METODOLOGIE OOHDM, JAZYK LUA A TVORBA WEBOVÝCH APLIKACÍ

Martin Molhanec

ČVUT-FEL, Technická 2, 166 27 PRAHA 6, Dejvice, Česká republika
tel.: ++420 (2) 2435 2118, email: molhanec@fel.cvut.cz, web: http://martin.feld.cvut.cz/~mmm

Abstrakt

Metodologie OOHDM a jazyk Lua jsou základem vývojového prostředí OOHDM-Web vyvinutého na PUC-Rio Universitě v Brasilii. Metodologie OOHDM je speciálně vyvinutá objektově orientovaná metodologie pro analýzu hypermediálních a webových aplikací. Jazyk Lua je vyspělý objektově orientovaný skriptovací jazyk. Vývojové prostředí OOHDM-Web je vývojový systém postavený na spolupráci obou výše uvedených skutečností.

1. Úvod

Webová sídla a aplikace založené na webových technologií vytváří v současné době kdekdo. Bohužel, dle mého mínění, ve značné míře s využitím intuice vývojáře nežli s pomocí některé systematicky vybudované metodologie určené speciálně pro tvorbu webových sídel a aplikací. Tuto skutečnost autor tohoto příspěvku kritizoval ve svých článcích na této i jiných konferencí [2], [13].

Nejvíce využívané a také nejvíce známé jsou početné webové technologie. Autor tohoto příspěvku sám zejména na této konferenci publikoval několik příspěvků na toto téma [3], [4], [5]. Jedná se o všeobecně známé technologie pro tvorbu dynamicky generovaných sídel (CGI, ASP, JSP, PHP a mnohé další). Existuje také značné množství vývojových nástrojů, které podporují tvorbu webových sídel a aplikací s pomocí těchto technologií (FrontPage, HTML-Kit, Cold Fusion, a další).

Bohužel prakticky neznámé jsou analytické metodologie pro tvorbu webových sídel a aplikací. Informace o některých z nich autor tohoto článku publikoval například na této konferenci [2] a na konferenci Objekty [13]. Tento příspěvek bude v této tradici pokračovat.

V loňském roce jsem na konferenci *Objekty 2001* prezentoval objektově orientovanou metodologii **OOHDM** [13]. Na rozdíl od jiných analytických metodologií se autoři této metodologie rozhodli, že pro její efektivní využití při vývoji webových sídel a aplikací vytvoří i vhodné vývojové prostředí, které bude tuto metodologii přímo podporovat. K tomu využili objektově orientovaný skriptovací jazyk **Lua**, který byl na jejich pracovišti již za jiným účelem vytvořen a pro jejich záměry měl vhodné vlastnosti.

2. Jazyk Lua

Jazyk **Lua** byl vytvořen záměrně jako efektivní skriptovací jazyk určený zejména jako *embedded language* čili jazyk zabudovaný do jiných aplikací. Zjednodušeně řečeno jedná se jazyk, který umožňuje danou aplikaci, která je napsaná například v jazyce C, C++, Ada,

Modula, Pascal, Cobol či Fortran, programovat v jazyce jiném, pro daný účel vhodnějším (interpretovaném, jednodušším), nežli je vlastní jazyk, ve kterém je samotná aplikace napsaná. Typickým reprezentantem takovéhoto jazyku je například *Visual Basic for Applications* nebo *Visual Basic Scripting Edition* od firmy Microsoft nebo v současné době populární *Javascript*.

2.1 Datové typy

Jazyk **Lua** mimo obvyklých datových typů (vytvářených dynamicky podobně jako je tomu například v jazycích *Javascript* nebo *Visual Basic Scripting Edition*) *number (floats)* a *string* obsahuje typ *function* (opět podobně jako je tomu například v jazyku *Javascript*) a tři speciální typy *nil*, *userdata* a *table*.

Typ *nil* je typ s jedinou hodnotou určenou pro označení, že daná proměnná nemá žádnou hodnotu, respektive má právě hodnotu *nil*, která však nemá žádný rozumný význam. Tuto hodnotu má pochopitelně každá neinicializovaná proměnná.

Typ *userdata* obsahuje libovolná data z hostující aplikace (respektive programovacího jazyka ve kterém je aplikace naprogramována) a ve skutečnosti reprezentuje ukazatel typu *void* * v řeči jazyka C. Je určen především pro komunikaci mezi jazykem **Lua** a vlastní aplikací.

Typ *table* implementuje *asociativní pole* (opět si všimněte podobnosti například s jazykem Awk, Tcl nebo Perl).

Všimněme si nyní některých *vychytávek* jazyka **Lua**. Zcela ojedinělá (nicméně například jazyk matematického prostředí *Matlab* umí totéž) je možnost, aby funkce vrátila více hodnot.

```
Function Pokus (par1, par2)
     Local    Nasobeno  = par1 * par2
     Local    Deleno    = par1 / par2
     Return   Nasobeno, Deleno
End

X, Y = Pokus(20, 2)
```

Pokud se týká asociativních polí, jsou pro přístup k jejich prvkům dovoleny pochopitelně oba obvyklé přístupy pomocí operatorů **.name** a **[“name”]**, včetně zkrácené možnosti inicializace pole. Na rozdíl od ostatních programovacích jazyků jsou pole v jazyce **Lua** vždy implementována jako ukazatel na pole.

```
Pole = {}
Pole[”barva”] = ”zelena”
X = pole.barva
Y = pole[”barva”]
Jine_pole = {barva  = ”modra”}
```

2.2 Objektová orientace

Podpora pro práci s objekty je v jazyce **Lua** velice podobná jako například v jazyce *Javascript* nebo *WebL*. Práci s objekty podporuje jednak několik syntaktických *sladkostí* a mechanismus *fallback*. Pokusme se naznačit o co se zde jedná. Podpora tzv. *konstruktoru* spočívá v náhradě volání *name({par,…})* za kratší *name{par,…}*. Je patrné, že pomocí *konstruktoru* můžeme vytvářet *třídy (Classes)* podobně jako například v jazyce *Javascript* nebo *WebL*.

```
Window_1 = Window {x = 20, y = 30, barva = ”zelena”}
```

