrollbarTabPageName="" RecordsBufferedCount="19"
           ScrollbarXPosition="688" OrderByClause="ORDER BY NAZEV"
           RecordVisualAttributeGroupName="CG$CURRENT_RECORD"
           QueryDataSourceName="CIS_OBORY_SS" ScrollbarYPosition="52"
           ScrollbarCanvasName="CG$PAGE_1" ScrollbarLength="352"
           RecordsDisplayCount="16" >
      <Item Name="CIS_OBORU" InsertAllowed="true" Width="78"
            ColumnName="CIS_OBORU" Label="Číslo oboru" Hint="Číslo oboru"
            ItemsDisplay="0" MaximumLength="10" Prompt="Číslo oboru"
            FormatMask="" UpdateAllowed="true" XPosition="92"
            Required="true" YPosition="52" Tooltip="Číslo oboru"
            InitializeValue="" KeyboardNavigable="true"
            PromptAttachmentEdge="Top" PromptAttachmentOffset="0"
            TabPageName="" CanvasName="CG$PAGE_1"/>
      <Item Name="NAZEV" XPosition="180" Width="498" ColumnName="NAZEV"
            YPosition="52" Label="Název" Tooltip="Název" Hint="Název"
            InitializeValue="" MaximumLength="240" ItemsDisplay="0"
            PromptAttachmentEdge="Top" PromptAttachmentOffset="0"
            TabPageName="" CanvasName="CG$PAGE_1" Prompt="Název"
            FormatMask=""/>
    </Block>
    <Canvas Name="CG$PAGE_1" ViewportHeight="440"
            WindowName="ROOT_WINDOW" Height="470" Bevel="None"
            Width="790" ViewportWidth="0">
      <Graphics Name="CG_BOILERPLATE_GROUP" GraphicsText=""
                VerticalMargin="18" GraphicsFontColor=""
                GraphicsFontSpacing="Ultradense" GraphicsFontStyle="0"
                ScrollbarWidth="12" HorizontalMargin="6"
                GraphicsFontSize="0" GraphicsFontWeight="Ultralight"
```

```
                StartPromptOffset="6" GraphicsType="Group"
                GraphicsFontColorCode="0" HorizontalObjectOffset="12"
                GraphicsFontName="">
          ...
        </Graphics>
      </Canvas>
      <Canvas Name="TOOLBAR" SubclassObjectGroup="true"/>
      <Canvas Name="INFO_CANVAS" BackColor="gray"
              SubclassObjectGroup="true" DisplayViewport="true"/>
      <ModuleParameter Name="CG$STARTUP_MODE"
                       ParameterInitializeValue="NORMAL"/>
      <ProgramUnit Name="CG$WHEN_NEW_FORM_INSTANCE"
                   ProgramUnitType="Procedure"
                   ProgramUnitText="/* CG$WHEN_NEW_FORM_INSTANCE
*/&amp;#10;PROCEDURE CG$WHEN_NEW_FORM_INSTANCE IS&amp;#10;BEGIN&amp;#10;/*
CGSM$SHOW_STRIP_MENU */&amp;#10;/* Display the strip menu at the top of
the screen on entry into form */&amp;#10;BEGIN&amp;#10;
show_menu;&amp;#10;END;&amp;#10;END;"/>
      <Trigger Name="PRE-FORM" TriggerText="/* CGAP$TES_SEQUENCE_BEFORE
*/&amp;#10;/* Nastavení správných datumových masek pro implicitní konverze
*/&amp;#10;set_application_property( DATE_FORMAT_COMPATIBILITY_MODE,
'5.0');&amp;#10;set_application_property( BUILTIN_DATE_FORMAT, 'DD.MM.YYYY
HH24:MI:SS');&amp;#10;set_application_property( PLSQL_DATE_FORMAT,
'DD.MM.YYYY HH24:MI:SS');&amp;#10;forms_ddl('ALTER SESSION SET
NLS_DATE_FORMAT = ''DD.MM.YYYY HH24:MI:SS''');"/>
      ...
      <VisualAttribute Name="TOOLTIP_VA" FontSize="800"
                       BackColor="r100g100b88" FontName="MS Sans Serif"
                       ForegroundColor="black" FontWeight="Demilight"
                       FontStyle="Plain" FontSpacing="Normal"/>
      ...
      <Window Name="ROOT_WINDOW" Width="790" Title="Obory středních škol"/>
      ...
    </FormModule>
</Module>
```

Kód 3: *XML fragment formulářového modulu CI0060.*

Přítomnost elementu *FormModule* znamená, že se jedná o formulářový modul (tedy ne menu nebo knihovnu objektů). Atribut *ConsoleWindow* říká, jaké okno se má zobrazit (zde existuje jen jedno okno, ale formuláře obecně mohou mít oken několik). Atribut *MenuModule* je odkaz na menu, které se má vytvořit v hlavním okně (hodnota je jméno menu modulu, jehož obsah je v jiném souboru). *FirstNavigationBlockName* určuje *block*, ve kterém má být kurzor po vytvoření formuláře.

Element *Coordinate* popisuje jednotky souřadnicového systému, ve kterých jsou vyjádřené pozice a rozměry oken, *itemů* a dalších vizuálních prvků. V tomto případě jsou to pixely.

Dále je ve formuláři trojice elementů typu *Block*. První dva odpovídají nástrojové liště, resp. dolní liště, jež jsou součástí téměř každého formuláře a jejichž vlastnosti jsou definovány v *Object Library* modulu. Zaměřil bych se na třetí *Block* B_CIOB, který je databázový a má řadu vlastností. Nejdůležitější informace je v *QueryDataSourceName*, což je zdroj dat *blocku*. Typ zdroje dat není uveden, takže je to implicitně tabulka. *RecordsDisplayCount* říká, že se má zobrazit 16 záznamů, neboli *block* je tvořen 16 řádky záznamů. *RecordsBufferedCount* říká, že při načítání dat se na jeden dotaz načte 19

záznamů. *OrderByClause* upravuje řazení záznamů podle sloupce NAZEV.

K *blocku* B_CIOB náleží dva *itemy*, které odpovídají tabulkovým sloupcům NAZEV a CIS_OBORU (atribut *ColumnName*). Mají řadu vlastností. Pozice *itemu* je určena souřadnicemi (*XPosition* a *YPosition*) relativně ke *canvasu*, na kterém je umístěn (*CanvasName*). *InitializeValue* určuje, jaká hodnota se má nastavit při vytváření nového záznamu. Typ *itemu* je implicitně *text item*.

U *Canvasu* jsou podstatné rozměry (*Height*, *Width*) a okno, ve kterém se nachází (*WindowName*). Typ *canvasu* je implicitně *content canvas*.

Ve formuláři je definován jeden *ModuleParameter* CG$STARTUP_MODE na hodnotu NORMAL.

Další typ elemetu je *ProgramUnit*. Jedná se o proceduru (ProgramUnitType), jejíž kód je v atributu *ProgramUnitText*. Za povšimnutí stojí, že v hodnotě atributu jsou použity HTML entity pro znak LF (
), s čímž je třeba počítat při zpracování. Další věc, se kterou se musí počítat, jsou komentáře.

Element *Trigger* obsahuje definici *triggeru PRE-FORM*. Je zde jen jméno a tělo *triggeru*. Platí stejné poznámky jako u *ProgramUnit*.

Element *VisualAttribute* definuje objekt stejného jména. Vlastnosti jsou popsány v *visual attribute* objektu jsou popsány v kap. 3.2.11. Za pozornost stojí atributy *BackColor* a *ForegroundColor*. Jednou je hodnota vyjádřena konstantou (*black*), podruhé jako rgb hodnota (*r100g100b88*).

Poslední typ elementu je *Window*. Je jím definováno okno formuláře. Zvláštní je, že není nastavena výška okna. Vysvětlení není z ukázky patrné, ale ve formuláři je i *trigger WHEN-NEW-FORM-INSTANCE*, jenž provede maximalizaci okna.

5.4 Analýza XML formátu menu

Ve STAG klientu jsou používány dva menu moduly: STAG2.MMB a ST_MENU.MMB. STAG2 modul obsahuje definici menu s nabídkou všech formulářů. Toto menu se nastaví po přihlášení. Po spuštění formuláře se menu přepne na ST_MENU modul, jenž obsahuje nabídku akcí, které lze provádět nad formulářem (např. provést dotaz).

Připravil jsem fragment modulu STAG2, na kterém se pokusím vysvětlit strukturu XML menu modulu (viz. Kód 4). Ponechal jsem jen to nejnutnější, úplný soubor má přes 7000 řádků.

