VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

# KONVERZE ASP NA ASPX

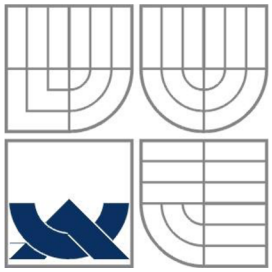
DIPLOMOVÁ PRÁCE

MASTER'S THESIS

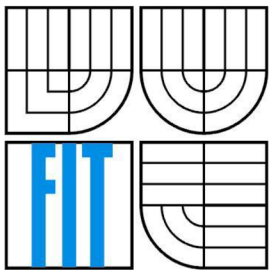
AUTOR PRÁCE
AUTHOR

Bc. Jan Vilímek

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

# KONVERZE ASP NA ASPX

TRANSLATION OF ASP INTO ASP.NET

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JAN VILÍMEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. TOMÁŠ KAŠPÁREK

BRNO 2007

# Zadání diplomové práce

**Řešitel:**

**Vilímek Jan, Bc.**

**Obor:**

Informační systémy

**Téma:**

Konverze ASP na ASPX

**Kategorie:**

Překladače

**Vedoucí:**

**Kašpárek Tomáš, Ing.**, CVT FIT VUT

**Oponent:**

**Ryšavý Ondřej, Ing., Ph.D.**, UIFS FIT VUT

**Zadání:**

1. Seznamte se s platformami APS/VBScript a ASP.NET (C#). Nastudujte problematiku jazyků, gramatik a překladačů.
2. Zvolte vhodné nástroje pro vývoj překladače.
3. Navrhněte nástroj pro automatizovaný převod aplikací v ASP-VBScript do ASP.NET (C#), který bude sloužit k převodu starších aplikací na novou platformu. Cílem je převod co nejvíce zautomatizovat a zjednodušit. Převod aplikace musí být proveditelný opakovaně.
4. Proveďte implementaci navrženého nástroje, zhodnoťte dosažené výsledky na reálných aplikacích a diskutujte další možný vývoj nástroje.

**Implementační jazyk:**

ASP/VBScript, C#

**Komerční software:**

Microsoft Visual Studio .NET

**Literatura:**

- jazyk C# (**http://msdn.microsoft.com/vcsharp/)**
- Microsoft MSDN (**http://msdn.microsoft.com/mobility/** )

**Datum zadání:**

28. února 2006

**Datum odevzdání:**

22. května 2007

# Licenční smlouva

Licenční smlouva je uložena v archivu Fakulty informačních technologií Vysokého učení technického v Brně.

Výňatek z licenční smlouvy - článek 2 - **udělení licenčního oprávnění**:

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizovaní výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti 1 rok po uzavření této smlouvy (z důvodu utajení v něm obsažených informací).
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

# Abstrakt

Cílem práce je implementovat aplikaci pro konverzi ASP stránek napsaných v jazyce VBScript na ASPX stránky a jazyk C#. Aplikace je vyvíjena na platformě .NET. Konverze stránek bude probíhat automaticky, cílový kód by se měl obejít bez dalšího zásahu programátora. První část práce uvádí do problematiky, poskytuje přehled současných řešení. Další část je analýza problému a návrh řešení. Zejména se pak práce zabývá konverzí VBScript gramatiky, konkrétními problémy a jejich řešením.



# Klíčová slova

ASP, ASPX, .NET, gramatiky, konverze, C#, VBScript, webová aplikace, syntaktická analýza, lexikální analýza, Microsoft Migration Asistant, NetCoole Asp2Aspx, ANTLR, GOLD Parser.



# Abstract

The goal of this dissertation is to implement an application for ASP to ASPX conversion. The ASP pages should be written in the VBScript language, the target language for ASPX will be C#. The application is developed on the .NET platform. The conversion process should be automatic. There should be no need to alter the converted files by a programmer. The first part of this dissertation introduces the whole problematic. It shows also current solutions. The next part is the analysis and the design of the application itself. The main part of this dissertation is the VBScript grammar conversion, problems while conversion and its solving.



# Key Words

ASP, ASPX, .NET, grammars, conversion, C#, VBScript, web application, syntactic analysis, lexical analysis, Microsoft Migration Asistant, NetCoole Asp2Aspx, ANTLR, GOLD Parse.



# Citace

Vilímek Jan: Konverze ASP na ASPX. Brno, 2007, diplomová práce, FIT VUT v Brně.

# Konverze ASP na ASPX

## Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Tomáše Kašpárka.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....

Jan Vilímek

V Brně dne 22. 5. 2007

## Poděkování

Děkuji vedoucímu práce Ing. Tomáši Kašpárkovi za odborné vedení. Chtěl bych poděkovat firmě GETMORE za poskytnutí zázemí (notebook, SW vybavení) a všem, kteří se jakýmkoliv způsobem podíleli na výsledné podobě této práce. Zejména bych chtěl poděkovat Ing. Martinu Košutovi za úpravu a testování konverzních pravidel balíčku asp2aspx.

# Obsah

# Úvod (1)

Tato diplomová práce se zabývá problematikou konverze z jednoho programovacího jazyka do druhého. Konkrétně jsem se zaměřil na konverzi skriptovacího jazyka VBScript, platformy ASP, do objektového jazyka C#, platformy .NET. Na základě zjištěných závěrů byl navržen a implementován nástroj, který tuto konverzi automaticky provádí.

Cílem práce bylo navrhnout a implementovat takový nástroj, který by za minimálního přispění uživatele umožnil (i **opakovanou**) konverzi z jednoho jazyka do druhého (ASP VBScript na C#.NET).

Práce vznikla na základě požadavků a zadání firmy Getmore, s.r.o. Byly předány aktuální zdrojové kódy aplikace GetmoreSystem. Tyto zdrojové kódy pak byly použity pro testování migračních nástrojů, konverzních pravidel, apod. Aplikace samotná obsahuje cca 240 zdrojových ASP souborů a cca 46 000 řádků kódu (včetně HTML).

## 1.1 Stručný přehled pojmů

Při výkladu pojmů předpokládám, že význam některých triviálních výrazů (HTML, webový server, aplikace…) je čtenáři znám. Pokud ne, odkazuji jej na internetové stránky **http://www.google.com**, případně **http://computing-dictionary.thefreedictionary.com/** a internet obecně.

### 1.1.1 VBScript

Skriptovací jazyk. Celý název by měl být Visual Basic Script, ale obecně se používá spíše zkratka VBScript. Gramatika a syntaxe je podobná jazyku Visual Basic. Na rozdíl od něj však nemá typové proměnné - jsou deklarované bez specifikace typu. Typ proměnné uchovává až interpretační prostředí při provádění skriptu v závislosti na aktuálním přiřazení hodnoty do proměnné. Na rozdíl od Visual Basic se aplikace VBScript nekompilují, ale jsou přímo interpretovány.

Pro oddělování příkazů slouží v tomto jazyce odřádkování. Každý příkaz (až na výjimky) musí tedy být na samostatném řádku. Namísto znaku nového řádku lze však použít symbol „:“. Příklad1 i příklad2 jsou ekvivalentním zápisem toho samého případu:

```
Příklad1:
If Not IsNumber(varA) Then
       Response.Write „Nezadal/a jste číslo!“
End If

Příklad2
If Not IsNumber(varA) Then : Response.Write „Nezadal/a jste číslo!“ : End If
```

### 1.1.2 ASP

Neboli také **A**ctive **S**erver **P**ages. Je to technologie, umožňující na webovém serveru vytvořit stránky s dynamicky vytvářeným obsahem (HTML se vytváří až při jejich odvolání uživatelem na základě parametrů, ale i konkrétního stavu webového serveru). ASP stránky obsahují HTML a skriptovací značky. Kód uvnitř skriptovacích značek se interpretuje jako program.

Více informací o ASP viz následující citace:

> *A Web server technology from Microsoft that allows for the creation of dynamic, interactive sessions with the user. An ASP is a Web page that contains HTML and embedded programming code written in VBScript or Jscript. It was introduced with Version 3.0 of Microsoft's Internet Information Server (IIS). When IIS encounters an ASP page requested by the browser, it executes the embedded program. ASPs are Microsoft's alternative to CGI scripts and JavaServer Pages (JSPs), which allow Web pages to interact with databases and other programs. Third- party products add ASP capability to non-Microsoft Web servers. The Active Server Page technology is an ISAPI program and ASP documents use an .ASP extension.*

***Citace z The Free Dictionary, url http://computing-dictionary.thefreedictionary.com/ASP***

### 1.1.3 ASP.NET

Neboli **A**ctive **S**erver **P**ages for **.NET**. Stejně jako ASP umožňuje vytváření dynamických stránek. Na rozdíl od ASP jsou však stránky psané kromě HTML i v některém z .NET jazyků (C#, VB.NET, …) a nejsou interpretované, ale kompilované (do tzv. IL kódu). Nejsou zpětně kompatibilní s ASP.

### 1.1.4 C#

Objektově orientovaný jazyk, vycházející z gramatiky/syntaxe jazyků C++, java i Visual Basic (např. příkaz foreach). Vyvinutý společností Microsoft, standardizován ECMA[1]. Jeden z primárních jazyků platformy .NET. Aktuální verze jazyka je 2.0.

Microsoft má registrovány následující standardy:

- CLI:**http://www.ecma-international.org/publications/standards/ECMA-335.HTM**
- C#: **http://www.ecma-international.org/publications/standards/ECMA-334.HTM**

### 1.1.5 .NET

Platforma pro vývoj a běh programů. Překladem jazyků vysoké abstrakce (C#, VB.NET,…) vzniká binární IL kód, který je na cílovém operačním systému z části interpretován, z části kompilován (již jako instrukce procesoru) a spouštěn. Program tak může být napsán v několika jazycích dle libovůle a zvyklostí programátora. Jádrem pro vývoj aplikací jsou pak knihovny ve jmenném prostoru System (nachází se zde například základní typy System.Int32, ale i rozsáhlé statické třídy typu System.Console, či abstraktní třídy pro vytváření například formulářů, apod.).