Pomocí mechanismu *fallback* je zde vtipným způsobem vytvořena možnost dědičnosti. V jazyce **Lua** je totiž možné odchytit uživatelsky událost, která způsobí výjimku. Tou může být ovšem také přístup k atributu (položce asociativního pole), který má být zděděn od předka. Potom může použití takto konstruované dědičnosti vypadat například takto.

```
Zam1 = zamestnanec {parent = osoba{RC=123456789,
                                   Vek=30,
                                   Jmeno=”John Novak”},
                    Telefon  = 123456,
                    Mistnost = ”450/B3”
       }
```

2.3 Jazyk Lua – shrnutí

Jak je vidět, jazyk **Lua** [14] obsahuje opravdu několik neobvyklých vlastností. Zejména *fallback* je mocný nástroj, jehož možnosti jsou takřka neomezené. Je zajímavé též poznamenat, za jakým účelem byl jazyk **Lua** původně vyvinut. Jazyk **Lua** byl totiž vyvinut jako skriptovací jazyk na podporu tzv. *scientific data entry* a *visualizing lithology profiles*. O poměrně velké míře rozšíření jazyka L**ua** hovoří seznam několika desítek projektů, které byly s jeho využitím realizovány [15]. Je zajímavé, že jedno z nejčastějších uplatnění jazyk L**ua** nalezl jako interní jazyk nejrůznějších her (*Baldur's Gate, Crazy Robot Ivan, Escape from Monkey Island* a mnohé další). Nicméně mezi další zajímavé využití patří například *nelineární řízení robotů, správa heterogenních sítí, rozšíření jazyku **Lua** pro podporu aktorů* a mnoho dalších využití zejména v oblasti webových technologií.

3. CGILua

CGILua je systém využívající jazyk **Lua** jako jazyk pro psaní CGI skriptů [16]. Je pochopitelné, že jazyk **Lua** je možné použít pro vytváření CGI skriptů podobně jako jiné programovací jazyky. Nicméně systém **CGILua** zahrnuje v sobě jednak speciální knihovny pro snadné vytváření obsahu HTML stránek a jednak možnost vytváření tzv. *HTML template* souborů, které v sobě zahrnují, jak HTML kód, tak program v jazyce **Lua,** podobně jako je tomu například v ASP, PHP a JSP technologiích.

3.1 Lua scripts

Lua skripty jsou soubory s koncovkou **.lua** obsahující výhradně kód v jazyce **Lua**. Pro snadnou práci systém **CGILua** poskytuje programátoru celou řadu funkcí, které mu usnadňují práci. Ukázka jednoduchého skriptu, který vytvoří HTML stránku s textem *Hello World* nebo *Olá Mundo* podle právě aktivního jazyka následuje.

```
cgilua.htmlheader()
write('<html>')

if cgi.language == 'english' then
  greeting = 'Hello World!'
elseif cgi.language == 'portuguese' then
  greeting = 'Olá Mundo!'
else
  greeting = '[unknown language]'
end

write('<head><title>'..greeting..'</title></head>')
write('<body>')
write('<b>'..greeting..'</b>')
write('</body>')
write('</html>')
```

3.2 HTML templates

HTML templates jsou soubory s koncovkou **.html** nebo **.htm,** které mimo vlastního kódu HTML obsahují kód v jazyce **Lua** a speciální značky (*special marking*). Celá koncepce je navržena tak, že **HTML templates** je možné editovat obvyklými HTML editory beze ztráty informace. Existují tři základní způsoby, jak jazyk **Lua** zahrnout do kódu HTML.

3.2.1 Expression fields

Podobně jako v ASP, JSP nebo PHP je možné, aby výraz v jazyce **Lua** umístěný mezi speciální omezovače (znaky **$|** a **|$**) byl při zpracování stránky na serveru nahrazen vypočtenou hodnotou.

```
Good Morning, <b>$|firstname|$</b>
<input type="text" name="cor" value="$|cgi.cor|$">
<a href="$|cgilua.mkurl("ajuda.html", cgi )|$">Ajuda</a>
```

3.2.2 Code fields

Celou souvislou sekvenci kódu v jazyce **Lua** je možné zahrnout do speciálně uvozených HTML komentářů (znaky **<!--$$** a **$$-->**). Výhoda tohoto řešení je, že normální editor HTML tento kód ignoruje.

```
<!--$$
      function SubscriptionCharge()
        dofile( 'charges.lua' )
        return value_subscription
      end
$$-->

<b><i>The value of the Subscription is:
$|SubscriptionCharge()|$</i></b>
```

3.2.3 Loop and If directives

Jedná se o speciální příkazy systému **Lua**, které jsou umístěny v HTML komentářích. Jejich účel je umožnit jednak vícenásobné opakování HTML kódu nebo jeho podmíněné generování. Tyto speciální příkazy mohou být do sebe vnořené. Ale nejlepší ukázkou jsou opět dva příklady.

```
Table of numbers 1 to 9:<br>

    <table border=1>
      <tr>
    <!--$$ LOOP start='i=1', test='i<10', action='i=i+1' $$-->
      <td>$|i|$</td>
    <!--$$ ENDLOOP $$-->
      </tr>
    </table>
```

```
<form ...>
    User name:
    <!--$$ IF test='cgi.editmode' $$-->
      <input type="text" name="user" value="$|cgi.user|$">
    <!--$$ ELSE $$-->
      $|cgi.user|$
    <!--$$ ENDIF $$-->
</form>
```

3.3 CGILua – shrnutí

Jak je z výše uvedených ukázek patrné, je systém **CGILua** poměrně jednoduchý, ale efektivní. Podmíněná a opakovaná tvorba HTML kódu je možná oproti systému ASP poněkud omezenější, ale na druhé straně neumožňuje vytváření těžko srozumitelného kódu, což lze hodnotit jako výhodu. Všechny proměnné předané http serveru (GET, POST) jsou v rámci systému **CGILua** dostupné v prostřednictvím pole **cgi**, například formou zápisu: **cgi.název_proměnné**.