```
<?xml version="1.0" encoding="UTF-8"?>
<Module version="101020002" xmlns="http://xmlns.oracle.com/Forms">
  <MenuModule Name="stag2" DirtyInfo="true" RoleCount="250"
              MainMenu="MENU2"
              StartupCode=":global.hl_menu_verze:='15.08';&amp;#10;populat
e_spec_menu;"
              UseSecurity="true" MenuFilename="STAG2">
    <MenuModuleRole Index="88" Value="ADMIN_PREDMETU"/>
    <MenuModuleRole Index="61" Value="VYUCUJICI"/>
    <MenuModuleRole Index="5" Value="STUDIJNI_REFERENTKA"/>
    ...
    <Menu Name="MENU2" TearOffMenu="true">
```

```
      <MenuItem Name="PŘIJÍMACÍ_ŘÍZENÍ" RoleCount="1" Hint="Pr_Menu"
                SubMenuName="PR_MENU" CommandType="Menu"
                Label="&amp;Přijímací řízení">
        <MenuItemRole Index="1" Value="USER_COMMON"/>
      </MenuItem>
      <MenuItem Name="ABSOLVENT" RoleCount="1" Hint="Abn_Menu"
                SubMenuName="ABN_MENU" CommandType="Menu"
                Label="&amp;Absolvent">
        <MenuItemRole Index="1" Value="USER_COMMON"/>
      </MenuItem>
      ...
    </Menu>
    ...
    <Menu Name="PR_MENU" TearOffMenu="true">
      <MenuItem Name="KOMISE,_TERMÍNY,_STUDENTI_NA_TERMÍNU" RoleCount="5"
                Hint="Pj0015"
                MenuItemCode="CALL_FORM('PJ0015', HIDE, DO_REPLACE);"
                Label="&amp;Komise, termíny, studenti na termínu">
        <MenuItemRole Index="2" Value="PROREKTOR"/>
        ...
      </MenuItem>
      <MenuItem Name="PŘEVOD_E-PŘIHLÁŠKY_DO_PŘIJÍMACÍHO_ŘÍZENÍ"
                RoleCount="5" Hint="Pj0025"
                MenuItemCode="CALL_FORM('PJ0025', HIDE, DO_REPLACE);"
                Label="&amp;Převod e-přihlášky do přijímacího řízení">
        <MenuItemRole Index="2" Value="PROREKTOR"/>
        ...
      </MenuItem>
      ...
      <MenuItem Name="CARA" RoleCount="1" Hint="Cara" CommandType="Null"
                MenuItemType="Separator" Label="">
        <MenuItemRole Index="1" Value="USER_COMMON"/>
      </MenuItem>
      ...
      <MenuItem Name="SOUHRNY_VÝSLEDKŮ" RoleCount="1"
                Hint="Souhrny_Vysledku" SubMenuName="SOUHRNY_VYSLEDKU"
                CommandType="Menu" Label="So&amp;uhrny výsledků">
        <MenuItemRole Index="1" Value="USER_COMMON"/>
      </MenuItem>
    </Menu>
    <Menu Name="SOUHRNY_VYSLEDKU" TearOffMenu="true">
      <MenuItem Name="SOUHRN_VÝSLEDKŮ_PŘIJÍMACÍHO_ŘÍZENÍ_PODLE_KÓDŮ"
                RoleCount="5" Hint="G_Pr_Sou"
                MenuItemCode="RUN_PRODUCT(REPORTS, 'G_PR_SOU',
SYNCHRONOUS, RUNTIME,FILESYSTEM,'','');"
               Label="&amp;Souhrn výsledků přijímacího řízení podle kódů">
        <MenuItemRole Index="2" Value="PROREKTOR"/>
        ...
      </MenuItem>
      ...
    </Menu>
    ...
    <ProgramUnit Name="VOLEJ_MODUL" ProgramUnitType="Procedure"
                 ProgramUnitText="..."/>
    <ProgramUnit Name="POPULATE_SPEC_MENU" ProgramUnitType="Procedure"
                 ProgramUnitText="..."/>
  </MenuModule>
</Module>
```

Kód 4: XML fragment menu modulu STAG2.

V elementu *MenuModule* jsou tři důležité atributy. *StartUpCode* obsahuje spouštěcí kód, který se vykoná při zavedení menu. *MainMenu* je kořenový menu objekt. *UseSecurity* určuje, zda se má používat zabezpečení, což je mechanismus pro zpřístupnění položek menu podle role uživatele.

V modulu jsou definovány role (element *MenuModelRole*), které jsou stejné jako role v databázi. V nezkráceném souboru je jich asi 250.

Po rolích následují elementy *Menu* a v nich vnořené *MenuItem*. *MenuItem* je jedna položka v menu. Má svůj popisek (atribut *Label*) a kód, který se vykoná při kliknutí na položku (*MenuItemCode*). Vedle položek, které spustí vykonání kódu, jsou v menu ještě položky, které rozbalí podmenu – jsou to ty s hodnotou *Menu* v atributu *CommandType* a v atributu *SubMenuName* mají jméno menu, které se má rozbalit. Poslední typ menu položky je oddělovač – *CommandType* má hodnotu *null* a *MenuItemType* má hodnotu *Separator*.

Při vytváření menu se začne od menu objektu, který je uveden v *MainMenu* atributu modulu. Pokud menu obsahuje položku s *CommandType* hodnotou *Menu*, vytvoří se podmenu podle definice v XML.

U každé položky jsou role (element *MenuItemRole*). Pokud je zapnuto zabezpečení, uživateli, který nemá žádnou z rolí uvedených u položky, není položka zpřístupněna. Aby mohlo být zabezpečení používáno, musí být vytvořen speciální databázový pohled *FMR50_ENABLED_ROLES* s rolemi. Pohled se vytvoří skriptem *frmsec.sql*. Skript musí být spuštěn databázovým administrátorem. ([MENU2])

5.5 Analýza XML formátu knihovny objektů

Formát knihovny objektů je prakticky stejný jako formát formulářového modulu. Místo *FormModule* elementu je použit element *ObjectLibrary* a v něm mohou být vnořené elementy objektů úplně stejně jako ve *FormModulu*. V *ObjectLibrary* se ale vyskytuje ještě jeden objekt navíc – je to *ObjectLibraryTab*. Ten slouží pro rozdělení objektů do záložek v nástroji Form Builder. Jiný význam nemá. Hodnoty atributů *Name* a *Label* nejsou podstatné.

Jak je vidět v ukázce (Kód 5), uvnitř *ObjectLibraryTab* jsou pak opět známé formulářové objekty. *ObjectLibraryTab* není pro zpracování důležitý, v něm vnořené objekty ano.