Běhové prostředí se nazývá .NET Framework. Aby bylo možné spustit .NET aplikaci, je nutné mít toto běhové prostředí nainstalováno. Aktuální verze, distribuovaná na všechny Windows systémy prostřednictvím aktualizací, je 2.0 (resp. nadstavba 3.0, která však obsahuje pouze Windows XXX Framework knihovny).

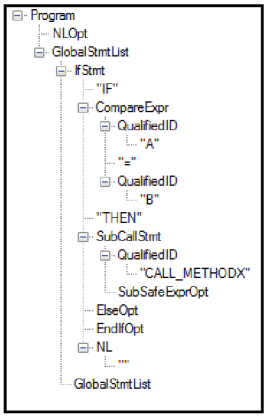
Stávající implementace překladače/CLR:

- Microsoft .NET Framework (**http://www.microsoft.com**) (pouze platformy Win)
- Microsoft Rotor („Shared source“ CLI) (**http://msdn.microsoft.com/**) (psaný v iso c)
- Ximian/Novell Mono (**http://www.go-mono.com/**) (Win/Linux/MacOs/Solaris..)
- DotGNU Portable.NET (**http://www.dotgnu.org/**) (Win/Linux/MacOs/Solaris..)
- OCL – není překladač, ale C# knihovna tříd od společnosti Intel – uvedeno jen pro zajímavost (**http://sourceforge.net/projects/ocl**)

[1] European Computer Manufacturers Association

### 1.1.6 Derivační strom

Je strom nonterminálů a terminálů získaných derivací zdrojového textu. Například zdrojový text „IF A=B THEN CALL_METHODX“ lze derivovat následovně viz Obr 1.1 Ukázkový derivační strom (pro přehlednost byla při analýze vynechána pravidla, která pouze přepisovala nonterminál za nonterminál).




**Obr 1.1 Ukázkový derivační strom**

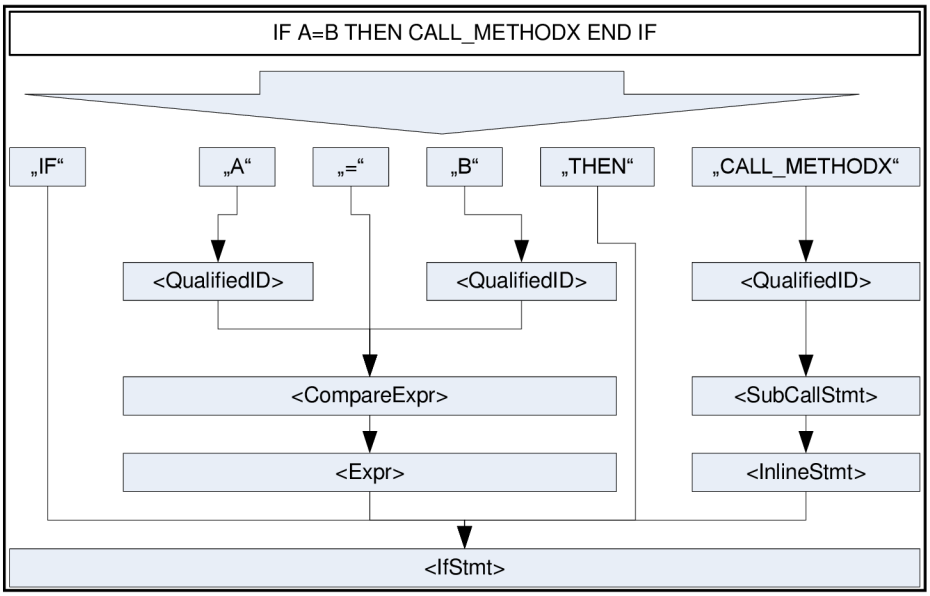
### 1.1.7 Derivační pravidlo

Pravidlo použité pro derivaci původních symbolů/tokenů (nonterminálů a terminálů) na symbol nový, tzv. derivovaný symbol. Po postupné derivaci všech symbolů získaných ze zdrojového řetězce na jediný symbol/token získáme derivační strom daného řetězce.

Derivační strom získaný z řetězce „IF A=B THEN CALL_METHODX“ byl získán následujícími derivačními pravidly (zjednodušeno, vynechána pravidla vedoucí k získání nonterminálu QualifiedID):

```
1.   <CompareExpr>  := <QualifiedID> "=" <QualifiedID>
2.   <Expr>         := <CompareExpr>
3.   <SubCallStmt>  := <QualifiedID>
4.   <InlineStmt>   := <SubCallStmt>
5.   <IfStmt>       := "IF" <Expr> "THEN" <InlineStmt>
```

Také viz následující obrázek:




**Obr 1.2 Ukázkový postup derivace za pomoci derivačních pravidel**

### 1.1.8 Konverzní pravidlo

Slouží pro konverzi symbolů derivačního stromu. Je svázáno s pravidlem derivačním a popisuje, jak se dané pravidlo přepíše tak, aby výstup odpovídal cílovému jazyku.

Například pro výše uvedené derivační pravidlo <IfStmt> := „IF“ <Expr> „THEN“ <InlineStmt> (jazyka VBScript) lze uvést následující příklad konverzního pravidla, konvertující derivační pravidlo do gramatiky jazyka C#:

```
1.  TranformRule("<IfStmt>:= "IF" <Expr> "THEN" <InlineStmt>") := "if(" [1] ") {" [3] "}"
```

> Poznámka: výše uvedená syntaxe zápisu se nikde nepoužívá, ve skutečnosti jsou pravidla zapsaná ve formátu XML až na vlastní výraz (uvedený za „:=“), který je zapsán v syntaxi Transform expression grammar. Více o konverzní gramatice viz 4 - VBScript gramatika a její konverze.

V současné době nejsou k dispozici žádné vhodné nástroje, které by beze zbytku splňovaly zadání diplomové práce. Dostupné nástroje pro konverzi ASP na ASP.NET jsou „Migrační asistent“ společnosti Microsoft a „Asp2Aspx“ společnosti NetCoole. Přednosti a nedostatky těchto nástrojů budou více rozebrány dále.

## 2.1 Migrační asistent (Microsoft)

Tento software je poskytován zdarma na stránkách společnosti Microsoft. Případní zájemci jej mohou stáhnout z adresy **http://www.asp.net/migrationassistants/asp2aspnet.aspx**.

Hlavním cílem tohoto asistenta je přinutit ASP stránky běžet v kontextu ASP.NET pouze s minimálními změnami. Žádná reálná konverze kódu zde tedy nenastává.

Hlavní činnosti programu jsou:

- Změna ASP přípony na ASPX a přepsání všech odkazů na názvy souborů v kódu, kterých se to týká (tj. html odkazy, formuláře,…).
- Změna některých značek, případně záměna za jiné (klauzule <!-- #INCLUDE ... -->, <%, apod.).
- Pokus o predikci typů deklarovaných proměnných a jejich přesun.

***Informace převzaty z http://www.asp.net/migrationassistants/GettingStarted_ASPtoASPNET.htm#WhatWillDo***

Po provedení několika zkušebních migrací na jednoduchých ASP stránkách a následně na celé aplikaci GetmoreSystem jsem došel k závěru, že tento nástroj je pro účely opakované a úplné konverze aplikace zcela nevhodný. Pro úplnou konverzi (i za přispění tohoto nástroje) by bylo třeba několik stovek člověko-hodin pro provedení úplné a dokonalé konverze. Zejména by však během této doby byla vyloučena jakákoliv úprava stávající aplikace. Také sám výrobce udává, že konverze je možná pouze za předpokladu **dodatečného upravení ze strany programátora**.

## 2.2 Asp2Aspx (NetCoole)

Tento software je placený – nejlevnější (standard) verze stojí 299$, nejdražší (enterprise) 595$. V trial verzi poskytován po registraci ke stažení na stránkách **http://www.netcoole.com/download.asp** (umožní konvertovat pouze 300 řádků kódu).

> Poznámka: Protože zdrojové kódy Getmore aplikace měly více řádků, než 300, které povolovala trial verze software, bylo třeba toto omezení odstranit. Zvolil jsem metodu dekompilace IL kódu (použitím **ildasm exe_file /out:src.il**), úpravy všech konstant, které omezovaly software a opětovné kompilace programu (**ilasm src.il /exe**). Nedopustil jsem se tím (ve smyslu českých zákonů) žádného porušení právních předpisů, smluvních dokumentů ani jiných nařízení, neboť přiložená licenční smlouva, která byla součástí programu a která článkem 4 podobnou činnost zakazovala, nebyla (ve smyslu právního řádu české republiky) mezi mnou a společností NetCoole řádně uzavřena. Jsem přesvědčen, že mé jednání bylo nejen legální, ale i legitimní, neboť jsem takto upravený program smazal, hned vzápětí po odzkoušení na reálných zdrojových kódech.

### 2.2.1 Zápory aplikace

Obecně (dle stránek výrobce) program neumí konvertovat

- VBScript stránky, kde je namísto nového řádku použit symbol “:“[1]
- Eval a GetRef

[1] Což je ve VBScriptu zcela normální způsob ukončení řádků. Viz 1.1.1 - VBScript, str. 7.

Výše uvedené nedostatky by šly napravit předzpracováním, resp. post zpracováním. Nicméně při testování konverze nad zdrojovými kódy aplikace GetmoreSystem nebyl program schopen zkonvertovat soubory tak, aby bylo možné stránky zobrazit bez chyb.