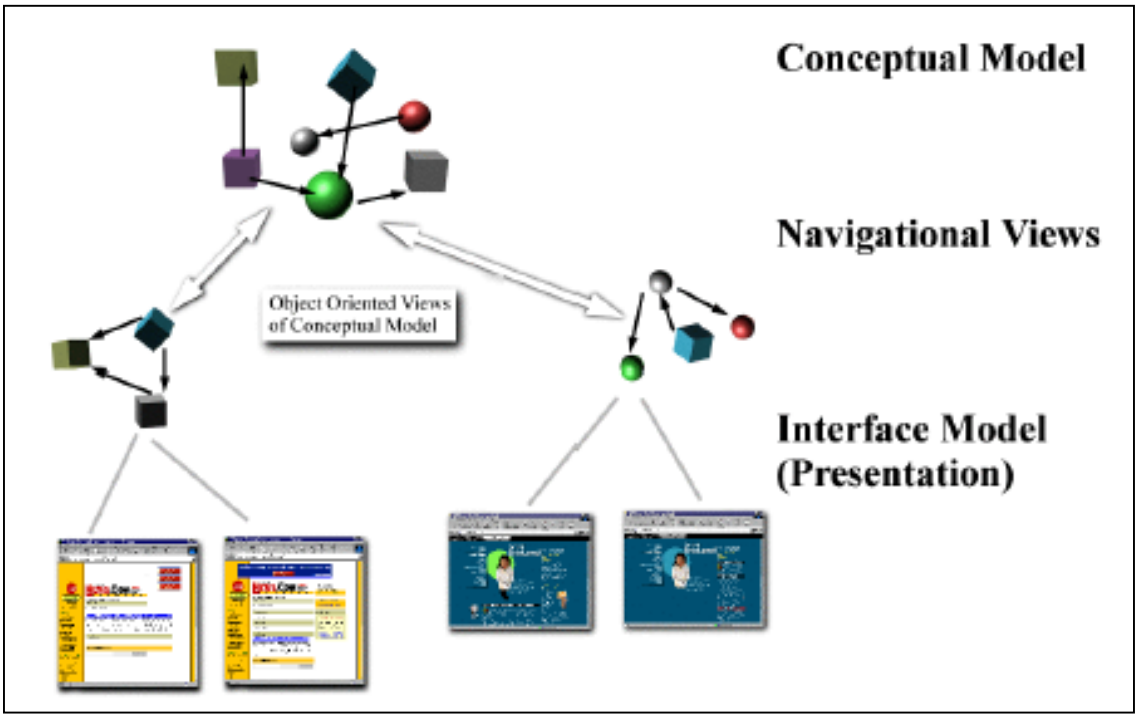
Mimo celé řady funkcí, které jsou součástí systému **CGILua** může vývojář využívat i bohaté množství knihoven již vytvořených pro jazyk **Lua**, například knihovnu **DBLua** umožňující přístup k **ODBC** databázím, což je pochopitelně bezpodmínečně nutné pro všechny webové aplikace, které svůj obsah vytvářejí na základě údajů uložených v databázi.

4. OOHDM

OOHDM čili *Object Oriented Hypermedia Design Method* byla popsána v mém příspěvku na konferenci Objekty 2001 [13]. Nebudu se proto pochopitelně vracet k podrobnému popisu této metodologie, takže zde pouze stručně. Celá metodologie je postavena na čtyřech základních krocích.

1. Konceptuální modelování *(Conceptual Modeling)*
2. Návrh navigace *(Navigational Design)*
3. Návrh abstraktního rozhraní *(Abstract Interface Design)*
4. Implementace *(Implementation)*

V každém z výše uvedených kroků se využívá jiný druh modelu našeho zájmu. Všechny modely jsou objektově orientované. Vztah mezi jednotlivými modely je možné vyjádřit následujícím obrázkem.



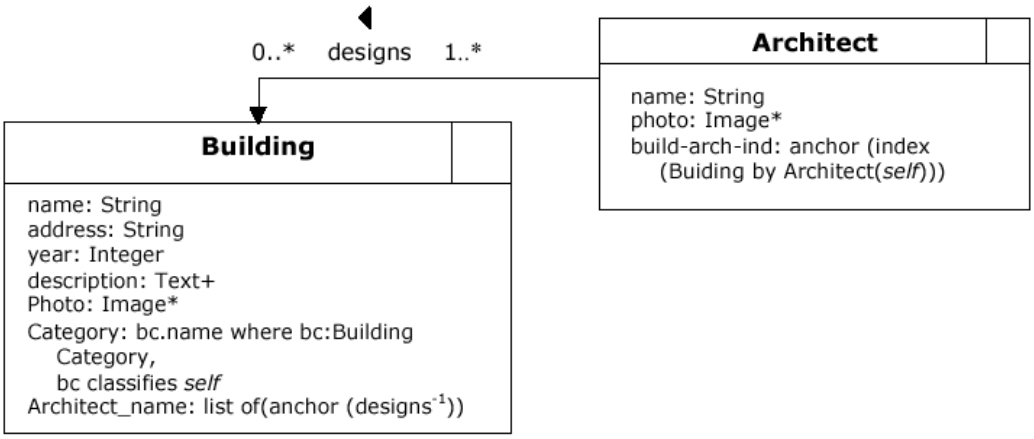
Obrázek 1: OOHDM – vztahy mezi objekty

4.1 Konceptuální model

Ve fázi konceptuálního modelování je naším cílem vytvořit konceptuální schéma reprezentující objekty (*objects*) a jejich vzájemné vztahy (*relationships*) a spolupráce (*collaborations*) v oblasti našeho zájmu (*target domain*). **OOHDM** je postaveno na konstruktech třída (*class*), vztah (*relationship*) a subsystém (*sub-system*).Třídy jsou popsány tak, jak je obvyklé v objektově orientovaných systémech, ačkoliv mohou obsahovat více-

typové (*multi-typed*) atributy, které tak mohou reprezentovat různý pohled na tu samou realitu. **OOHDM** používá notaci podobnou notaci **UML** a také využívá metodu „*Karty tříd a vztahů*“ (*Class and Relationship Cards*) pro dokumentační účely.