```
<?xml version="1.0" encoding="UTF-8"?>
<Module version="101020002" xmlns="http://xmlns.oracle.com/Forms">
  <ObjectLibrary Name="OFGSTND" ObjectCount="35">
    ...
    <ObjectLibraryTab Name="MR" ObjectCount="1" Label="MR">
      <Block Name="CGSO$BLOCK_MR" ScrollbarWidth="20" ShowScrollbar="true"
             ScrollbarLength="0" ScrollbarOrientation="Vertical"/>
    </ObjectLibraryTab>
    <ObjectLibraryTab Name="MD"  ObjectCount="3" Label="MD">
      <Item Name="CGSO$CHECK_BOX_MD" FontSize="1000" AutoHint="true"
            Height="22" FontName="Arial" PromptFontName="Arial"
```

```
            PromptFontSpacing="Normal" PromptFontSize="1000"
            ForegroundColor="black"
            Width="667" QueryLength="255"
            TooltipVisualAttributeGroup="TOOLTIP_VA" FontSpacing="Normal"
            FillPattern="transparent" FontWeight="Bold"
            PromptAttachmentOffset="1" ItemType="Check Box"
            FontStyle="Plain" PromptForegroundColor="black"
            PromptFontStyle="Plain" PromptFontWeight="Bold"/>
    </ObjectLibraryTab>
    ...
  </ObjectLibrary>
</Module>
```

Kód 5: *Fragment XML souboru knihovny objektů OFGSTND.*

6 Programová realizace

Tato kapitola popisuje programovou realizaci práce. Obsahuje analýzu klíčových problémů, popis zvolených technologií, navrženého řešení a implementace.

6.1 Analýza

6.1.1 Technologie

Zadání ukládá vytvořit (desktopovou) aplikaci pro platformu Windows, která bude graficky interpretovat XML soubory (formulářů) co nejblíže způsobu, jakým to provádí Oracle Forms Runtime.

Technologie, ve které bude aplikace implementována, by měla poskytovat programovou podporu pro:

- tvorbu grafického uživatelského rozhraní,
- zpracování XML souborů,
- přístup k databázi Oracle.

Jestliže má být vytvořená aplikace co nejvíc podobná svému předobrazu, měla by to být taktéž MDI aplikace.

Předem bych vyloučil použití poněkud zastaralých jazyků typu C, Pascal apod. Jazyk C má nepochybně stále svůj význam, ale není vhodný pro řešení tohoto zadání zejména kvůli své složitosti.

V úvahu připadají dva programovací jazyky – Java a C#. Oba jsou to objektové programovací jazyky, podporují tvorbu GUI, zpracování XML i práci s databází. Java a Swing má podporu pro tvorbu MDI rozhraní (třída *JInternalFrame*). V jazyce C# existují dvě grafické knihovny: starší WinForms a novější WPF (Windows Presentation Foundation). WinForms nativně podporuje tvorbu MDI, WPF nikoliv. Nicméně pro WPF je dostupná knihovna, která rozšiřuje schopnosti WPF právě o MDI.[1]

Oba jazyky tedy splňují základní požadavky a domnívám se, že jsou pro řešení problému stejně vhodné. Každý má určitá specifika, které ovšem nejsou diskvalifikující. Pokud by byl např. požadavek na platformovou nezávislost, pak by byla Java jednoznačná volba, ale takový požadavek není. Naopak je zadáno, že implementace má být pro platformu Windows, takže jsem se rozhodl aplikaci implementovat v jazyce C#, který má k platformě Windows mnohem blíže než Java. Grafické uživatelské rozhraní budu implementovat v novějším a modernějším WPF.

6.1.2 Klíčový problém

Jednoznačně největší problém implementace vlastního interpreteru je zpracování aplikační logiky formulářů, která je napsaná v PL/SQL. A co víc – v PL/SQL rošířeném

1 http://wpfmdi.codeplex.com

o speciality jako vestavěné procedury, formulářové proměnné, property atd.

Řešení Oraclu je jasné – má svůj PL/SQL engine pro Oracle Forms. Neumím říct, jak složitý je to systém, ale hádám, že na něm pracoval tým lidí s tím nejlepším „know how“ o PL/SQL – koneckonců kdo jiný ví o PL/SQL víc než firma, která ho stvořila – a pravděpodobně byl průběžně vyvíjen po dobu několika roků.

Pro zajímavost, jediný produkt z kapitoly 4 – YoForms – implementuje vlastní PL/SQL engine. Ostatní produkty jdou jinou cestou.

Jak vyřešit zpracování PL/SQL ve vytvářeném interpreteru, aby mohla být zachována implementovaná aplikační logika? Jazyk PL/SQL je natolik složitý a komplexní, že bych pravděpodobně nebyl v konečném čase schopen implementovat jeho interpreter, jako to dělá PL/SQL engine v Oracle Forms Runtime. Hledal jsem nějaká existující řešení, která by uměla vykonávat PL/SQL, ale kromě PL/SQL gramatiky pro ANTLR[1] jsem nic užitečného nenašel.

Po provedeném pátrání se tedy zdá, že jediný dostupný prostředek, který umí vykonávat PL/SQL, je databáze Oracle. Základní myšlenka, jak zpracovat PL/SQL v triggerech a programových jednotkách, je vzít PL/SQL kód a poslat ho jako anonymní PL/SQL blok do databáze k vykonání. Ale to nejde jen tak.

Obecně se musí všechna rozšíření Oracle Forms PL/SQL transformovat na standardní PL/SQL. Jsou tu vestavěné procedury, formulářové proměnné a property. Jak je transformace implementována, bude popsáno později (kap. 6.5.3.4). Na tomto místě bych chtěl rozebrat, co přesunutí exekuce PL/SQL z lokálního systému na vzdálený přináší za další problém. Jako příklad použiji vestavěnou proceduru *set_item_property*. Řekněme, že chceme změnit výšku *itemu*.