Některé z chyb

- Běžně se v aplikacích pracuje s tzv. vkládanými soubory (klauzule INCLUDE). Týká se to zejména obecných funkcí (vykreslení HTML stránky, přihlašování a práce s právy,...), které jsou pak (umístěny v jednom souboru) odkazovány v ostatních ASP souborech. Tato klauzule však způsobí, že ve zkonvertovaných stránkách se tyto vložené soubory vloží **fyzicky**, což znamená „rozmnožení“ tohoto obecného kódu.
- Problémy s konverzí některých systémových procedur. Například **UCase(proměnná)** je zkonvertováno jako **proměnná.ToUpper()**, což vede v některých případech k běhovým chybám, pakliže je například **proměnná** neinicializovaná. UCase(null) je totiž v pořádku (výsledná hodnota je null), nicméně (null).ToUpper() již způsobí runtime chybu NullReferenceException – také viz příloha.
- Názvy jmenných prostorů tříd se berou podle názvů adresářů. Pakliže je adresář pojmenovaný jako klíčové slovo (například base), znemožní to úspěšnou kompilaci (stačilo by například vždy dát první písmeno adresáře velké, nebo použít prefix či postfix)
- Chyba v uzavírání závorek – metody, které byly součástí tříd, měly někdy „uprostřed těla“ zpětnou složenou závorku navíc. Nebylo vypozorováno, kdy nebo proč to nastává.
- Občas dvojnásobná deklarace globálních proměnných. Nebylo vypozorováno, kdy nebo proč to nastává.
- Špatná konverze volání SQL příkazů přes objekt SqlConnection. Přidána dvojitá závorka – například **Conn.Execute()()**.
- Špatná konverze polí a některých metod. Metody považovány za pole a naopak.
- Ačkoliv pracuje s komentáři, nedává je k metodám jako <summary>, ale pouze jako komentáře, což značně omezuje potenciál jejich využití. Pokud by byly komentáře ve značce <summary>, byly by zobrazovány ve Visual Studiu jako InteliSence nápověda.
- Špatná konverze Regex metody pro regulární výrazy. Nesprávné použití .NET třídy. Nutný zásah programátora.
- Nerozpozná metodu CreateObject pokud není uvozená třídou Server.
- Nezvládá „neznámé typy“ COM objektů - například Scripting.FileSystemObject.
- Nepodporuje příkaz „On Error“

### 2.2.2 Přínosy aplikace

Aplikace je přínosná tím, že:

- Umí zjistit použití proměnných a odhadnout typ (int, string,…)

### 2.2.3 Závěr

Ani tato aplikace pro konverzi ASP stránek není použitelná pro zamýšlený cíl této diplomové práce.

Na základě studia současného stavu dostupných aplikací jsem usoudil, že bude nutné vytvořit zcela novou aplikaci, která by plně vyhovovala cílům této diplomové práce. Tato kapitola se bude zabývat konkrétními požadavky na aplikaci, výběrem vhodných nástrojů pro implementaci aplikace a postupem při samotné konverzi.

## 3.1 Požadavky

Aplikace by měla umožnit jednostrannou, plně automatickou konverzi ASP stránek, psaných ve VBScript jazyce, do jazyka C#, platformy .NET.

Aplikaci bude předán (případně bude moci uživatel zvolit přímo v aplikaci) adresář, ve kterém se nachází ASP stránky a adresář, kam se mají vygenerovat ASPX stránky (v případě potřeby bude vytvořen).

Nedílnou součástí aplikace jsou jiné než ASP soubory (například obrázky, CSS styly, JS soubory,…). Všechny tyto součásti musejí být také zkopírovány do cílového adresáře.

Mimo samotné konverze kódu bude aplikace provádět přejmenování stránek z ASP na ASPX koncovku. Tyto úpravy se musejí projevit i v kódu samotném.

Přejmenování ASP souborů se dotkne zejména:

- Formulářů (<form> tag)
- Odkazů (<a> tag)
- JavaScriptu
    - Javascript v html tagu (<a> tag, onclick a jiné události na prakticky všech prvcích)
    - Javascript přímo na ASP stránce (<script> tag)
    - Javascript zdrojový soubor (*.js soubory)

Všude na výše uvedených místech musí dojít ke konverzi přípony .asp na .aspx. Bude se tedy hledat následující regulární výraz:

```
\.(asp)[ \f\n\r\t\v\?\"\'$]
```

Aplikace také musí umožňovat implicitní a uživatelské přiřazení .NET tříd pro vytvářené COM objekty v ASP stránkách. Například objekt Server.CreateObject(“Scripting.Dictionary“) se musí zkonvertovat jako System.Collections.Specialized.HybridDictionary (implicitně), nebo takovou třídu, kterou zvolí uživatel.

Samotná konverze tedy musí probíhat následovně:

- Analýza zdrojových ASP souborů a generování C# kódu
    - Rozdělení na logické celky
    - Vytvoření derivačního stromu za pomoci VBScript gramatiky
    - Konverze derivačního stromu na objektový DOM model jazyka C# (viz 3.4 - Objektový model jazyků .NET - CodeDOM).
    - Přetypování proměnných, dohledání jmenných prostorů symbolů, vymezování působnosti…
    - Nahrazení koncovky ASP na ASPX ve všech řetězcích
    - Generování C# kódu do souborů v cílových adresářích
- Kopie HTML/js souborů a nahrazení referencí na ASP soubory
- Kopie ostatních souborů v rámci původního adresáře aplikace

Samotná analýza původního kódu tedy musí jako vstup použít vhodnou gramatiku a s tím se pojící vhodnou knihovnu pro práci s touto gramatikou. O tom více viz další podkapitola.

## 3.2 Systémy pro práci s gramatikami

Zkoumal jsem vlastnosti a možnosti následujících tří produktů:

- Yacc
- ANTLR
- GOLD Parser

První dva produkty dokážou dle definice gramatiky vygenerovat zdrojový kód pro Lexér/Parser/… (do zvoleného cílového jazyka. V případě produktu GOLD Parser se negeneruje zdrojový kód, ale je poskytnut přímo syntaktický analyzátor, který dynamicky používá zvolenou gramatiku. Produkty se liší v zápisu gramatik, v podporovaných typech gramatik a tím i v analýze, kterou používají (např. LALR…)

### 3.2.1 Yacc

Tímto generátorem jsem se přestal zabývat hned v okamžiku, kdy jsem zjistil, že v současné době nepodporuje generování do C#. Nenalezl jsem vhodnou VBScript gramatiku. Generování parseru probíhá metodou LALR. Gramatika je zapsaná v BNF[1] formě. Často se používá ve spojení s programem Lex.

### 3.2.2 ANTLR

V současné době (1.1.2007) je aktuální **Verze 2.7.7 (2006/11/01)**. Souběžně s verzí 2 je již cca čtyři roky vyvíjena i verze 3 (poslední public **verze 3.0 beta 5 (2006/12/15)**). Verze 3 na rozdíl od verze 2 má odlišnou meta gramatiku pro zápis gramatik a generuje parsery jinou metodou - LL(*). Verze 2 generuje LL(k) parsery.

Aplikace podporuje (nativně) generování C# kódu.

Verze 2 se chovala stabilně, nicméně nalezl jsem pouze jedinou VBScript gramatiku, ze které však nešel vygenerovat kód. Gramatika nešla převést ani do zápisu kompatibilního s verzí 3 (konvertorem, který byl ke stažení na autorových stránkách).

Aplikace ve verzi 3 není prozatím přístupná jako balíček “easy install“, nepodařilo se mi ji bohužel korektně nainstalovat a spustit.

### 3.2.3 GOLD Parser

V současné době (1.1.2007) je aktuální **Verze 3.2.4 (2006/12/09)**. Syntaktická analýza probíhá stylem LALR (Look Ahead Left-Right parsing), používá se tabulka symbolů a přechodů. Ta je ještě ve fázi návrhu gramatiky „zkompilována“ z gramatiky do binární podoby.

Gramatika je zapsaná v BNF formě.

Přesto, že se jedná o relativně pomalé řešení, jeho hlavní výhoda spočívá v možnosti změny gramatiky bez nutnosti opětovné kompilace aplikace.

Ze stránek GOLD parseru jsem stáhnul VBScript gramatiku vytvořenou Vladimirem Morozovem. Gramatiku jsem otestoval nad celou aplikací GETMORE a nalezl jsem pouze několik drobných problémů.

Na stránkách je k dispozici kompilátor gramatiky na překladovou tabulku, dále pak několik různých „strojů“ pro zpracování této zkompilované gramatiky a vytváření derivačních stromů pro nejrůznější jazyky.

[1] Backus–Naur form, také viz **http://en.wikipedia.org/wiki/Backus-Naur_form**

Pro jazyk C# jsou k dispozici následující knihovny

- Klimstra
- Calitha
- Mozorov

## 3.3 Logika konverze ASP na ASP.NET

V této části se budeme zabývat konkrétním postupem při konverzi ASP souborů.

Struktura ASP souboru je následující:

```
ASP soubor = FILE SCOPE
<!-- other ASP file includes -->
<%
    ASP kód
        Deklarace „globálních" proměnných (Dim), přístupných z vnořených ASP
        volání metod tohoto souboru ale i jiných

    Funkce = FUNCTION SCOPE
        ASP kód
            deklarace lokálních proměnných (Dim)
            volání metod tohoto souboru ale i jiných

%>
HTML tags…
<% … %>
```

ASP.NET zdrojové soubory se pak skládají z ASPX souboru a CS souboru (tzv. code behind). ASPX soubor by měl obsahovat pouze (x)HTML a ASPX XML značky (např. <asp:label …>). CS soubor pak obsahuje zdrojový text psaný v některém z .NET jazyků, my budeme generovat C#. V souboru je deklarace třídy, která obaluje konkrétní přiřazenou ASPX stránku a obsahuje implementaci obsluh událostí (onLoad, onPostBack,…). Zdrojový text je možno překompilovat do DLL knihovny, nebo je možno jej nechat on-fly generovat přímo .NET frameworkem při přístupu prvního uživatele.