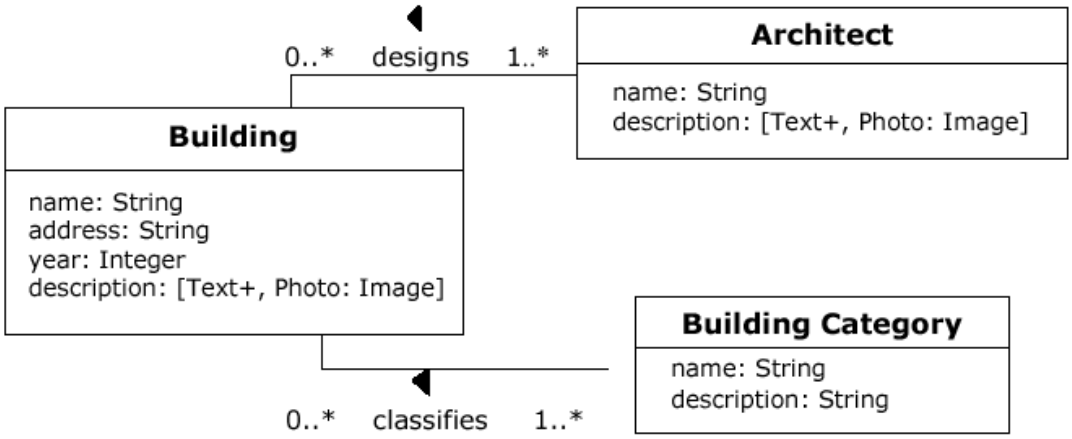
Konceptuální schéma se v OOHDM skládá z množiny tříd spojených vztahy. Objekty jsou instancemi tříd, a proto vztahy mezi třídami korespondují se vztahy mezi jednotlivými objekty. Třídy budou později v návrhu navigace použity pro odvození uzlů (*Nodes*) a vztahy pro odvození odkazů (*Links*). Diagram konceptuálního modelu webového sídla věnovaného architektuře následuje (příklad je převzat z publikace [17]). Webové sídlo obsahuje informace a stavbách a jejich architektech.



Obrázek 2: Konceptuální model

4.2 Navigační model

Návrh navigace je vyjádřen pomocí dvou schémat, *schématu navigačních tříd* a *schématu navigačních kontextů*. V **OOHDM** existuje sada předdefinovaných typů navigačních tříd, jsou to *uzly*, *odkazy* a *přístupové struktury*. Specifikace *navigačních transformací* popisuje dynamické chování aplikace. V **OOHDM** jsou používány varianty *Harelových Statecharts* nebo *Colemanových Objectcharts*. Pro přesnou specifikaci uzlů je používán dotazovací jazyk podobný SQL. Ale zanechme suchých popisů a podívejme se, jak vypadá navigační schéma (*diagram navigačních tříd*) pro náš již výše uvedený příklad webového sídla architektů.



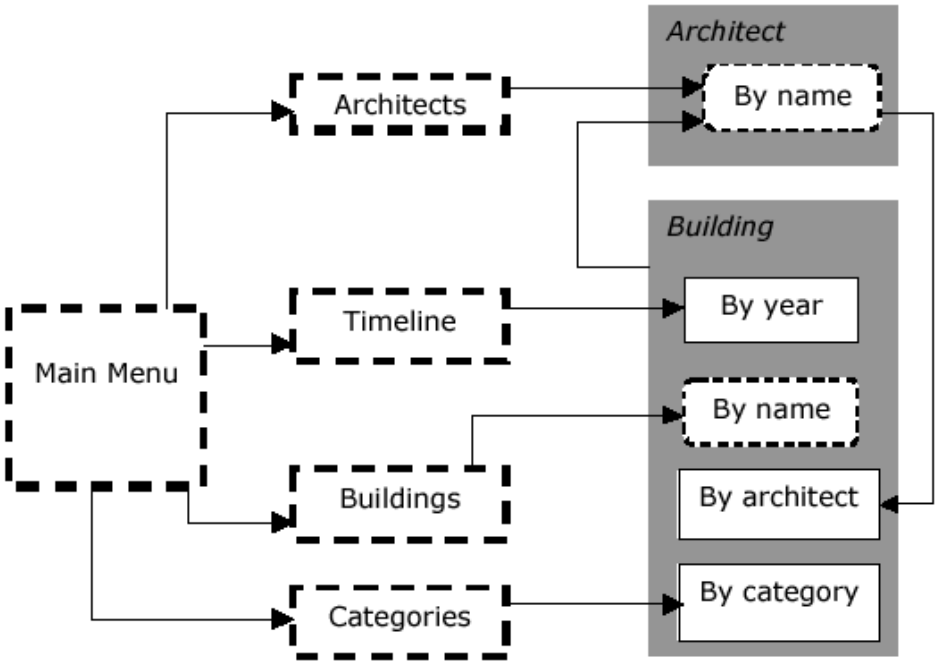
Obrázek 3: Schéma navigačních tříd

Jak je patrné, webové sídlo bude obsahovat dva **navigační uzly**. Jeden uzel je přiřazen stavbám (*Buildings*) a druhý architektům (*Architect*). Uživatel se může pohybovat z jednoho uzlu do druhého. Ale k tomu, abychom věděli jakým způsobem, je nutné ještě vytvořit *schéma navigačních kontextu.*

Navigační kontext je množina uzlů, odkazů, tříd kontextů a dalších (*vnořených*) navigačních kontextů. V **OOHDM** se běžně používá následujících pět typů kontextů.

Simple class based	Splňují objekty, které jsou téže třídy C a jsou vybrány prostřednictvím atributu **P**, který splňuje danou podmínku. Například „*all Stories*“ pokud je **P** vždy pravdivé.
Class based group	Je množina kontextů, které jsou typu *Simple class based*. Například „*Stories by type*“ je skupina kontextů, kde každý jednotlivý kontext je *Story* určitého typu.
Link based	Objekty v tomto kontextu jsou stejné třídy a jsou vybrány prostřednictvím vztahu **1:M**. Například „*all Stories by Bob Woodward*“.
Link based group	Je množina kontextů, kde každý z nich je *Link based*. Například „*Stories by Author*“.
Enumarated	V tomto případě jsou jednotlivé elementy kontextu jmenovitě vyjmenovány.