```
set_item_property('block.item', HEIGHT, 50);
```

V databázi vytvoříme proceduru se stejným jménem a parametry. Potřebujeme, aby se při vykonání procedury – vykoná se v databázi – dostala do aplikace (interpreteru) informace, že se má danému *itemu* zvětšit výška. To může na první pohled vypadat poněkud zvláštně a nestandardně. Z nějakého vzdáleného systému přijde do databáze požadavek, že se má vykonat anonymní PL/SQL blok a jedna konkrétní procedura má implementovat, že se pošle na ten samý vzdálený systém zpráva „zvětši výšku itemu“. Jen s prostředky PL/SQL by to bylo asi obtížené, pokud vůbec možné. V každém případě v databázi Oracle existují tzv. Java Stored procedury. Jsou to uložené procedury (a funkce), jež jsou implementované v Javě, nikoliv v PL/SQL. Tím se rozšiřují možnosti zejména o podporu síťové komunikace.

6.1.3 Uložené Java procedury

Databáze Oracle poskytuje podporu pro vývoj, uložení a nasazení aplikací v Javě. Apliace jsou vykonávány ve speciálním virtuálním stroji – Oracle JVM. Oracle JVM má určité odlišnosti oproti standardnímu virtuálnímu stroji, který je součástí Java Runtime

1 ANTLR (ANother Tool for Language Recognition) umí z definice gramatiky jazyka vygenerovat zdrojové kódy parseru pro lexikální analýzu jazyka.

Environment. Je třeba rozlišit dva pojmy, které souvisí s používáním Javy v databázi:

- Sezení (session) – doba připojení uživatele k databázovému serveru
- Volání (call) – spuštění Java kódu, musí být provedeno v rámci sezení

V databázi Oracle může aplikace běžet jen v kontextu sezení. Každý klient má své sezení a v něm vlastní Java prostředí. Může se zdát, že každý klient má k dispozici svůj vlastní virtuální stroj (implementace je ve skutečnosti efektivnější). Důležitá vlastnost je, že jsou oddělené paměti sezení, takže statické proměnné nastavené v jednom sezení nemění hodnoty statických proměnných v jiných sezeních. Po skončení sezení se existující objekty zruší. ([JAVDE])

Další rozdíl oproti standardnímu Java programu je v určení vstupního bodu programu. Běžně je to statická metoda *main()*. V databázi Oracle může být vstupním bodem Java progamu jakákoliv veřejná statická metoda.

Tyto vlastnosti poměrně významně ovlivňují způsob, jak má být Java aplikace pro Oracle databázi implementována. A ještě jedna neméně podstatná informace, kterou je třeba vzít v úvahu, je verze Javy, která je v databázi nasazena. Pro verzi databáze 11g je to Java 1.5. Verzi Javy nelze nijak nastavit, je pevně vestavěna v databázi.

6.1.4 Komunikace mezi C# a Javou

Další problém, který je potřeba vyřešit, je komunikace mezi částí aplikace napsanou v C# a částí realizovanou v Javě. K dispozici je obecně síťová komunikace a je možné využít její možnosti a vytvořit kupříkladu vlastní komunikační protokol. Ale rozumnější bude použít nějaký existující a osvědčený protokol.

Pro podobné případy byla v minulosti vytvořena celá řada protokolů. Je možné použít webové služby a Simple Object Access Protocol (SOAP), který je založen na textovém přenosu XML, další protokol XML–RPC (XML Remote Procedure Call) nebo binární CORBA (Common Object Request Broker Architecture).

Prinicipálně by měly být funkční všechny uvedené možnosti. Vzhledem k současnému trendu používání webových služeb a předchozím nulovým zkušenostem s jejich implementací jsme využil příležitosti je použít k vyřešení problém a současně se naučit používat novou technologii.

Programovou podporu pro implementaci webových služeb v Javě poskytuje Java API for XML Web Services (JAX-WS), v C# je to WCF (Windows Communication Foundation), jenž je součástí .NET od verze 4.

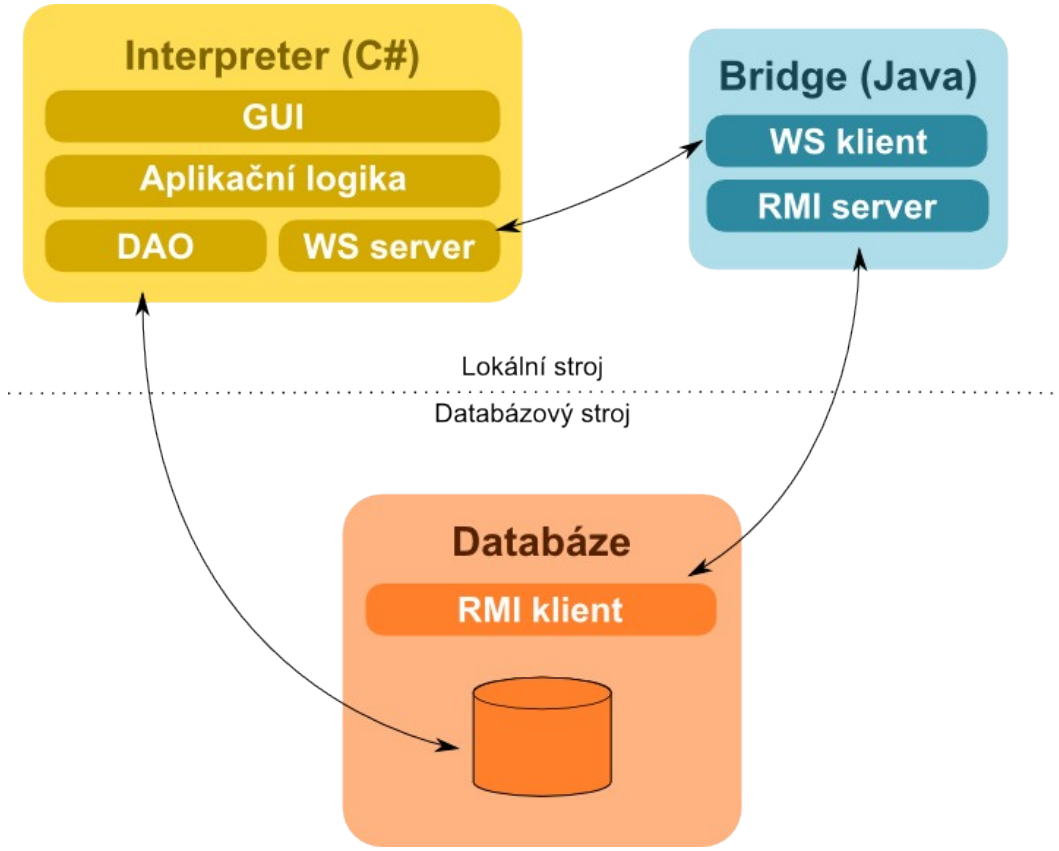
6.2 Architektura

Navržená aplikace sestává ze tří částí/komponent (Obr. 15):

- Interpreter
- Databáze
- Bridge

Interpreter je hlavní komponenta aplikace a je implementována v C#. Běží na lokálním stroji (stroji uživatele). Zpracovává Oracle Forms soubory, vytváří grafické uživatelské rozhraní, zajišťuje interakci s uživatelem, komunikuje s databází, provádí načítání, úpravu a mazání záznamů (DAO), implementuje Oracle Forms built-in procedury, přijímá a zpracovává volání webových služeb (Web Service Server).

Databáze neznamená jen tabulky a data. V kontextu této kapitoly je myšleno především **rozšíření** databáze Oracle o uložené Java procedury a o vrstvu, která realizuje komunikaci s klientským strojem. Jak bylo uvedeno výše, bylo potřeba propojit databázi (Java) s interpreterem (C#). Jako prostředek byly zvoleny webové služby. Ovšem ukázalo se, že nasadit do databáze Java třídy, resp. kód, který by implementoval volání webové služby na interpreteru, není triviální záležitost. Jedno omezení je, že verze Javy v databázi – jak již bylo uvedeno – je 1.5, takže implicitně neobsahuje podporu pro JAXB (Java API for XML Binding) a JAX-WS (Java API for XML Web Services). Tyto a další potřebné



Obr. 15: *Architektura.*

knihovny by bylo nutné uložit do databáze a poté vložit vlastní třídy, které je budou používat.

Při zkušební realizaci takového řešení jsem musel nahrát 7 dalších JAR souborů, s jejichž přítomností teoreticky měl jít kód uložené Java procedury přeložit. Bohužel pokus ztroskotal na nevyřešených závislostech. Kód, který šel mimo databázi úspěšně přeložit, v databázi spustit nešel. Proto vznikla komponenta *Bridge.*

Bridge je most mezi *databází* a *interpeterem*. Myšlenka byla taková, že když nejde jednoduše realizovat volání webových služeb z databáze, vytvoří se mezikrok. Z databáze se bude volat přes RMI (Remote Method Invocation) Java aplikace, která poběží na stejném stroji jako interpreter, a až tato aplikace zavolá požadovanou webovou službu. Tato aplikace není limitována verzí Javy, ani složitým sestavováním a spouštěním v databázi.

Pro realizaci RMI volání z databáze nebyly potřeba žádné další knihovny. Postačuje to, co je standardně obsaženo v JDK 1.5.

6.3 Bridge

Jak již bylo uvedeno, komponenta *bridge* je implementovaná v Javě (verze 1.6) a její účel je přeposílat požadavky z databáze do *interpreteru*. Je tvořena dvěma částmi – Web service klientem a RMI serverem.

6.3.1 Web service klient

Web service klient používá pro volání webových služeb JAX-WS API, které je standardní součástí Java JDK 1.6. Nutno podotknout, že ve verzi 2.1 (současná nejnovější verze je 2.2).

Klient je tvořen třídou `oracleformsinterpreter.webservice.WSClient` a množinou tříd vygenerovaných utilitou *wsdl2java* z frameworku Apache CXF[1]. Utilita generuje rozhraní a třídy podle definice webové služby ve WSDL[2] jazyce, která je automaticky vystavena při registraci webové služby v interpreteru. Z vygenerované množiny tříd je důležitá jedna třída a jedno rozhraní:

- `org.tempuri.OracleFormsInterpreterWS`
- `org.tempuri.IOracleFormsInterpreterWS`

`OracleFormsInterpreterWS` je potomek `javax.xml.ws.Service` a hlavní funkce je, že vytváří implementaci `IOracleFormsInterpreterWS`, což je rozhraní pro volání jednotlivých metod webové služby. Třída `WSClient` už jen zapouzdřuje volání metod rozhraní.

Důležitá poznámka je, že při změně definice webové služby v interpreteru musí být třídy znovu vygenerovány utilitou *wsdl2java* a změny rozhraní zapracovány do `WSClient`. V projektu je připraven skript *wsdl2java.bat*, který volá utilitu s příslušnými parametry a zajistí vygenerování zdrojových kódů do adresáře *generated-sources*.

1 http://cxf.apache.org/
2 Web Services Description Language

6.3.2 RMI server

Implementaci tohoto modulu tvoří třídy z balíku `oracleformsinterpreter.rmi`. `RemoteInterface` definuje rozhraní pro vzdálené volání a jeho metody odpovídají jedna ku jedné rozhraní webové služby interpreteru. Pokud dojde ke změně rozhraní webové služby, musí tomu být přizpůsobeno toto rozhraní.

Rozhraní je implementováno třídou `RemoteInterfaceImpl`. Implementace je přímočará – v každé metodě se zavolá odpovídající metoda webové služby prostřednictvím instance `WSClient`, předají se parametry a případně vrátí výsledek (viz Kód 6).