Konverze jednoho ASP souboru ({soubor}.asp) tedy vygeneruje soubor ASPX ({soubor}.aspx) a code behind ({soubor}.aspx.cs)).

**Šablona soboru** {soubor}**.aspx:**

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="{soubor}.aspx.cs" Inherits =
"{NameSpace(soubor)}.{ClassName(soubor)} " %>
```

Jmenný prostor odpovídá zanoření adresářů, název třídy pak názvu souboru s postfixem „Page"

```
<%@ Reference Page="{odkaz na původní include ASP souboru}.aspx" %>
```

0..n referencí

```
<!-- file {soubor}.aspx; Copyrights GETMORE,s.r.o. (c) 1997-2006 All rights reserved -->
```

**Šablona soboru** {soubor}**.aspx.cs:**

```
{using directivy - system, referencované ASPX stránky…}

namespace {NameSpace(soubor)}
{
    public partial class {ClassName(soubor)} : AspPage
    {
        {DEPENDED CLASSES - původně include ASP stránky, nyní jako privátní vlastnosti}
        {globální promenné/properties}
        {veřejně přístupné metody - původní subprocedury a funkce}
        {vlastní původní kód stránky jako metoda ExecutePage}
    }
}
```

Stejně jako v .NET budeme uvažovat, že každá ASP stránka je jedna třída. Jmenný prostor této třídy budiž závislý na adresáři, ve kterém se ASP stránka nachází. Všechny funkce a sub procedury deklarované v ASP stránce budou vygenerovány jako public metody této třídy. Všechny globální proměnné pak budou přístupné jako veřejné vlastnosti třídy (public properties).

Třída bude dále obsahovat privátní instance tříd stránek, které jsou ASP stránkou vkládány (je na nich závislá).

CS soubor bude vytvořen na základě DOM struktury jazyka C#. Konvertor, převádějící stránku ASP, tedy musí na základě původního derivačního stromu a konverzních pravidel (potažmo metod) vygenerovat přímo cílový strom objektového modelu jazyka C# (viz 3.4 - Objektový model jazyků .NET - CodeDOM).

Tímto degradujeme konverzi skriptovacího jazyka ASP na ASP.NET pouze na konverzi posloupnosti příkazů, ustanovení jazyka (bloky if, while,…) a vyřešení lokace závislostí (voláme metodu XY… tato se nachází ve třídě aktuální ASP stránky nebo jinde?)…

Při konverzi bude generován/updatován XML dokument, který bude obsahovat všechny nejasnosti, resp. „dotazy“ na uživatele i s odkazem na původní a nový soubor/umístění. Uživatel tak bude mít možnost bez ztráty znovu generovat konvertovaný kód i s jeho úpravami. Eventuálně lze také tento upravený kód psát do komentářů…

## 3.4 Objektový model jazyků .NET - CodeDOM

**Code Document Object Model** (CodeDOM) je pojem označující objektový model dokumentu zdrojového kódu. CodeDOM pomocí jednotlivých objektů (kontejnerů) by měl odrážet kompletní hierarchii jazykových elementů zdrojového kódu (tj. jmenných prostorů, tříd, metod, příkazů, výrazů, komentářů, apod.).

CodeDOM tedy slouží pro vyjádření zdrojového kódu libovolného (podporovaného) jazyka v objektové podobě.

Koncept modelu byl vytvořen na základě výběru **podmnožiny** jazykových elementů jazyků .NET. Pomocí CodeDOM tedy **nelze** vyjádřit jakýkoliv zdrojový kód libovolného jazyka .NET. Někdy schází i základní prvky, jako třeba unární operátor. Většinou je ale možné nahradit daný prvek bez ztráty jazykové neutrality.

CodeDOM je logicky a funkčně rozdělen do následujících jmenných prostorů:

- **System.CodeDOM** – obsahuje třídy, které můžeme využít při vytváření CodeDOM stromové struktury.
- **System.CodeDom.Compiler** – definuje bázové třídy pro generování zdrojového kódu z CodeDOM grafů a struktur, a prostředky pro kompilaci zdrojového kódu v podporovaných jazycích.

Je třeba zejména zdůraznit, že výše uvedené jmenné prostory jsou obecně použitelné pro jakýkoliv jazyk, podporující CodeDOM. Konkrétní implementace (například pro jazyk C#) je nutné hledat v jiném jmenném prostoru. O tom viz níže.

### 3.4.1 Poskytovatelé CodeDOM služeb

V současné implementaci .NET frameworku existují následující nativní poskytovatelé CodeDOM služeb:

- Pro jazyk C# je to třída Microsoft.CSharp.**CSharpCodeProvider**
- Pro jazyk VB.NET je to třída Microsoft.VBscript.**VBCodeProvider**
- Pro jazyk JScript.NET je to třída Microsoft.JScript.**JScriptCodeProvider**
- Pro jazyk J# je to třída Microsoft.VJSharp.**VJSharpCodeProvider**
- Pro jazyk C++ je to třída Microsoft.MCpp.**MCppCodeProvider** (MCppCodeDomProvider.dll)

Je samozřejmě možné napsat vlastního poskytovatele pro svůj vlastní jazyk. Jazyk však musí splňovat podmínku, že vyhovuje CodeDOM specifikaci. CodeDOM není nicméně žádný veřejný standard, našel jsem (kromě MSDN) jedinou trochu schůdnou specifikaci:

**http://kahu.zoot.net.nz/codedom/CodeDom Grammar.html**

### 3.4.2 Použití CodeDOM

Jak lze tedy CodeDOM použít? Níže uvádím některé případy použití CodeDOM.

Dynamické vytváření assembly:

- Vytvoříme strukturu programu (třídy z System.CodeDOM)
- Vytvoříme poskytovatele CodeDOM služeb (např. Microsoft.CSharp.**CSharpCodeProvider**)
- Zkompilujeme daný program metodou CompileAssemblyFromDom (uvedeného poskytovatele CodeDOM služeb)

Způsoby vytváření CodeDOM struktur:

- Ručně
- Analýzou textu za pomocí poskytovatele CodeDOM služeb, metoda Parse

Úprava/refaktorizace zdrojového kódu:

- Vytvoření struktury programu (např. použitím metody Parse)
- Procházení CodeDOM struktury a provádění úprav (např. přejmenování metody a všech výskytů dané metody a to včetně komentářů, …)
- Opětovné vygenerování zdrojového textu programu metodou GenerateCodeFromCompileUnit poskytovatele CodeDOM služeb.

### 3.4.3 CodeSnippets

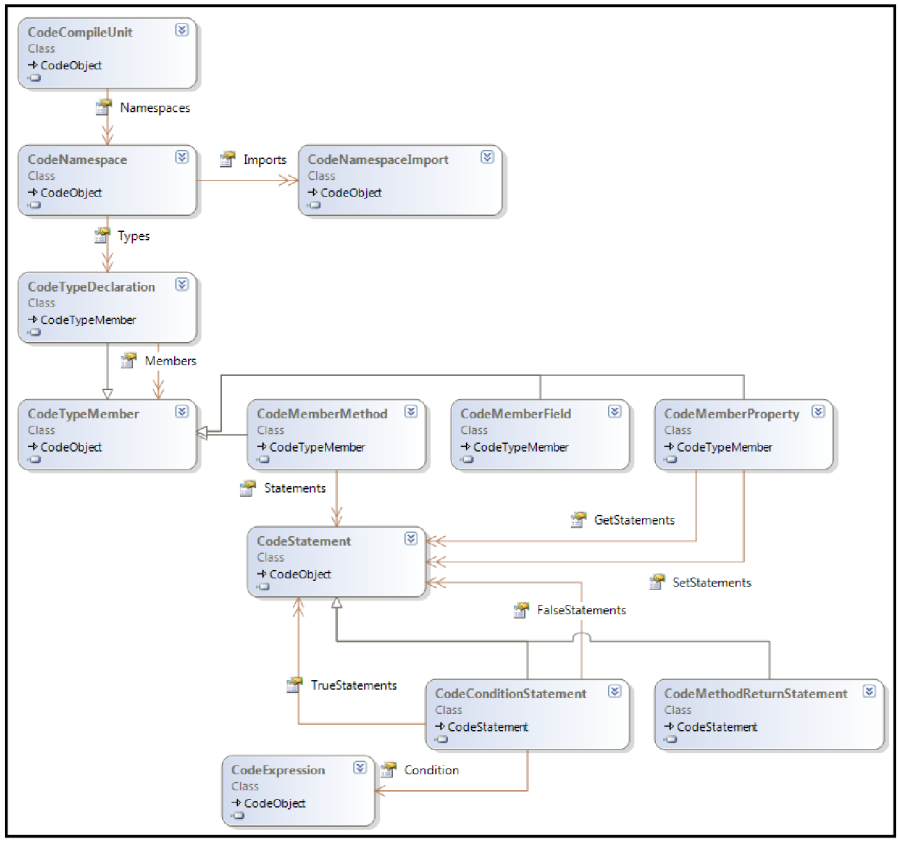
Jak bylo v úvodu naznačeno, některé prvky konkrétních jazyků nelze vyjádřit, resp. nejsou podporovány všemi jazyky (např. unární operátor). Takové prvky je možné vyjádřit instancí některé ze tříd CodeSnippetXXX. Instance těchto tříd obsahují zejména vlastnost „Value“, která je typu string a může být nastavena na teoreticky jakoukoliv hodnotu. Je pak na konkrétní implementaci třídy CodeProvider, jak se s daným textem vyrovná. CodeSnippets jsou tedy jazykově závislé!

Konkrétní „snippet“ třídy jsou následující:

- CodeSnippetCompileUnit
- CodeSnippetExpression
- CodeSnippetStatement
- CodeSnippetTypeMember

### 3.4.4 Přehled a diagram tříd CodeDOM

Přehled diagramu tříd podle CodeDOM modelu je vidět na Obr 3.1 CodeDOM diagram tříd. Pro jednoduchost jsem zobrazil třídy pouze po úroveň „členů“ typů (tj. metoda, vlastnost, proměnná) a z nižší úrovně jsem vybral pouze příkazy IF a RETURN. Vynechal jsem také zobrazení ostatních vlastností (např. „Name“ ve třídě „CodeNamespaceImport“ apod.).




**Obr 3.1 CodeDOM diagram tříd**

Výsledkem použití CodeDOM modelu jako výstupu konverze je mimo jiné i možnost generovat výstup v syntaxi libovolného podporovaného .NET jazyka.

Celá použitá gramatika viz příloha 10.4 - Gramatika VBScript, str. 56. Tato část více rozebere konkrétní výrazy VBScript jazyka a jejich konverze do C#.

Konverzi budeme provádět zejména pomocí konverzních metod, které budou generovat samotné objekty CodeDOM modelu. Již při samotné konverzi tak ověříme, zda lze ze zdrojového textu za použití konverzních pravidel vyprodukovat odpovídající strukturu jazyka C#. Pokusíme se napsat pravidla tak, aby nezáleželo na konkrétním kontextu a vždy se mohl očekávat stejný typ výsledku.

Pravidla a metody budou uložena v konverzním balíčku – více viz 5.4 - Konverzní balíček.

## 4.1 Zápis konverzních pravidel

Zápis konkrétního výrazu pro konverzi bude zapsán v syntaxi podle gramatiky „Transform expression grammar“. Definice této gramatiky viz 10.3 - Gramatika Transformace.

Lze definovat klíčová slova:

- NOT_SUPPORTED – konverze daného pravidla není podporována
- DEFAULT – konverze daného pravidla bude probíhat implicitně dle každého jeho „pod nonterminálu/terminálu“ a eventuálně jejich konverzních pravidlům
- NOTHING – při použití tohoto konverzního pravidla se nic neprovede

Lze definovat následující prvky konverze

- Text ve dvojitých uvozovkách – tento text se generátorem připojí jako tzv. „SnippetExpression“. V textu lze uvádět speciální „escaped“ znaky (např. "term(\"a\")= b\n")
- Odkaz na non/terminál z derivačního stromu ve formě hranatých závorek a nulou začínajícím indexem non/terminálu v původním výrazu pro derivaci (např. [0] odkazuje na první non/terminál původního výrazu). Pro tento non/terminál se spustí samostatná konverze.
- Volání metody správce konverze jako název metody a kulaté závorky. Metoda může mít 0-n parametrů. Parametr může být buď text, odkaz na non/terminál nebo klíčové slovo (viz níže). Volání metody je citlivé na velikost písmen. Příklad: generateIf([0],[2],null,“and“).

Klíčová slova sloužící jako parametry konverzním funkcím:

- Null
- False
- True

Následující podkapitoly řeší vždy jednu oblast gramatiky VBScript. Nejprve je obecný popis, následuje výpis pravidel a jejich konverze do jazyka C# (zapsaný podle notace viz výše). Zcela na konci je pak vypsán soupis všech použitých metod i jejich očekávaný CodeDOM výstup.

Prázdná pravidla (nonterminál → ε), která jsou v původní gramatice a **která se nekonvertují**, nebudou vypsána.

Výpis pravidel a jejich konverze uvádím v následujícím formátu:

```
<Derivovaný symbol>   ::= Původní symboly pravidla, které toto pravidlo derivuje
```

*Konverzní výraz podle gramatiky „Transform expression grammar“*

## 4.2 Deklarace třídy

Tato syntaxe není nyní v aplikaci GETMORE použita, bude rezervováno pro další rozšíření. Všechna pravidla mají tedy příznak NOT_SUPPORTED.

Toto se týká následujících nonterminálů z VBScript gramatiky a jejich derivačních pravidel:

- <ClassDecl>
- <MemberDeclList>
- <MemberDecl>
- <FieldDecl>
- <FieldName>
- <FieldID>
- <PropertyDecl>
- <PropertyAccessType>

## 4.3 Deklarace proměnné

Při deklaraci proměnné

- Uložíme si její název a oblast působnosti
- Zkusíme odhadnout její typ (při prvním průchodu zjistíme, kde všude a jak se používá její identifikátor, podle toho měníme přiřazení typu)

Globální proměnné celého ASP souboru je nutné generovat jako vlastnosti přidružené třídy.

VBScript má proměnné typu Variant. Pakliže nebudeme moci určit typ proměnné, vytvoříme ji právě jako typ Variant s obdobnou funkcionalitou jako ve VBScriptu. Tedy – proměnné tohoto typu budou vždy inicializovány na instanci třídy. Hodnota bude nastavována jako proměnná.SetValue(). Bude mít vlastnost VarType vracející podobně jako ve VBScriptu „String“, „Int“, „Empty“, „VbNull“, …

```
<VarDecl> ::= Dim <VarName> <OtherVarsOpt> <NL>
```
```
        [1] [2]
```

```
<VarName> ::= <ExtendedID> ( <ArrayRankList> )
```
```
        GenerateNewVariable([0],[2])
```

```
<VarName> ::= <ExtendedID>
```
```
        GenerateNewVariable([0])
```

```
<OtherVarsOpt> ::= , <VarName> <OtherVarsOpt>
```
```
        [1] [2]
```

```
<OtherVarsOpt> ::=
```
```
        NOTHING
```

```
<ArrayRankList> ::= <IntLiteral> , <ArrayRankList>
```
```
        NOT_SUPPORTED /* CodeDom doesn't support mutil-dimension array*/
```

```
<ArrayRankList> ::= <IntLiteral>
```
```
        DEFAULT
```

```
<ArrayRankList> ::=
```
```
        NOT_SUPPORTED
```

## 4.4 Deklarace konstanty

Konstanty je nutné generovat jako součást přidružené třídy a při jejich odvolání uvádět i název této třídy. Tedy platí více méně to, co při deklaraci proměnné. Podporované typy konstant budou pouze logické (Boolean), číslo (int, float) a řetězec (string).

```
<ConstDecl>          ::= <AccessModifierOpt> 'Const' <ConstList> <NL>
```
```
        GenConstants([0],[2])
```

```
<ConstList>          ::= <ExtendedID> '=' <ConstExprDef> ',' <ConstList>
                     | <ExtendedID> '=' <ConstExprDef>
```
```
        <nelze vygenerovat jako překlad gramatiky, je řešeno metodou GenConstants - viz výše>
```

```
<ConstExprDef>          ::= '(' <ConstExprDef> ')'
                        | '-' <ConstExprDef>
                        | '+' <ConstExprDef>
                        | <ConstExpr>
```

*<**není třeba** konvertovat - syntaxe stejná jak v C#>*

## 4.5 Sub procedura

Procedura – při její deklaraci se musí uložit na zásobník její SCOPE. Nebude se kontrolovat platnost argumentů (počet, typy,…) – toto bude ponecháno na finální kompilaci .NET frameworku.

Nutné vyřešit ByRef a ByVal proměnné (při ByRef musí být atribut argumentu „ref“).

```
<SubDecl>           ::= <MethodAccessOpt> Sub <ExtendedID> <MethodArgList> <NL> <MethodStmtList>
                        End Sub <NL>
```

*GenNewMethod(False,[0],[2],[3],[5])*

```
<SubDecl>           ::= <MethodAccessOpt> Sub <ExtendedID> <MethodArgList> <InlineStmt> End Sub
                        <NL>
```

*GenNewMethod(False,[0],[2],[3],[4])*

## 4.6 Funkce

Procedura – při její deklaraci se musí uložit na zásobník její SCOPE. Nebude se kontrolovat platnost argumentů (počet, typy,…) – toto bude ponecháno na finální kompilaci .NET frameworku.

Návratová hodnota funkce bude buď vždy VbVariant nebo lze po skončení prvního průchodu určit její návrat podle přiřazení uvnitř funkce (return True,…).

Je nutné vyřešit návrat z vnitřku funkce Exit Function (návrat VbVariant.Null?). Přiřazení návratové hodnoty při deklaraci název metody = hodnota – nutné vyřešit přes return new VbVariant(hodnota).

```
<FunctionDecl>      ::=   <MethodAccessOpt>   Function   <ExtendedID>   <MethodArgList>   <NL>
                        <MethodStmtList> End Function <NL>
```

*GenNewMethod(True,[0],[2],[3],[5])*

```
<FunctionDecl>      ::= <MethodAccessOpt> Function <ExtendedID> <MethodArgList> <InlineStmt> End
                        Function <NL>
```

*GenNewMethod(True,[0],[2],[3],[4])*

Je nezbytné vzít v potaz, že přiřazení návratové hodnoty funkci může probíhat vícekrát:

```
1.  <%
2.
3.  function TestMe()
4.      Response.Write "TestMe(): Start<br>"
5.      TestMe = 1
6.      Response.Write "TestMe(): Behind first ass<br>"
7.      TestMe = 2
8.  End Function
9.
10.
11. Response.Write "<br>TestMeResult: "&TestMe()
12.
13.
14. %>
```

Výše uvedený kód způsobí následující výstup:

```
1.  TestMe(): Start
2.  TestMe(): Behind first ass
3.
4.  TestMeResult: 2
```

Je tedy nutné s tím počítat – návratová hodnota funkce tak musí být deklarována na začátku funkce jako proměnná „ret“ a při příkazu „return“ se tato hodnota musí vrátit.

## 4.7 Modifikátory a jiné prvky použité u funkcí/procedur

Operátory public a private odpovídají významu public a private deklarovaných na úrovni tříd. Výraz „Public Default“ bude prozatím nepodporovaný. U parametrů metod pak musíme vzít v úvahu jejich umístění (jmenný prostor, název třídy) a v neposlední řadě i způsob odkazu (zda ByVal nebo ByRef).