K výše uvedeným skupinám kontextům je možné dále přiřadit *dynamické kontexty*, například *historii* nebo *nákupní košík*. V obou případech se jedná o tzv. *Enumarated Dynamic Contexts*. Pro objasnění právě uvedených pojmů si opět ukážeme schéma navigačního kontextu našeho webového sídla architektů.



Obrázek 4: Schéma navigačního kontextu

Na výše uvedeném *schématu navigačního kontextu* jsou vidět některé typické konstrukty v těchto schématech užívané. Například *indexy*, které jsou znázorněny obdélníkem z přerušované čáry a *kontexty*, které jsou znázorněny obdélníky na pravé straně obrázku. Všechny grafické prvky diagramu navigačního kontextu mají svůj pevný význam, například použití plné či přerušované čáry ve znázornění kontextů, atp. Vzhledem k záměru tohoto příspěvku je však zde nebudu popisovat.

4.3 Abstract Interface Design

Jakmile byla definována navigační struktura aplikace musíme definovat aspekty uživatelského rozhraní. To znamená, že musíme definovat způsob jakým se jednotlivé prvky navigace budou uživateli zobrazovat. Oddělení navigace a rozhraní umožňuje, aby se táž navigace zobrazovala různým uživatelům různým způsobem. V praxi jsme zde však často omezeni nutností přizpůsobit vzhled své aplikace vývojovému prostředí, které bohužel mnoho důležitých aspektů správného návrhu rozhraní aplikace ignoruje. V **OOHDM** se používá pojem *Abstract Data View* (*ADV*) pro popis uživatelského rozhraní. ADV jsou objekty, které mají svůj stav a rozhraní a jsou abstraktní, protože nepopisují svojí implementaci! ADV také popisují svoji organizaci a chování, ale skutečné fyzické zobrazení je definováno a popsáno až ve fázi implementace. Jednotlivé ADV je možné skládat z ADV nižší úrovně a také je možné pro skládání využívat hierarchie. ADV nám umožňují vyjádřit následující skutečnosti.

- Způsob, jak jsou jednotlivé prvky rozhraní strukturovány pomocí *agregace* a *hierarchie*.

- Způsob, jak jsou staticky spojeny s navigačními objekty. V OOHDM se zde využívají tzv. *Configuration Diagrams*.
- Jak se chovají na vnější události, například klik nebo dvojklik myší, atp. V OOHDM se využívají tzv. *ADV-Charts* nebo *Petri-Net* like notation.

Určitou nevýhodou současné technologie s využitím realizace webového sídla nástrojem **OOHDM-Web** je skutečnost, že neexistuje automatizovaný způsob, jak převést například *ADV-Charts* do skutečného návrhu HTML stránky (*HTML template*), který je v systému **OOHDM-Web** využíván.

4.4 Implementace

Implementace se skládá ze tří základních činností, které budou vzápětí stručně popsány. Je dobré se však ještě zmínit o faktu, že **OOHDM** nevyžaduje nutně pro svoji implementaci objektově orientované prostředí, ač je to objektově orientovaná metodologie.

4.4.1 Mapování informačních položek

Informační položky, které se zobrazují, mohou být uloženy v souborech nebo databázích, dnes nejčastěji relačních. Proto se musí nejprve provést mapování objektově orientovaného modelu do modelu relačního. Autor tohoto příspěvku se sám touto problematikou dříve zabýval a některé poznatky publikoval na konferencích *Programování* [10, 11], *Datasem* [8, 9], *Objekty* [6] a *Tvorba software* [7]. Při mapování objektů navigačního modelu se s výhodou dá využít možnosti tzv. *view*, které moderní relační databáze poskytují. Je také nutné ve většině případů do databáze uložit objekty, které umožňují dekoraci navigačních objektů a různé chování objektů uživatelského rozhraní, například barva, typ, stav atp.

4.4.2 Implementace kontextu

Implementování kontextu vyžaduje zachování *stavové informace*. Protože však protokol http je *bezestavový*, je nutné využít všech známých prostředků pro uchování stavu (*kontextu*) naší aplikace, jako jsou například *cookies*.

4.4.3 Implementace rozhraní

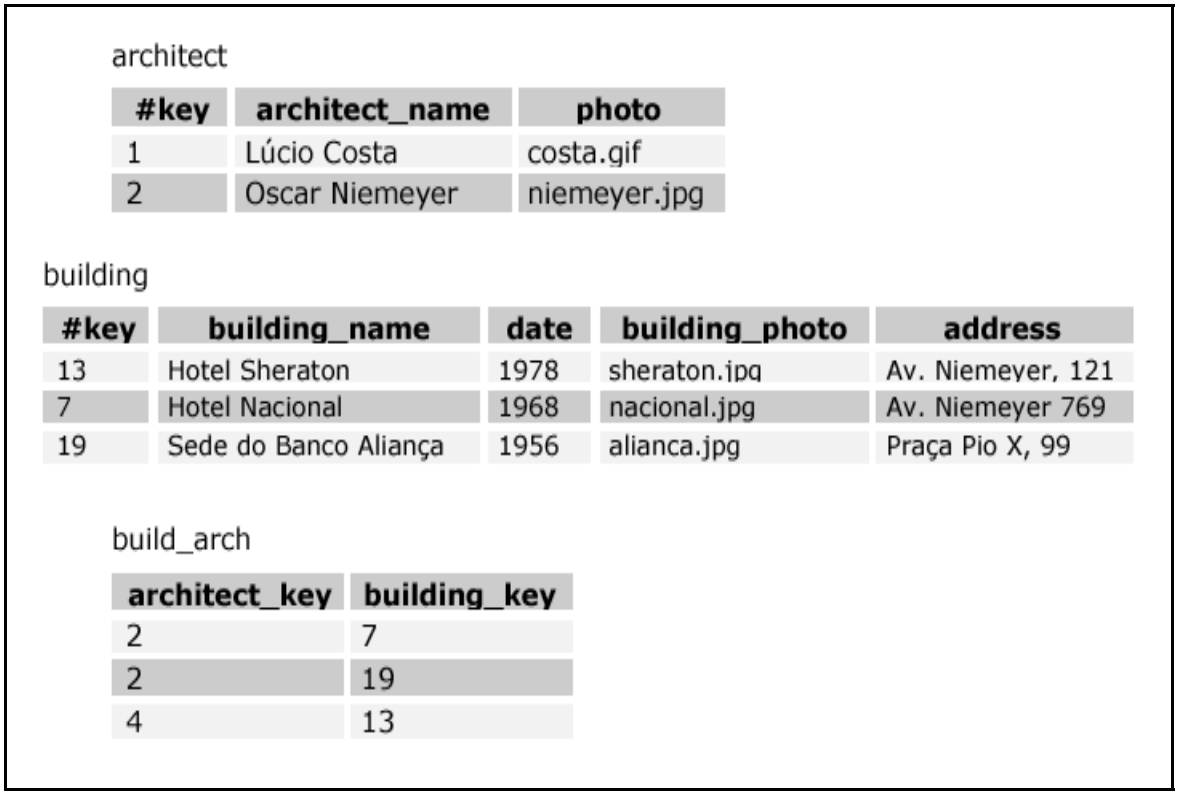
Vzhledem k tomu, že rozhraní bude ve většině případů nutné generovat dynamicky, protože bude zobrazovat proměnlivý obsah, je nutné využívat různé technologie pro *generování dynamických stránek*. Autor tohoto příspěvku publikoval přehled různých těchto technologií ve svých příspěvcích na konferencích *Tvorba software* [3, 4, 5]. Další možností mimo obecně použitelné technologie, jaké jsou například **ASP**, **PHP**, **JSP**, aj., je využití systémů postavených na tvorbě tzv. *templates*, jako je například *Cold Fusion* firmy Allaire nebo *StoryServer* firmy Vignette a další. Autoři OOHDM vytvořili systém **OOHDM-Web** postavený nad skriptovacím jazykem ***Lua***. Jeho výhoda spočívá především v tom, že systém **OOHDM-Web** přímo rozumí metodologii **OOHDM**.