```
@Override
public String getFormProperty(String form, String name)
                              throws RemoteException {
  logger.debug(String.format(
      "Vzdalene volani getFormProperty(form=%s, name=%s)",
      form, name));

  return client.getFormProperty(form, name);
}
```

Kód 6: *Implementace RemoteInterface.getFormProperty().*

Třída `RMIServer` registruje `Remote` objekt do `LocateRegistry` a současně je to vstupní bod programu, takže zde probíhá načtení jednoduché konfigurace.

6.4 Implementace databázové části

Programová realizace v databázi je tvořena dvěma částmi: Java RMI klientem a PL/SQL balíkem, který obsahuje definici uložených procedur.

6.4.1 RMI klient

RMI klient se skládá z rozhraní `RemoteInterface` a třídy `RMIClient`. `RemoteInterface` je totožné s rozhraním z RMI serveru. Třída `RMIClient` realizuje RMI volání bridge aplikace a je implementována s ohledem na to, že metody budou volány z obalovacích PL/SQL procedur (viz kap. 6.4.2). Tento fakt v praxi znamená, že všechny metody a atributy jsou statické (alespoň takový přístup je demonstrován v [JAVDE]). `RMIClient` obsahuje veřejnou statickou metodu pro každou metodu vzdáleného rozhraní tak, aby bylo pokryto volání všech metod rozhraní.

Konfigurace pro spojení s RMI serverem (adresa serveru, port a jméno vzdáleného objektu) je do třídy dodána zvnějšku přes *set* metody.

6.4.2 PL/SQL balík

PL/SQL balík obsahuje procedury a funkce, které obalují zavolání metod třídy `RMIClient` popsané v předchozí kapitole. Použití PL/SQL balíku bylo zvoleno pro jednoznačné oddělení procedur, které jsou součástí této aplikaci, od ostatních procedur, jež

jsou v databázi z jiných důvodů.

Nejlépe bude vše demonstrovat na ukázce (Kód 7). Vytvořený balík se jmenuje *oraforms*. Definice se skládá ze dvou částí: specifikace a těla. Ve specifikaci jsou deklarovány hlavičky a v těle balíku těla procedur a funkcí. V ukázce je jedna procedura a jedna funkce. Jsou to implementace Oracle Froms build-in procedury *add_list_element* a funkce *populate_group*.

Na hlavičce procedury není nic neobvyklého, za pozornost stojí tělo, kde je specifikováno, jaká Java metoda se při zavolání procedury vykoná. Je použita posloupnost klíčových slov LANGUAGE JAVA NAME a následuje řetězec s určením třídy (plně kvalifikované jméno) a statické metody, v závorce následují typy parametrů a v případě funkce ještě za závorkou následuje typ návratové hodnoty. Počet a typ parametrů procedury musí odpovídat parametrům volané Java metody. Typ VARCHAR2 odpovídá Javovskému Stringu, NUMBER typu Integer.

```
CREATE OR REPLACE PACKAGE oraforms AS

  PROCEDURE add_list_element(
    list_name   VARCHAR2,
    list_index  NUMBER,
    list_label  VARCHAR2,
    list_value  VARCHAR2);

  FUNCTION  populate_group(
    recgrp_name    VARCHAR2)
    RETURN         NUMBER;

END oraforms;

CREATE OR REPLACE PACKAGE BODY oraforms AS

  PROCEDURE add_list_element(
    list_name   VARCHAR2,
    list_index  NUMBER,
    list_label  VARCHAR2,
    list_value  VARCHAR2)
  IS
  LANGUAGE JAVA NAME
      'oracleformsinterpreter.rmi.RmiClient.addListElement(
           java.lang.String, java.lang.Integer, java.lang.String,
           java.lang.String)';



  FUNCTION  populate_group(
    recgrp_name    VARCHAR2)
    RETURN         NUMBER
  IS
  LANGUAGE JAVA NAME
       'oracleformsinterpreter.rmi.RmiClient.populateGroup(
            java.lang.String) return java.lang.Integer';