```
<MethodAccessOpt>     ::= Public Default
        NOT_SUPPORTED

<MethodAccessOpt>     ::= <AccessModifierOpt>
        NOT_SUPPORTED

<AccessModifierOpt>   ::= Public
        GenMemberAttribute("Public")

<AccessModifierOpt>   ::= Private
        GenMemberAttribute("Private")

<AccessModifierOpt>   ::=
        GenMemberAttribute("Public") /*DEFAULTNI MODIFIKATOR!!!*/

<MethodArgList>   ::= ( <ArgList> )
        [1]

<MethodArgList>   ::= ( )
        NOTHING

<MethodArgList>   ::=
        NOTHING

<ArgList>         ::= <Arg> , <ArgList>
        [0] [2]

<ArgList>         ::= <Arg>
        DEFAULT

<Arg>             ::= <ArgModifierOpt> <ExtendedID> ( )
        NOT_SUPPORTED

<Arg>             ::= <ArgModifierOpt> <ExtendedID>
        GenArg([0],[1])

<ArgModifierOpt>  ::= ByVal
        GenArgModifierOptType("ByVal")

<ArgModifierOpt>  ::= ByRef
        GenArgModifierOptType("ByRef")

<ArgModifierOpt>  ::=
        GenArgModifierOptType("ByRef") /*v ASP jsou parametry procedur a funkcí implicitně
                nahrazovány odkazem*/
```

## 4.8 Obecné prvky

Tato část se týká spíše obecného rozdělení VBScript jazyka a není třeba definovat speciální překladová pravidla. Nonterminál <OptionExplicit> bude ignorován (z přirozenosti jazyka C# musí být všechny proměnné definovány před použitím, toto pravidlo tedy nemá opodstatnění).

Nonterminály definované v této části:

- <GlobalStmt>
- <MethodStmt>
- <BlockStmt>

- <InlineStmt>
- <GlobalStmtList>
- <MethodStmtList>
- <BlockStmtList>
- <OptionExplicit>

## 4.9 Ošetření chyb

Celá část týkající se ošetření chyb není zatím vyřešena. Jelikož se týká zejména ošetření stavu, kdy „něco“ skončilo chybou a tu je třeba sdělit uživateli, bude pravděpodobně stačit celé volání přeloženého ASP skriptu obalit do jednoho velkého Try-Catch bloku. Týká se to tedy pravidla <**ErrorStmt**>.

## 4.10 Exit statement

Prvky jazyka VBScript, které jsou s klíčovým slovem Exit. Ukončení bloku a odskok o blok výše. Pro vygenerování do CodeDOM je nutné vědět, jaký ukončovací blok se provádí – proto je metodě předán řetězec s popisem (např. „ExitDo“).

```
<ExitStmt>        ::= Exit Do
        GenExitStatement("ExitDo")
```

```
<ExitStmt>        ::= Exit For
        GenExitStatement("ExitFor")
```

```
<ExitStmt>        ::= Exit Function
        GenExitStatement("ExitFunction")
```

```
<ExitStmt>        ::= Exit Property
        GenExitStatement("ExitProperty")
```

```
<ExitStmt>        ::= Exit Sub
        GenExitStatement("ExitSub")
```

## 4.11 Příkaz přiřazení

VBScript zná dva způsoby přiřazení hodnoty proměnné. Dělí proměnné na objektové a hodnotové. Hodnotové proměnné se přiřazují operátorem “=“. Objektové proměnné se přiřazují příkazem „Set“ spojeným s operátorem “=“. V jazyku C# je toto zcela jedno. Oba dva výrazy tedy budou nahrazeny voláním stejné metody.

Třídy, jak bylo zmíněno na začátku, nebudou prozatím podporovány, proto příkaz “new“ nebude překládán.

```
<AssignStmt>      ::= <LeftExpr> = <Expr>
        GenAssignStatement([0],[2])
```

```
<AssignStmt>      ::= Set <LeftExpr> = <Expr>
        GenAssignStatement([1],[3])
```

```
<AssignStmt>      ::= Set <LeftExpr> = New <LeftExpr>
        NOT_SUPPORTED
```

## 4.12 Volání sub-rutin (procedur, funkcí)

Při volání sub-rutin je třeba k názvu procedury/funkce přidat odkaz na třídu, kde se nachází. Třída bude buď „this“, nebo privátní vlastnost “this“ třídy s instancí třídy cílové. VBScript dále povoluje (resp. přímo vyžaduje) volání procedur bez uvedení závorek. Na rozdíl od něj, C# naopak vyžaduje u volání metod mít uvedeny závorky vždy. Jinak se obecně volání metod v C# syntaxi od jazyka VBScript příliš neliší. Pro přehlednost zde neuvádím veškerá pravidla

z původní gramatiky, jejich konverze je totiž podobná, jako je konverze následujícího pravidla:

```
<SubCallStmt>     ::= <QualifiedID> ( <Expr> )
        GenMethodInvokeExpression([0], [2])
```

### 4.12.1 Příkaz Redim

Redim umožňuje dynamicky měnit velikost proměnné, deklarované jako pole. Budeme tedy počítat s tím, že byla proměnná deklarovaná jako pole – kontrola při konverzi nebude prováděna. Na případný problém se přijde v okamžiku kompilace. Pokud je u Redim příkazu specifikovaný modifikátor „Preserve“, musí se zachovat původní prvky pole. V opačném případě se nastaví na hodnotu „Nothing“.

```
<RedimStmt>       ::= Redim <RedimDeclList> <NL>
        GenRedimStatement([1],"false")

<RedimStmt>       ::= Redim Preserve <RedimDeclList> <NL>
        GenRedimStatement([2],"true")

<RedimDeclList>   ::= <RedimDecl> , <RedimDeclList>
        [0] [2]

<RedimDeclList>   ::= <RedimDecl>
        DEFAULT

<RedimDecl>       ::= <ExtendedID> ( <ExprList> )
        GenRedimDeclaration([0],[2])
```

### 4.12.2 If Statement

Pro konverzi příkazu IF použijeme konverzní metodu GenIfStatement. U ELSE pravidel musíme vypustit generování „else“ tokenu, neboť je již součástí generované instance CodeConditionalStatement.

```
<IfStmt>          ::= If <Expr> Then <NL> <BlockStmtList> <ElseStmtList> End If <NL>
        GenIfStatement([1],[4],[5])

<IfStmt>          ::= If <Expr> Then <InlineStmt> <ElseOpt> <EndIfOpt> <NL>
        GenIfStatement([1],[3],[4])

<ElseStmtList>    ::= ElseIf <Expr> Then <NL> <BlockStmtList> <ElseStmtList>
        GenIfStatement([1],[4],[5])

<ElseStmtList>    ::= ElseIf <Expr> Then <InlineStmt> <NL> <ElseStmtList>
        GenIfStatement([1],[3],[5])

<ElseStmtList>    ::= Else <InlineStmt> <NL>
        [1]

<ElseStmtList>    ::= Else <NL> <BlockStmtList>
        [2]

<ElseStmtList>    ::=
        NOTHING

<ElseOpt>         ::= Else <InlineStmt>
        [1]

<EndIfOpt>        ::= End If
        NOTHING
```

### 4.12.3 With Statement

Příkaz “with“ není v gramatice C# podporován. Konverze tedy musí proběhnout tak, aby došlo k vytvoření ekvivalentního sledu příkazů, které by nahradili příkaz with. Následuje příklad takové konverze (vlevo původní, vpravo zkonvertovaný kód). Ihned po příkladu je zobrazeno odpovídající konverzní pravidlo.

```
1.   With Classs1                                    1. Class1.Send("done", Class1.Auto);
2.       Send "done", .Auto
3.   End With
```

```
<WithStmt>            ::= 'With' <Expr> <NL> <BlockStmtList> 'End' 'With' <NL>
        GenWithStatement([1], [3])
```

### 4.12.4 Příkazy cyklu – While, Do, For a For Each

Tyto příkazy se liší jen velmi málo svou syntaxí od jazyka C#. Protože však generujeme výstup v podobě CodeDOM, musíme i zde použít generování za pomocí metod.

```
<LoopStmt>        ::= Do <LoopType> <Expr> <NL> <BlockStmtList> Loop <NL>
        GenLoopStatement("true",[4],[1],[2])
```

```
<LoopStmt>        ::= Do <NL> <BlockStmtList> Loop <LoopType> <Expr> <NL>
        GenLoopStatement("false",[2],[4],[5])
```

```
<LoopStmt>        ::= Do <NL> <BlockStmtList> Loop <NL>
        GenLoopStatement("false",[2])
```

```
<LoopStmt>        ::= While <Expr> <NL> <BlockStmtList> WEnd <NL>
        GenLoopStatement("true",[3],[1])
```

```
<LoopType>        ::= While
        GenLoopType("while")
```

```
<LoopType>        ::= Until
        GenLoopType("until")
```

```
<ForStmt>         ::= For <ExtendedID> = <Expr> To <Expr> <StepOpt> <NL> <BlockStmtList> Next
                      <NL>
        GenForStatement([1],[3],[5],[6],[8])
```

```
<ForStmt>         ::= For Each <ExtendedID> In <Expr> <NL> <BlockStmtList> Next <NL>
        NOT_SUPPORTED
```

```
<StepOpt>         ::= Step <Expr>
        NOT_SUPPORTED /* Step u For cyklu se zpracovává v rámci GenForStatement(...), sem to
                         nesmí dojít !*/
```

```
<StepOpt>         ::=
        NOT_SUPPORTED /* Step u For cyklu se zpracovává v rámci GenForStatement(...), sem to
                         nesmí dojít !*/
```

### 4.12.5 Příkaz Select

Je nutné nahradit příkazem **switch**. Zde však vyvstává problém, protože CodeDOM nepodporuje příkaz switch. Řešením je tedy použití CodeSnippet* třídy. Toto by však znemožnilo generování například Visual Basic jazyku. Pro generování do jiných jazyků, než je C# tedy použijeme CodeDOM prvek CONDITION.