5. OOHDM-Web

Jak bylo již zmíněno v kapitole předešlé je systém **OOHDM-Web [17], [18]** vytvořen speciálně pro podporu metodologie **OOHDM** autory a spoluautory této metodologie. Tento systém využívá systém **CGILua** postavený na skriptovacím jazyku **Lua**. Ukažme si nyní, jak

pomocí systému **OOHDM-Web** realizovat naše webové sídlo architektů. Příklad je opět přebrán z [17].

První krok je vytvoření databázových tabulek, které odpovídají konceptuálnímu diagramu. Jedná se o tabulky *architect*, *building* a *build_arch* (která je tzv. *vazební tabulkou*).



architect

#key	architect_name	photo
1	Lúcio Costa	costa.gif
2	Oscar Niemeyer	niemeyer.jpg

building

#key	building_name	date	building_photo	address
13	Hotel Sheraton	1978	sheraton.jpg	Av. Niemeyer, 121
7	Hotel Nacional	1968	nacional.jpg	Av. Niemeyer 769
19	Sede do Banco Aliança	1956	alianca.jpg	Praça Pio X, 99

build_arch

architect_key	building_key
2	7
2	19
4	13

Obrázek 5: Tabulky konceptuálních tříd

Pro další pochopení našeho příkladu je nutné vědět, že systém **OOHDM-Web** v sobě obsahuje definice 6ti tabulek, které popisují všechny kontexty v našem systému obsažené. První je tabulka se jménem *context*, která má dva sloupečky. V sloupečku prvém je *název* kontextu a ve sloupečku druhém jeho *typ*. Dalších 5 tabulek obsahuje pro jednotlivé typy kontextů dle kapitoly 0 další nezbytné údaje. Pro náš fiktivní případ může tabulka *context* mít následující obsah.

context

#ctx_name	type
building_alpha	1
building_by_year	2
building_architect	5

where:

ctx_name	is the name that identifies the context, given by the user
type	identifies the kind of context. The possible values are: 1 for simple class derived; 2 for class derived groups; 3 for arbitrary; 4 for simple link derived; 5 for link derived groups.

Obrázek 6: Tabulka kontextů

A jako další příklad si ukážeme obsah tabulky, která obsahuje další informace potřebné pro úplnou definici kontextu *building by year*.

Field	Contents
ctx_name	Is the context name
class_name	class to which the elements belong
class_atr	Attribute of class 'class_name' whose value is used to select the elements to include in the context
order	Attribute of class 'class_name' used to sort the elements in this context
target_page	HTML template used to present elements in this context
nav_type	Specifies the possible navigation beween elements in this context. Possible values are: 'sequential', 'circular', 'free' or 'index'.

(a)

type2

#ctx_name	class_ name	class_ atr	order	target_ page	nav_ type
building_by_year	building	date	building_name	build_yr.html	sequential

(b)

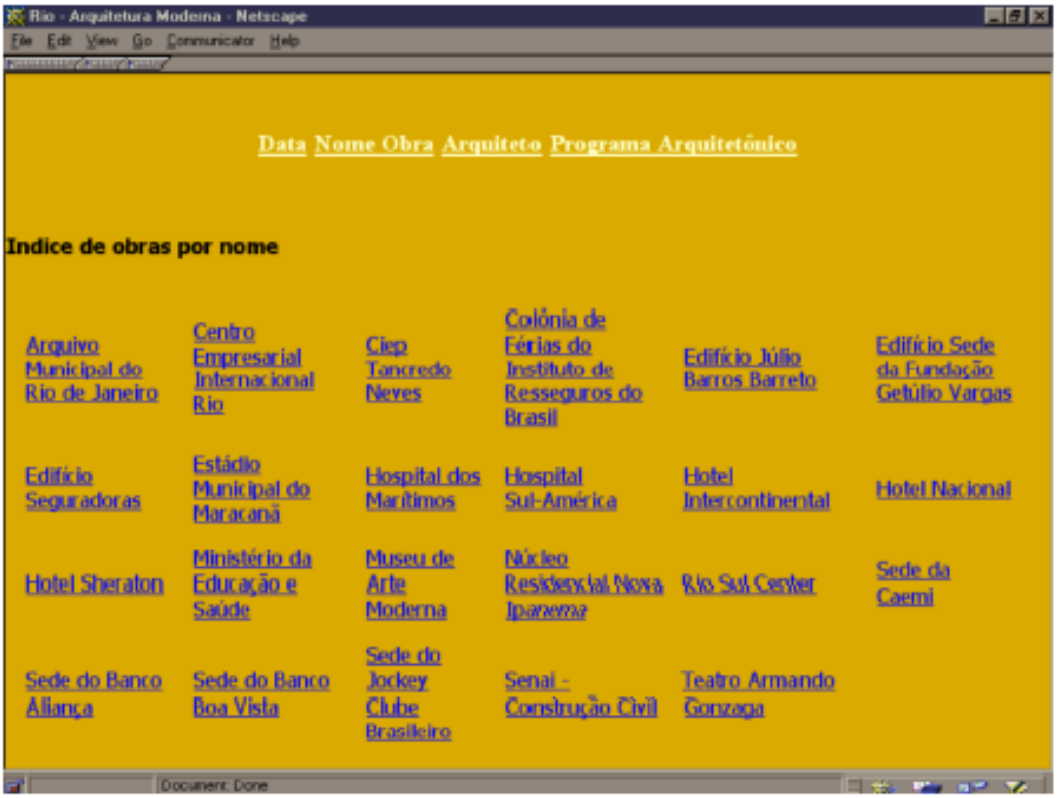
Obrázek 7: Tabulka dalších informací pro kontext *building by year*.