END oraforms;
```

Kód 7: *Vytvoření PL/SQL balíku s uloženými Java procedurami.*

Volání procedur a funkcí z balíku *oraforms* přináší jeden drobný rozdíl oproti volání nebalíkových procedur. Při volání musí být před jménem procedury uvedeno jméno balíku oddělené od jména procedury tečkou.

6.4.3 Postup nasazení databázové realizace

Jestliže jsou vytvořeny třídy RMI klienta i skript *oraforms* balíku, nasazení se provede ve dvou krocích:

1. Nahrání tříd do databáze
2. Vytvoření balíku

K nahrání Java tříd do databáze Oracle slouží nástroj *loadjava*, který je umístěn v adresáři product\11.2.0\db_1\BIN[1] instalace databáze (pozn. k odstranění tříd slouží *dropjava*). Mohou se nahrát přímo zdrojové kódy nebo class soubory. V projektu je připraven skript *load.bat*, který zavolá *loadjava* nahraje správné třídy z adresáře *src*. Je ovšem potřeba jej nakonfigurovat – nastavit cestu k BIN adresáři a zadat SID databáze a jméno uživatele. Heslo pro přihlášení k databázi bude vyžádáno na příkazovém řádku po spuštění.

Balík je definován ve skriptu *create_package.sql* a lze ho vytvořit obvylkým způsobem, jakým se spouští SQL skripty, např. přes nástroj *sqlplus*.

První krok je nutné vykonat přímo na stroji, kde je instalována databáze, druhý je možné provést vzdáleně.

1 liší se podle verze databáze

6.5 Interpreter

Interpreter je hlavní a nejsložitější komponenta celého systému. Je implementována v .NET frameworku verze 4.0, projekt byl vytvořem ve vývojovém prostředí Microsoft Visual Studio 2010.

6.5.1 Použité knihovny

Vedle standardních knihoven jsou v projektu použity i knihovny, které nejsou součástí .NET frameworku. Jsou to následující:

- WPF.MDI.dll (pro tvorbu MDI, získáno z http://wpfmdi.codeplex.com)
- Oracle.DataAccess.dll (pro připojení k databázi přes ODP.NET, což je zkratka pro Oracle Data Provider for .NET)

6.5.2 Návrh interpreteru

Jak je navržen interpreter, znázorňuje UML diagram na Obr. 16 na následující straně. Nejedná se úplný diagram tříd, kde by byly všechny třídy včetně atributů a metod, ale pouze o hrubý náčrt základních tříd a vazeb. Třídy jsou rozděleny do balíků, které ve výsledné aplikaci odpovídají jmenným prostorům používaných v jazyce C#.

Balík *DataAccess* reprezentuje datovou vrstvu interpreteru (myšleno ve smyslu přístupu k databázovým datům). Účel této vrstvy je vytvářet, udržovat a zavírat spojení s databází, vykonávat dotazy na data, provádět SQL příkazy. Datová vrstva je oddělená rozhraním *DAI* (Data Access Interface) a implementována třídou *DAO* (Data Access Object).

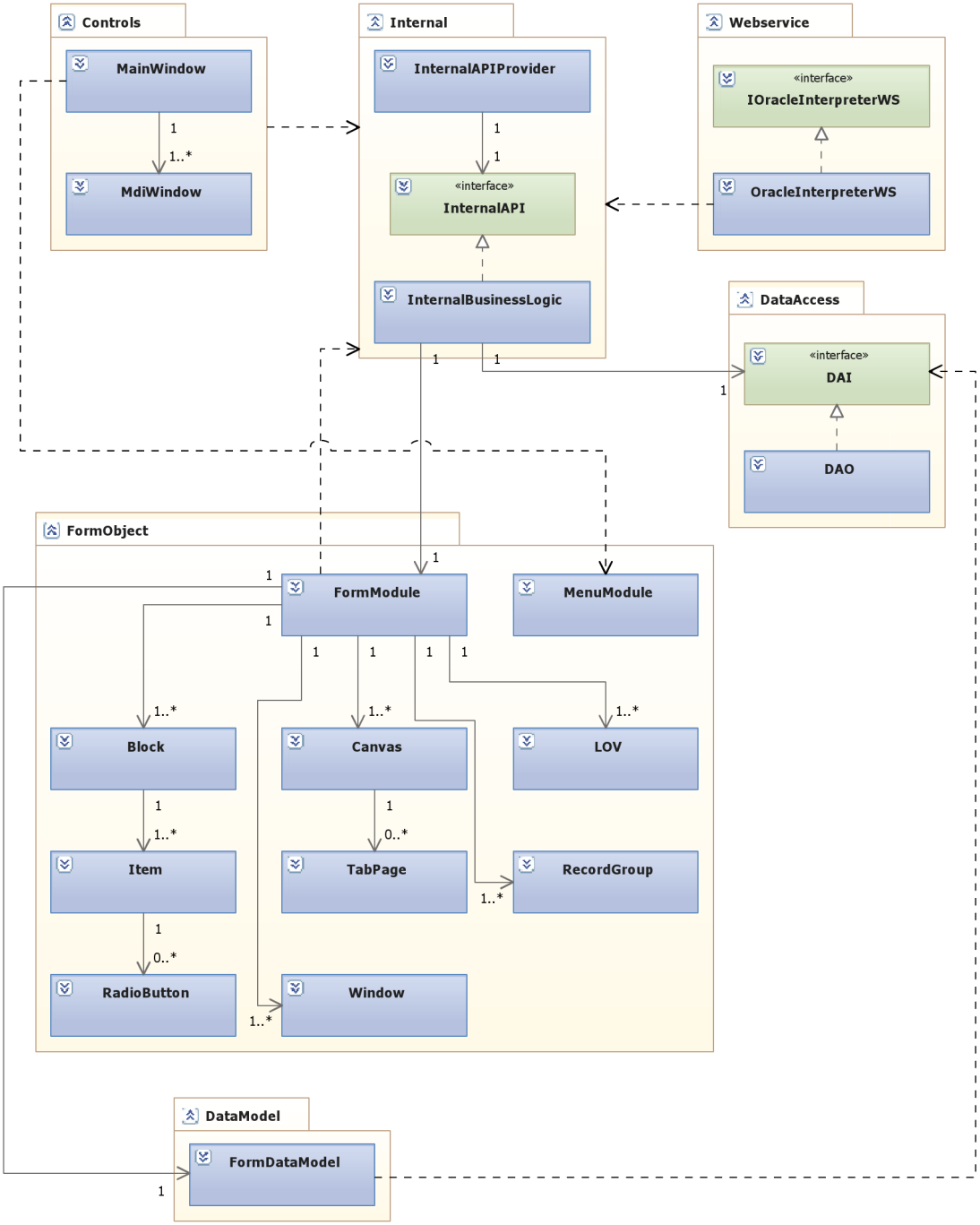
Hierarchicky nad datovou vrstvou je datový model reprezentovaný třídou *FormDataModel*. Ten je specifický pro každý formulářový modul (vazba jedna ku jedné se třídou *FormModule*). Jeho funkcí je načítat a udržovat data pro všechny bloky formuláře a připravit je pro zobrazení.

Velký balík tříd je *FormObject*. Obsahuje třídy, které reprezentují formulářové objekty, viz kapitola 3.2. Na nejvyšší úrovni je *FormModule*, který má vazby na další formulářové objekty.

Důležitý je balík *Internal*. Tento balík tvoří vrstvu aplikační logiky interpreteru. Je v něm definováno rozhraní *InternalAPI*, které obsahuje zejména metody odpovídající build-in procedurám Oracle Forms, ale může obsahovat i další metody. Třída *InternalAPIProvider* je singleton a poskytuje instanci *InternalAPI*. Je tak možné získat instanci *InternalAPI* v každé třídě, aniž by do ní musela být explicitně předána. *InternalAPI* je implementováno třídou *InternalBusinessLogic*.

Další balík je *Webservice* a podle očekávání obsahuje rozhraní deklarující metody webových služeb interpreteru, které jsou volány z databáze, resp. bridge aplikace. Rozhraní se jmenuje *IOracleInterpreterWS* a je implementováno třídou *OracleInterpreterWS*.

Balík *Controls* obsahuje komponenty prezentační vrstvy interpreteru.



Obr. 16: *UML diagram tříd.*

6.5.3 Implementace problémů

6.5.3.1 Vytvoření formulářových objektů z XML

Formulářové objekty mají atributy odpovídající vlastnostem, které byly popsány v kap. 3.2. Např. *Item* má atributy *Height* a *Width*, *Block* má *QueryDataSourceName*, atd. Všechny potencionální atributy včetně datových typů přesně specifikuje soubor forms.xsd. Atribut, který mají všechny formulářové objekty, je atribut *Name*.

Načtení formulářového XML souboru a vytvoření objektů je implementováno v `InternalBusinessLogic.CreateForm()` použitím třídy `XmlTextReader`. Každý formulářový objekt má v konstruktoru parametr `XElement` a při vytváření objektu se rovnou provede inicializace daty z XML elementu.

Stejně jako je v XML elementech hierarchie (*FormModule* obsahuje *Blocky*, *Block* obsahuje *Itemy*), je hierarchie i v objektech, takže `FormModule` vytváří `Blocky` a `Block` zase `Itemy`.

Většina atributů není v XML povinných, takže je třeba před načtením otestovat, zda atribut existuje a pokud ne, použít implicitní hodnotu. Dále u atributů, které nejsou řetězcové, se musí provést konverze z řetězce na požadovaný typ. Obojí je ošetřeno ve třídě `XmlUtils`, která obsahuje metody `GetAttributeValue` pro návratový typ `string`, `int`, `double` a `bool`.

V XML se v některých atributech může vyskytnout omezení výčtem, např. typ *itemu*. Konverze ze `stringu` na `enum` je genericky řešena ve `XmlUtils.GetEnumValue()`.

```
public static T GetEnumValue<T>(string value, T defaultValue)
{
    T retVal;
    try
    {
        retVal = (T)Enum.Parse(typeof(T), value);
    }
    catch (ArgumentException)
    {
        retVal = defaultValue;
    }
    return retVal;
}
```

Kód 8: *Generická konverze výčtu.*

6.5.3.2 Rozvržení komponent v okně formuláře

Rozmístění komponent (*canvas*, *záložka*, *item*) ve formuláři je realizováno ve třídě `MdiWindow` a jejích metodách `AddCanvas()` a `AddControl()`.

- `AddCanvas(Canvas canvas)` umístí na formulář `Canvas` podle jeho souřadnic a pokud obsahuje záložky, je vytvořena komponenta `TabControl`.
- `AddControl(Control control, String canvasName, String tabName)` vloží grafickou komponentu control na cílový *canvas* nebo *záložku*. Vytvoření `Control`

objektu z formulářového objektu typu `Item` je implementováno v metodě `Item.CreateItemControls()`. Vytvoří se `Control` podle typu *itemu*.

6.5.3.3 Svázání GUI s daty

Pro svázání grafických komponent s daty neboli *data binding* má WPF velmi dobrou programovou podporu. Klíčové třídy jsou `DataSet`, `DataTable` a `BindingListCollectionView`. Zejména poslední třída je užitečná, protože obsahuje metody pro pohyb v záznamech (`MoveCurrentToFirst`, `MoveCurrentToPrevious`, `MoveCurrentToNext`, `MoveCurrentToLast`).

Z načteného `DataSetu`, resp. `DataTable` lze vytvořit `BindingListCollectionView` objekt a ten použít pro *data binding* grafických komponent. Je-li správně nastaven *data binding*, komponenty automaticky zobrazí data a především dojde-li ke změně dat v UI nebo `DataSetu`, změna se projeví i v opačném konci spojení.

Ve WPF se *data binding* u různých grafických komponent nastavuje k odlišným `DependencyProperty` vlastnostem, záleží na účelu. Aby bylo možné nastovovat svázání dat jednotně pro všechny komponenty, je v interpreteru vytvořeno rozhraní `IBindable` s metodou `SetBinding(ICollectionView view)`. Každá komponenta implementuje toto rozhraní a nastaví *data binding* podle sebe (příklad viz Kód 9).