Informace obsažené v kontextových tabulkách jsou potřebné proto, aby speciální programové funkce systému **OOHDM-Web,** postaveném na systému **CGILua** a napsané v jazyce **Lua,** věděly jakým způsobem generovat příslušné *dynamické HTML* stránky webového sídla.

Jednou z těchto speciálních funkcí je například funkce *Index*. Pokud například chceme vygenerovat HTML stránku, která bude obsahovat seznam odkazů na stavby, které budou setříděny na stránce horizontálně dle abecedy, stačí do HTML kódu uvést volání této funkce. Způsob, jak v HTML stránce volat funkci jazyka **Lua** je vysvětlen v kapitole 3.2.1.

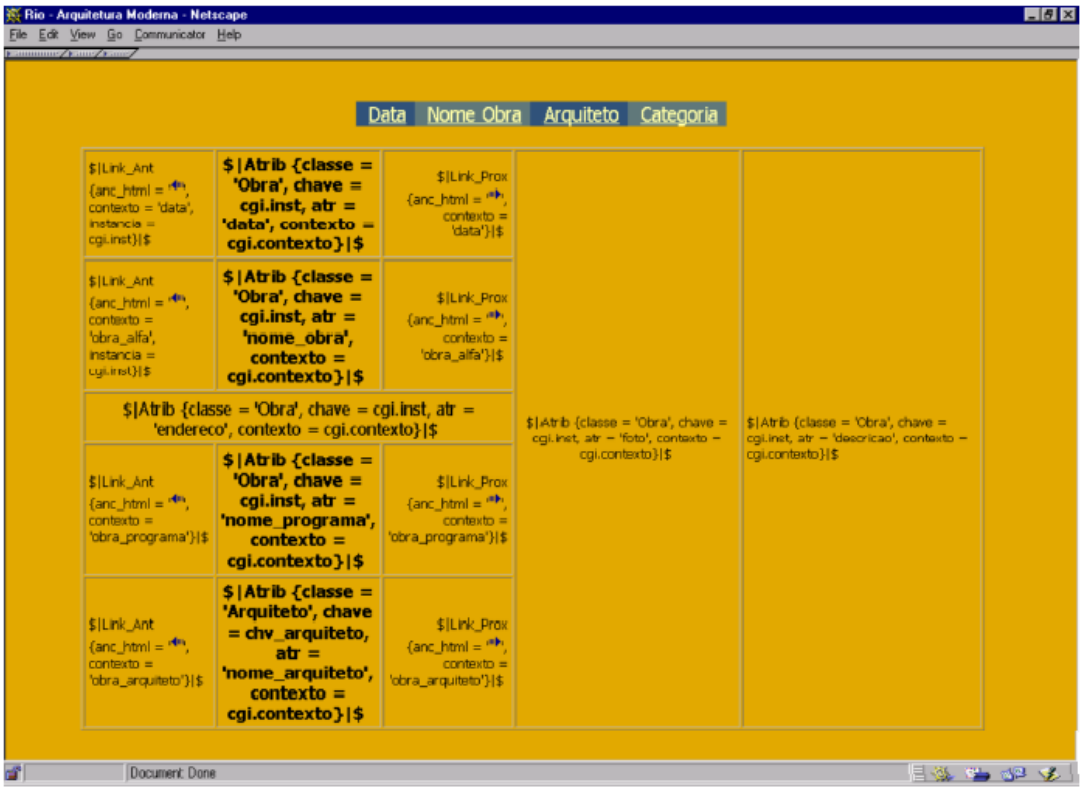
```
Index {context  = "build_alpha",
       anchor   = "building_name",
       function = 'Horizontal_Tab(
                     col = 6,
                     par_table  = "align-center cellspacing = 12",
                     par_cell   = "<center>"
                  )'
}
```

Výsledek volání této funkce bude následující.

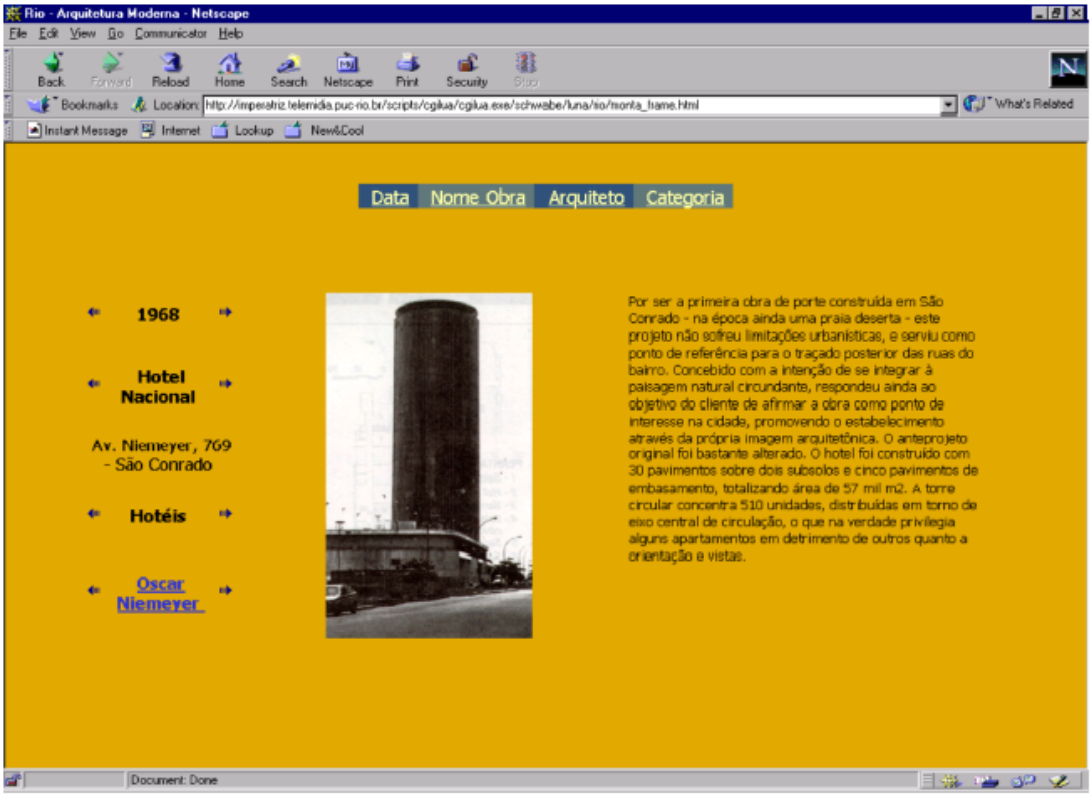


Obrázek 8: Stránka odkazů vygenerovaná funkcí *Index*

Dalším příkladem budiž komplexní návrh stránky obsahující údaje o jednotlivých stavbách včetně odkazů dle jednotlivých kontextů a její výsledek, jak jej uvidí uživatel.



Obrázek 9: Komplexní návrh stránky



Obrázek 10: Výsledek, jak jej vidí uživatel

6. Závěr

Metodologie **OOHDM** je pouze jednou z mnoha metodologií, které existují pro návrh webových sídel. Bohužel v České republice nejsou znalosti, nejen této, ale i ostatních metodologií, prakticky žádné, aspoň co mohu soudit z četby odborného tisku, webových stránek zabývajících se vývojem webových sídel, stránek různých kursů na školách a příspěvků na různých konferencí. Pokud jsem na omylu, budu jenom rád.

Řešení, které nabízejí autoři metodologie **OOHDM**, skriptovacího jazyka **Lua**, systému **CGILua** a vývojového prostředí **OOHDM-Web** jsou veskrze moderní. Skutečnost, že nástroj na vývoj webových aplikací je přímo napojen na jeho analýzu je tím nejpodstatnějším přínosem tohoto systému. Musím se totiž jen usmívat nad autory vývojových nástrojů většiny firem, kteří doposud nepochopili, že návrh programového díla započíná jeho analýzou! Jak může fungovat návrhář databáze nebo uživatelského prostředí, když systém neobsahuje informace z předcházející fáze analýzy? Když neví nic o vztazích mezi entitami konceptuálního modelu? Dle mého názoru jsou takové nástroje nedomyšlené!

Avšak řešení nabízené autory **OOHDM-Web** se snaží nastolit správnou cestu.
Cestu od analýzy k implementaci!

Literatura:

1. Daniel Schwabe and Gustavo Rossi, "An Object Oriented Approach to Web-Based Application Design", Theory and Practice of Object Systems 4(4), 1998. Wiley and Sons, New York, ISSN 1074-3224, revised text on http://www-di.inf.puc-rio.br\~schwabe\papers\TAPOSRevised.pdf
2. Martin Molhanec, "Tvorba webových sídel, jako inženýrský úkol ", Tvorba software 2001, Ostrava
3. Martin Molhanec, "Novinky intranetu", Tvorba software 2000, Ostrava
4. Martin Molhanec, „Novinky v technologiích intranetu", Tvorba software 1999, Ostrava
5. Martin Molhanec, „Technologie tvorby intranetových aplikací", Tvorba software 1998, Ostrava
6. Martin Molhanec, „Souvislosti mezi objektově orientovanými a relačními metodologiemi návrhu programů", Objekty 1998, Praha
7. Martin Molhanec, „Algoritmus převodu schématu objektově orientovaného modelu do schématu entitně-relačního modelu", Tvorba software 1996, Ostrava
8. Martin Molhanec, „Algoritmus generování příkazů definujících strukturu relačního databázového systému z objektově orientovaného datového slovníku", Datasem 1996, Brno
9. Martin Molhanec, „Realizace datového slovníku objektově orientovaného modelu", Datasem 1994, Brno
10. Martin Molhanec, „Některé souvislosti mezi převodem objektově orientovaného modelu do modelu entitně-relačního", Programování 1994, Ostrava
11. Martin Molhanec, „Souvislosti mezi objektově orientovanými a relačními metodologiemi návrhu programů", Programování 1993, Ostrava
12. http://www.telemidia.puc-rio.br/oohdm/oohdm.html, stránky věnované metodologii OOHDM
13. Martin Molhanec, „OOHDM – Object Oriented Hypermedia Design Method", Objekty 2001, Praha

14. Roberto Ierusalimschy, Luiz Henrique de Figueiredo, Waldemar Celes Filho, “Lua - an extensible extension language”, Software: Practice & Experience 26 #6 (1996) 635-652. Copyright © 1996 John Wiley & Sons, Ltd., http://www.lua.org/spe.html
15. http://www.lua.org/uses.html, Lua – list of projects
16. http://www.tecgraf.puc-rio.br/cgilua/, CGILua – What is CGILua?
17. Daniel Schwabe, Rita de Almeida Pontes and Isabela Moura, „OOHDM-Web: An Environment for Implementation of Hypermedia Applications in the WWW“, SigWeb
18. Daniel Schwabe, Rita de Almeida Pontes, „OOHDM-WEB: Rapid Prototyping of Hypermedia Applications in the WWW“, PUC-Rio Inf. MCC-08/98, March, 1998