```
public class FormTextBox : TextBox, IBindable
{
    private string columnName;

    public FormTextBox(string columnName)
    {
        this.columnName = columnName;
    }

    public void SetBinding(ICollectionView view)
    {
        Binding binding = new Binding();
        binding.Source = view;
        binding.Path = new PropertyPath(columnName);
        base.SetBinding(TextBox.TextProperty, binding);
    }
}
```

Kód 9: *Vytvoření Bindingu ve třídě FormTextBox.*

6.5.3.4 Transformace kódu triggeru

Transformace kódu triggeru tak, aby byl vykonatelný, je zásadní úkol interpreteru. Je implementován třídou `TriggerPreprocessor`. Trigger je po načtení z XML zpracován následující sekvencí kroků:

1. Odstranění HTML entity

2. Odstranění komentářů – blokových i řádkových

3. Transformace propert

   Property jsou uzavřeny do apostrofů a tím pádem nebudou při vykonání vyhodnoceny jako číselné konstanty, ale jako řetězce. Nalezení propert v kódu je řešeno metodou `Regex.Replace()`. Hledají se konstanty definované ve třídě `FormConstants`, poli `FORM_CONSTANTS`. Před a za nalezenou konstantu se vloží apostrof.

4. Transformace volání build-in procedur

   Při volání build-in procedur musí být transformováno s ohledem na fakt, že nyní jsou v databázi identické procedury uloženy v PL/SQL balíku oraforms, takže je třeba přidat před každý výskyt volání build-in procedury prefix „oraforms.“. Není potřeba žádná analýza kódu, postačuje opět použití `Regex.Replace()`.

5. Transformace přiřazení do formulářových proměnných

   Přiřazení do formulářové proměnné (začíná dvojtečkou) má obecně tvar „*:proměnná := výraz;*“. Při vykonání je třeba, aby se v databázi vyhodnotil výraz, ale výsledek aby byl přenes do interpreteru. Tento problém je řešen transformací na volání speciální procedury *write_form_var*, která v interpreteru zapíše výsledek výrazu. Výsledkem transformace je zápis „*oraforms.write_form_var(':proměnná', výraz);*“. Implementace se nachází v metodě `TriggerPreprocessor.TransformAssignmentStatements()`. Pro nalezení přiřazovacího příkazu je použit regulární výraz.

6. Transformace čtení formulářových proměnných

   Čtení formulářové proměnné je řešeno obdobně jako přiřazení. Při nalezení formulářové proměnné, která není v uvozovkách, je provedeno nahrazení na „*oraforms.read_form_var(':proměnná')*“.

Jako ukázku přikládám vzorek PL/SQL kódu před transformací (Kód 10) a po jeho transformaci `TriggerPreprocessor`em (Kód 11).

```
declare
  curform VARCHAR2(40);
  jmeno VARCHAR2(30);
begin
  :global.verze := '2.1';
  curform := :System.Current_Form;
  jmeno := GET_FORM_PROPERTY(curform,FORM_NAME);
  :toolbar.nazev := jmeno;

  SET_WINDOW_PROPERTY(FORMS_MDI_WINDOW,WINDOW_STATE,MAXIMIZE);
  SET_WINDOW_PROPERTY(FORMS_MDI_WINDOW,TITLE,'UIS');
end;
```

Kód 10: *PL/SQL kód před transformací.*

```
declare
  curform VARCHAR2(40);
  jmeno VARCHAR2(30);
begin
  ORAFORMS.write_form_var(':global.verze', '2.1');
  curform := ORAFORMS.read_form_var(':System.Current_Form');
  jmeno := ORAFORMS.GET_FORM_PROPERTY(curform,'FORM_NAME');
  ORAFORMS.write_form_var(':toolbar.nazev', jmeno);

  ORAFORMS.SET_WINDOW_PROPERTY('FORMS_MDI_WINDOW','WINDOW_STATE',
      'MAXIMIZE');
  ORAFORMS.SET_WINDOW_PROPERTY('FORMS_MDI_WINDOW','TITLE','UIS');
end;
```

Kód 11: *PL/SQL kód po transformaci.*

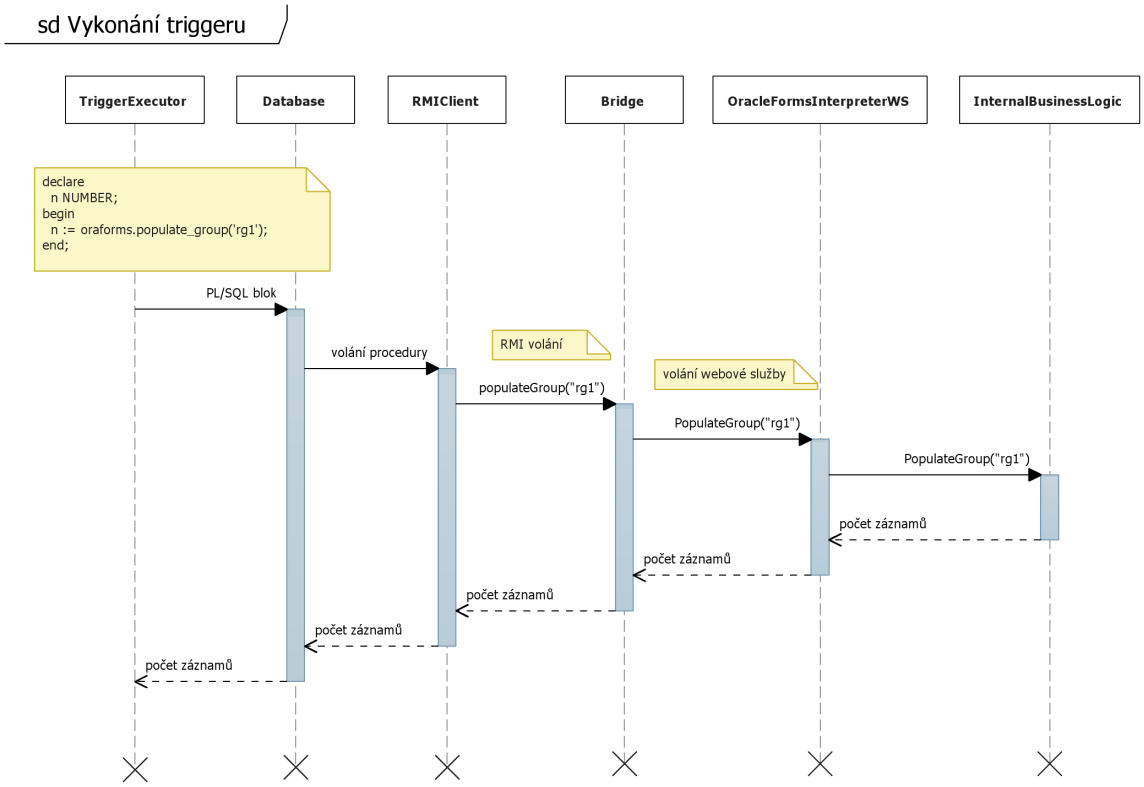
6.5.3.5 Vlákna

WPF aplikace implicitně vytváří vlákna podle své implementace. Důležité je např. vlákno Dispatcher, které zajišťuje odezvu uživatelského rozhraní. Nicméně vedle WPF vláken jsou v interpreteru explicitně definovaná i další vlákna.

První vlákno se stará o příjem volání webových služeb. Druhé vlákno slouží k sekvenčnímu spouštění PL/SQL bloků. PL/SQL bloky jsou před odesláním do databáze vloženy do fronty, aby nedošlo např. k paralelnímu vykonání. Vlákno z fronty postupně odebírá bloky a přes DAI je odesílá databáze. Další blok je spusštěn, až když je dokončen předchozí. Implementace se nachází ve třídě `TriggerExecutor`.

6.6 Komunikace mezi komponentami

Při vykonání triggeru, který obsahuje build-in procedury, dochází ke zpětnému volání z databáze zpět do interpreteru. UML diagram na Obr. 17 znázorňuje sekvenci volání, které vznikne v TriggerExecutor v interpreteru a pokračuje přes databázi a bridge zpět do aplikační logiky interpreteru. Jako příklad je použita build-in procedura *populate_group*.



Obr. 17: *Sekvenční diagram komunikace při vykonání triggeru.*

6.7 Testování

Testování aplikace bylo prováděno v průběhu vývoje. Za uspokojivě fungující lze považovat implementaci prezentační vrstvy interpreteru. Základní typy komponent (textové pole, check box, combo box, radio button, button) jsou rozpoznány a umístěny na formulář do správných souřadnic.

Problémy činí aplikační logika, nepodařilo se implementovat všechny existující build-in procedury, takže vykonání triggeru může selhat.

Pro testování byla vybrána množina formulářů tak, aby obsahovala co nejvíce různorodé vlastnosti formulářů. Byly zvoleny formuláře AN0010, PJ0062, PM0060 (co do velikost XML největší formulář), CI0060 a ZK0030.

7 Závěr

Tato práce se zabývala produktem Oracle Forms, jeho principy, vlastnostmi a souborovými formáty. Především byl analyzován XML formát trojice typů modulů. V kapitole 3.2 jsou potom rozebrány jednotlivé objekty Oracle Forms – jejich význam a vlastnosti. Nebylo možné z důvodu rozsahu uvést u každého typu plný výčet vlastností, které pro něj existují. Snažil jsem se vybrat ty podstatné. Kompletní přehled vlastností lze zjistit např. z XSD souboru pro XML formát.

Jedním z bodů zadání bylo nalézt na softwarovém trhu produkty, které by svojí funkcionalitou odpovídaly Oracle Forms. Bylo nalezeno celkem šest produktů, které jsou představeny v kapitole 4. Pět z nich buď poskytuje nějakou formu podpory pro migraci Oracle Forms formulářů na svoji platformu, nebo přímo používá původní souborové formáty. Všechny produkty vytváří webové aplikace (akorát *Forms2Net* může být nasazen i jako desktopová aplikace), což dokazuje jasný trend orientace na Web, SOA a Cloud. Za pozornost stojí *Forms2Net* (desktopové nasazení) a *Yo!Forms* (vlastní PL/SQL engine).

Realizačním úkolem bylo navrhnout a implementovat vlastní interpreter Oracle Forms (XML) formátů. Prezentační vrstvu formuláře nebylo tak těžké implementovat, ale zachování aplikační logiky bylo problematické. Bylo zvoleno řešení s integrací databáze, které rozhodně není ideální. Na druhou stranu když se podíváme na ostatní produkty, zachování aplikační logiky není samozřejmé.

Realizované řešení bohužel není v současném stavu v žádném případě stoprocentně funkční. Je navrženo tak, že jsou principiálně vyřešeny hlavní problémy, ale nejsou dosud implementovány všechny build-in procedury.

Přehled zkratek

AJAX	Asynchronous Javascript and XML
BLOB	Binary Large Object
CIV	Centrum informatizace a výpočetní techniky
CORBA	Common Object Request Broker Architecture
DAO	Data Access Object
GUI	Graphical User Interface
HTML	HyperText Markup Language
JAXB	Java API for XML Binding
JAX-WS	Java API for XML Web Services
LOV	List of Values
MDI	Multiple Document Interface
ODP.NET	Oracle Data Provider for .NET
OWASP	Open Web Application Security Project
PL/SQL	Procedural Language/Structured Query Language
SOAP	Simple Object Access Protocol
STAG	Studijní agenda
UI	User Interface
WCF	Windows Communication Foundation
WPF	Windows Presentation Foundation
WSDL	Web Services Description Language
XML	Extensible Markup Language
XSD	XML Schema Definition

Zdroje

[DEVE] DAS GUPTA, Pranab Kumar a GHOSH, Pranab. *Oracle Developer 2000 – Basics to Implementation*. New Delhi: Prentice-Hall of India Private Limited, 2008. ISBN 978-81-203-3510-3

[REFE] *Oracle Forms Developer – Form Builder Reference, Volume 1, Release 6i*, 2000 [online]. [cit. 25.4.2013]. Dostupné z: http://download.oracle.com/otn_hosted_doc/forms/forms/A73074_01.pdf

[TECH] LAKSHMAN, Bulusu. *Oracle Developer Forms Techniques*. Indianapolis: Sams Publishing, 2000. 264 s. ISBN 0-672-31846-6

[ARCH] *An Overview of Oracle Forms Server Architecture (An Oracle Technical White Paper)*, 2000 [online]. [cit. 25.4.2013]. Dostupné z: http://www.oracle.com/technetwork/developer-tools/forms/275632-133265.pdf

[PRE13] *Oracle 10g Forms, Lesson 13*, 2004 [online]. [cit. 5.5.2013]. Dostupné z: http://www.slideshare.net/KAMA3/oracle-10g-forms-lesson-13

[PRE14] *Oracle 10g Forms, Lesson 14*, 2004 [online]. [cit. 6.5.2013]. Dostupné z: http://www.slideshare.net/sudharsan2020/les14-3115841

[APEX] *Oracle Application Express, Application Builder User's Guide, Release 4.2*, 2013 [online]. [cit. 23.4.2013]. Dostupné z: http://docs.oracle.com/cd/E37097_01/doc/doc.42/e35125.pdf

[APEXM] *Oracle Application Express, Application Migration Guide, Release 4.2*, 2013 [online]. [cit. 23.4.2013]. Dostupné z: http://docs.oracle.com/cd/E37097_01/doc/doc.42/e35126.pdf

[MENU1] *Creating Oracle Forms Menus* [online]. [cit. 11.5.2013]. Dostupné z: http://www.oracle.com/webfolder/technetwork/tutorials/obe/forms/11g/formsmenuscreate/formsmenuscreate.htm

[MENU2] *Managing Oracle Forms Menu Modules* [online]. [cit. 11.5.2013]. Dostupné z: http://www.oracle.com/webfolder/technetwork/tutorials/obe/forms/11g/formsmenusmanage/formsmenusmanage.htm

[JAVDE] *Oracle Database Java Developer's Guide, 11g Release 1 (11.1)*, 2009 [online]. [cit. 14.5.2013]. Dostupné z: http://docs.oracle.com/cd/B28359_01/java.111/b31225.pdf

[RADE] RADECK, Kirk. *C# and Java: Comparing Programming Languages*, 2003 [online]. [cit. 13.5.2013]. Dostupné z: http://msdn.microsoft.com/en-us/library/ms836794.aspx

[ORAPL] *Java/ADF/.Net Solution Architecture White Paper*, 2011 [online]. [cit. 6.5.2013]. Dostupné z: http://www.oraplayer.com/resources/

[FEUER] FEUERSTEIN, Steven. PRIBYL, Bill. *Oracle PL/SQL Programming, Four Edition*, 2005. O'Reilly Media, Inc., Sebastopol, CA. ISBN 0-596-00977-1

[KYTE] KYTE, Thomas. *Expert One-on-One Oracle*, 2001. Apress, Berkeley, CA. ISBN 1-59059-243-3

[PRICE] PRICE, Jason. *C# - programování databází*, 2005. Grada, Praha, CZ. ISBN 80-247-0982-1

[ALBA] ALBAHARI, Joseph. *Threading in C#*, 2011 [online]. [cit. 21.3.2013]. Dostupné z: http://www.albahari.info/threading/threading.pdf

[WIKI] *Wikipedia: Oracle Forms* [online]. [cit. 26.4.2013]. Dostupné z: http://en.wikipedia.org/wiki/Oracle_Forms

[ORAFAQ] *Oracle FAQ's: Oracle Forms* [online]. [cit. 26.4.2013]. Dostupné z: http://www.orafaq.com/wiki/Oracle_Forms

[WCF] Web *Windows Communication Foundation*, [online]. Dostupné z: http://msdn.microsoft.com/en-us/library/dd560536.aspx