```
<SelectStmt>      ::= Select Case <Expr> <NL> <CaseStmtList> End Select <NL>
        GenSwitchStatement([2],[4])
```

```
<CaseStmtList>    ::= Case <ExprList> <NLOpt> <BlockStmtList> <CaseStmtList>
        GenSwitchCaseStatement([1], [3]) [4]
```

```
<CaseStmtList>    ::= Case Else <NLOpt> <BlockStmtList>
        GenSwitchDefaultStatement([3])
```

### 4.12.6 Výrazy (Boolean, Matematické)

Pro konverzi výrazů typu A x B, kde x je binární operátor, použijeme výhradně konverzní metodu GenBinaryOperatorType. Příklad konverze:

```
<CompareExpr>     ::= <CompareExpr> Is <ConcatExpr>
        GenBinaryOperatorType([0],"IdentityEquality",[2])
```

Možné hodnoty typu operace:

- Add
- Subtract
- Multiply
- Divide
- Modulus
- Assign
- IdentityInequality
- IdentityEquality
- ValueEquality
- BitwiseOr
- BitwiseAnd
- BooleanOr
- BooleanAnd
- LessThan
- LessThanOrEqual
- GreaterThan
- GreaterThanOrEqual

### 4.12.7 Konstantní výrazy

VBScript a C# mají trochu jiná klíčová slova, jiný způsob zápisu některých literálů. Nonterminál „Nothing“ bude vždy přeložen jako „null“ – u jazyka C# nemá smysl rozlišovat. Pro konverzi vlastních hodnot použijeme následujících metod:

```
<BoolLiteral>     ::= True
        GenBoolean(True)
```

```
<BoolLiteral>     ::= False
        GenBoolean(False)
```

```
<IntLiteral>      ::= IntLiteral
        GenIntExpression([0])
```

```
<IntLiteral>      ::= HexLiteral
        GenHexExpression([0])
```

```
<IntLiteral>      ::= OctLiteral
        GenOctExpression([0])
```

```
<Nothing>         ::= Nothing
        GenNull()
```

```
<Nothing>         ::= Null
        GenNull()
```

```
<Nothing>         ::= Empty
        GenNull()
```

## 4.13 Použité metody v třídě pro překlad

Tato část blíže popisuje konverzní metody, které byly použity v předcházející části pro konverzi VBScript gramatiky.

Následující tabulka je přehledem názvů metod a objektů bázového typu CodeObject, které generují.

| Název metody | Co primárně za typ generuje |
|---|---|
| **GenBoolean** | CodePrimitiveExpression |
| **GenMemberAttribute** | CodeMemberAttributeHolder |
| **GenConstants** | *(generuje více typů)* |
| **GenConstantDecl** | CodeMemberField |
| **GenIdentifierDescriptorHolder** | CodeIdentifierDescriptorHolder |
| **GenIdentifierDescriptorHolder** | CodeIdentifierDescriptorHolder |
| **GenWithStatement** | CodeStatement |
| **GenSwitchStatement** | CodeSnippetStatement |
| **GenSwitchCaseStatement** | CodeExpression |
| **GenSwitchDefaultStatement** | CodeExpression |
| **GenFloatExpression** | CodePrimitiveExpression |
| **GenStringExpression** | CodePrimitiveExpression |
| **GenDateExpression** | CodePrimitiveExpression |
| **GenIntExpression** | CodePrimitiveExpression |
| **GenHexExpression** | CodePrimitiveExpression |
| **GenOctExpression** | CodePrimitiveExpression |
| **GenerateNewVariable** | CodeVariableDeclarationStatement |
| **GenerateNewVariable** | CodeVariableDeclarationStatement |
| **GenRedimStatement** | *(generuje více typů)* |
| **GenRedimDeclaration** | CodeArrayDimensionChangeDeclarationExpression |
| **GenLeftExprTransform** | CodeMethodInvokeExpression |
| **GenMethodInvokeExpression** | CodeMethodInvokeExpression |
| **GenNewMethod** | *(generuje více typů)* |
| **newline** | *(generuje více typů)* |
| **GenBinaryOperatorType** | CodeBinaryOperatorExpression |
| **GenIfStatement** | CodeConditionStatement |
| **GenLoopStatement** | CodeIterationStatement |
| **GenLoopStatement** | CodeIterationStatement |
| **GenLoopStatement** | CodeIterationStatement |
| **GenLoopType** | CodeLoopTypeExpression |
| **GenForStatement** | CodeIterationStatement |
| **GenAssignStatement** | CodeAssignStatement |
| **GenNegationExpression** | CodeBinaryOperatorExpression |
| **GenExitStatement** | *(generuje více typů)* |
| **GenNull** | CodePrimitiveExpression |
| **GenArgModifierOptType** | CodeArgModifierOptExpression |
| **GenArg** | CodeArgExpression |

**Tabulka 4.1 Použité metody a typ návratu**

### 4.13.1 GenBoolean

Metoda vrátí výraz s primitivní hodnotou typu Boolean.

### 4.13.2 GenMemberAttribute

Na základě předaného typu (puplic nebo private) vygeneruje na výstup instanci třídy CodeMemberAttributeHolder s daným typem. Parametr strAccessAttribute tedy musí být typu string a musí být převoditelný na hodnotu enumerátoru typu MemberAttributes.

#### 4.13.3 GenConstants

Podle předaného modifikátoru přístupu (public, private,…) vygeneruje seznam konstant. Protože je možné, že každá konstanta bude mít jiný typ, musí se výpis každé konstanty provést zvlášť. Modifikátor však budou mít společný. Není proto jiný způsob, než předat modifikátor přístupu dál. Generování typu bude probíhat ze samotné deklarace konstanty, neb již podle svého určení nelze za chodu aplikace měnit. Konstanty se předpokládají buď boleanovské, číselné, nebo řetězcové. Jiné typy nejsou povoleny.

#### 4.13.4 GenConstantDecl

Zajistí vytvoření nové deklarace konstanty. Konstanty jsou deklarovány jako CodeMemberField dané třídy s atributem Const.

#### 4.13.5 GenIdentifierDescriptorHolder

Vytvoří nový popis identifikátoru (CodeIdentifierDescriptorHolder) podle předaného terminálu. Kontext (zda odkaz na metodu nebo proměnnou) je zpracován o úroveň výš.

#### 4.13.6 GenIdentifierDescriptorHolder

Vytvoří nový popis rozšířeného identifikátoru (CodeIdentifierDescriptorHolder) podle předaných parametrů. První parametr typu terminál ukazuje na název hlavního identifikátoru, další parametr (nonterminál) je odkazovaný (rozšířený) identifikátor. Tato metoda tedy vytváří "vnořené" identifikátory typu Response.Write. Kontext (zda odkaz na metodu nebo proměnnou) je zpracován o úroveň výš.

#### 4.13.7 GenWithStatement

Vygeneruje WITH příkaz.

#### 4.13.8 GenSwitchStatement

Vygeneruje SWITCH příkaz (původní case select). Jelikož CodeDOM nepodporuje SWITCH, vygeneruje se instance typu CodeSnippetStatement. Pro případ, že by se generovalo i do jiných jazyků, než je C#, je nutné převést na posloupnost IF příkazů.

#### 4.13.9 GenSwitchCaseStatement

Generuje jeden CASE výraz příkazu switch.

#### 4.13.10 GenSwitchDefaultStatement

Generuje DEFAULT výraz příkazu switch.

#### 4.13.11 GenFloatExpression

Vygeneruje reprezentaci FLOAT čísla, tedy čísla s desetinou čárkou.

#### 4.13.12 GenStringExpression

Vygeneruje reprezentaci řetězce.

#### 4.13.13 GenDateExpression

Vygeneruje reprezentaci datumu.

#### 4.13.14 GenIntExpression

Vygeneruje reprezentaci celého čísla.

#### 4.13.15 GenHexExpression

Vygeneruje hexadecimální reprezentaci čísla (šestnáctková soustava).

#### 4.13.16 GenOctExpression

Vygeneruje oktadecimální reprezentaci čísla (osmičková soustava).

#### 4.13.17 GenerateNewVariable

Metoda generuje deklaraci proměnné nebo deklaraci pole. Proměnná (nebo pole) může být lokální nebo globální. Pokud se jedná o globální proměnnou (nebo pole), bude namísto deklarace proměnné (nebo pole) vytvořena deklarace nové vlastnosti (property).

#### 4.13.18 GenRedimStatement

Tato metoda generuje příkaz znovudeklarace pole. V jazyce VBasic je tato operace realizována pomocí klíčového slova "Redim" nebo "Redim Preserve", v jazyce C# je syntaxe následující: "A = new Type[X]" nebo "Array.Copy(A, A = new Type[X], X)".

#### 4.13.19 GenLeftExprTransform

Generuje levé přiřazení. Kontextově může znamenat buď přiřazení do prvku pole, nebo přiřazení do výsledku volání funkce. Záleží na stavu zásobníku s definicemi ´používaných symbolů.

#### 4.13.20 GenMethodInvokeExpression

Vygeneruje odkaz na použití metody. Po dokončení první fáze konverze je nutné doplnit přesné jméno odkazované metody.

#### 4.13.21 GenNewMethod

Vygeneruje deklaraci nové globální metody v dané třídě. Metoda bude mít vždy příznak public, aby mohla být přístupná z ASPX stránek, které na danou třídu odkazují. Název metody včetně umístění se uloží na zásobník symbolů.

#### 4.13.22 GenBinaryOperatorType

Metoda zpracuje výraz A x B, kde x je binární operátor (pracuje se dvěma operandy). Příklad výrazů: "A == B", "A + B". Operátor (parametr tOperator) musí být řetězec odpovídající výčtové hodnotě typu CodeBinaryOperatorType (např. "Add").

#### 4.13.23 GenIfStatement

Vygeneruje řídící příkaz typu IF. Povinné parametry jsou tCondition (podmínka IF příkazu - musí se vygenerovat jako CodeExpression) a tTrueStmt (Příkazy, které se provedou, je-li splněna podmínka - musí se vygenerovat jako CodeStatementCollection). Parametr tFalseStmt povinný není - buď může být null, nebo se může jednat o prázdný nonterminál (v opačném případě se musí vygenerovat jako CodeStatementCollection).

#### 4.13.24 GenLoopStatement

Vygeneruje příkaz typu cyklus. Jelikož CodeDOM podporuje pouze iterační cykly, všechny for/do/while/loop cykly převede právě na FOR cyklus.

#### 4.13.25 GenLoopType

Vygeneruje typ cyklu (while vs until).

#### 4.13.26 GenForStatement

Vygeneruje příkaz typu cyklus typu "for".

#### 4.13.27 GenAssignStatement

Vygeneruje příkaz přiřazení. Povinné parametry jsou tLeft (výraz, do kterého se přiřazuje) a tRight (výraz, ze kterého se přiřazuje).

#### 4.13.28 GenNegationExpression

Metoda zpracuje a vrátí negaci výrazu. Výraz jako "! X" není podporován specifikací CodeDOM. Alternativním řešením je výraz jako "X == false".

#### 4.13.29 GenExitStatement

Metoda zpracuje a vrátí příkaz přerušení některé rutiny. Může to být například "return" uvnitř metody nebo "break" cyklu atd.

#### 4.13.30 GenNull

Metoda vrátí výraz s primitivní "null" hodnotou.

#### 4.13.31 GenArgModifierOptType

Vygeneruje typ ArgModifierOpt (ByVal vs ByRef).

#### 4.13.32 GenArg

Tato metoda zpracuje a vrátí výraz argumentu (např. parametr funkce, tj. nonterminal <Arg>).

# Návrh řešení (5)

Tato část popisuje návrh řešení na základě analýzy a ostatních podkladů.

## 5.1 Automatizovaný překlad

Cílem implementace je vytvořit aplikaci, umožňující automatizovaný překlad.

Tato aplikace bude fungovat na následujícím principu:

- Uživatel vytvoří nový projekt, určí zejména
  - Zdrojový adresář obsahující ASP aplikaci
  - Cílový adresář, kam se budou generovat konvertované soubory
  - Konverzní balíček, který má být při konverzi použit (viz dále)
- Aplikace následně umožní spuštění konverze
  - Uživatel otevře projekt a stiskne tlačítko „provést konverzi“
  - Další možností je dávkové spuštění přes příkazovou řádku
- Probíhá konverze. Pokud nelze určit, jak daný aspekt cílového jazyka konvertovat, výskyt případu se „zapamatuje“. Toto se týká zejména:
  - Neznámá třída
  - Neznámá vlastnost třídy
  - Neznámá metoda
  - Nelze určit typ proměnné
  - Nelze určit typ návratu metody
- Po skončení prvního běhu konverze se zobrazí přehled nevyřešených aspektů.
- Uživatel je vyzván k vyřešení těchto aspektů
  - Zadá správné jméno třídy
  - Zadá správnou vlastnost třídy
  - Určí název metody
  - Určí typ proměnné
  - Určí typ návratu metody
- Při zadávání bude uživatel moct zvolit, zda se úprava týká
  - daného (ASP) souboru (např. typ návratu lokální metody)
  - celého projektu (např. typ globální proměnné, konstanty,..)
  - obecně překladu daného jazyka (např. Response.Write, převod Server.CreateObject(„FileSystem“),...)
  - nebo nebude aspekt řešen (tedy „Neřešit – chyba v ASP, opravím v ASP“).
- Soubor/projekt se ukládá do projektového souboru, globální nastavení se ukládají do globálního nastavení programu.
- Konverze půjde spustit opakovaně. Pro již jednou vyřešené záležitosti nebude třeba znovu zadávat řešení (Odpovědi uživatele se budou ukládat)

Samotnou konverzi bude řídit konverzní balíček. Tento bude obsahovat gramatiku zdrojového jazyka (ASP), konverzní pravidla a DLL s třídami poskytující další metody konverze.

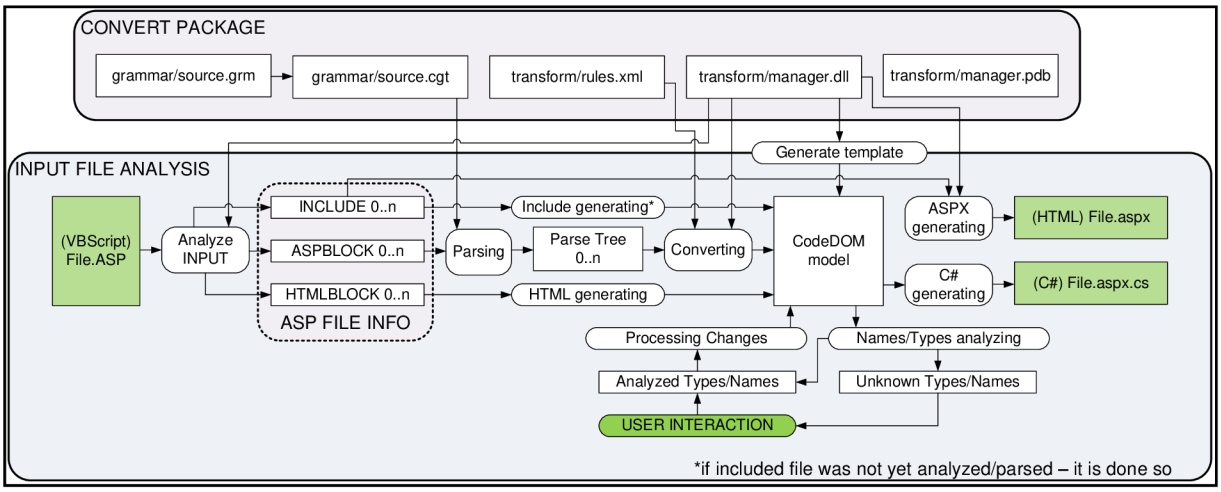
Kromě aplikace, která by spouštěla automatický překlad, bude nutné vytvořit další dvě pomocné aplikace. Aplikaci na vytváření a editaci konverzních balíčků a aplikaci, která by umožňovala konverzní balíčky testovat.

Cílem je tedy navrhnout a implementovat následující (v pořadí implementace)

- Aplikaci na editaci konverzního balíčku. **Convert Package Editor**
- Aplikaci na testování konverzního balíčku: **Convert Package Tester**
- Konverzní balíček: **asp2aspx.zip**
- Aplikaci na spouštění automatické konverze: **Code Converter Studio**

## 5.2 Průběh konverze jednoho ASP souboru

Průběh procesu je zobrazen na Obr 5.1 Průběh konverze jednoho ASP souboru.




**Obr 5.1 Průběh konverze jednoho ASP souboru**

Konverze jednoho souboru probíhá následovně:

- Analýza souboru (Analyze INPUT)
- Vytvoření výstupní šablony (Generate template)
- Analýza/generování odkazů pro závislé soubory (Include generating)
- Syntaktická analýza a konverze bloků ASP (Parsing, Converting)
- Generování HTML bloků (HTML generating)
- Analýza proměnných, metod (Names/Types analyzing)
- Interakce s uživatelem (USER INTERACTION)
- Úprava CodeDOM - zpětná vazba (Processing Changes)
- Finální generování primárního výstupu (C# generating)
- Generování sekundárních výstupů (ASPX generating)

> Bloky „Include generating“, „Parsing, Converting“ a „HTML generating“ probíhají v tom pořadí, v jakém se nacházejí ve zdrojovém souboru ASP!

### 5.2.1 Analýza souboru (Analyze INPUT)

Nejprve se soubor analyzuje – vyhledají se následující části:

- Odkazy na jiné soubory (INCLUDE)
- Bloky ASP kódu (ASPBLOCK)
- Bloky všeho ostatního – zejména tedy HTML (HTMLBLOCK)

Analýza souboru probíhá voláním metody TransformManagerBase.AnalyzeInputFile (tato metoda je virtuální, může být přepsána v konverzním balíčku).

Obecně je analýza prováděna konečným automatem, který reaguje na HTML značky:

- Odkaz na jiný soubor je HTML značkou <!-- #include file="**název souboru**" -->
- Bloky ASP kódu je HTML značkou <%**ASP KÓD**%>
- Vše ostatní se bere jako HTML blok

### 5.2.2 Vytvoření výstupní šablony (Generate template)

Proces vytvoří šablonu primárního výstupu. Původní šablona se generuje ve virtuální metodě PrepareFromTemplate třídy OutputGeneratorBase, nicméně její potomek v konverzním balíčku může tuto metodu přepsat. Šablona má za cíl určit pevnou strukturu daného výstupního souboru. Danou strukturu musí splňovat všechny vygenerované soubory.

Jelikož generujeme ASPX stránku, bude vygenerovaná šablona následující:

- Jmenný prostor šablony musí odpovídat názvu vnořeného adresáře, kde se původní ASP soubor nacházel
- Název třídy musí odpovídat názvu souboru původního ASP souboru s postfixem Class (případně jiným)
- Třída musí dědit z Web.UI.Page
- Třída musí obsahovat property Response (typ HTTPResponse) – tato je zde pro to, aby při volání tzv. „included“ stránek šel „přepsat“ výstup
- Musí obsahovat metodu Page_Load – toto je obsloužení ASPX události načtení stránky. Tato metoda bude dále volat metodu „ExecutePage“
- Bude obsahovat metodu ExecutePage – tato bude obsahovat všechen globální kód.

Pro kompletní C# výpis šablony viz příloha 10.2 - Šablona pro generování ASP C# Code Behind.

### 5.2.3 Analýza/generování odkazů pro závislé soubory (Include generating)

Tento proces zpracovává seznamu odkazovaných souborů z (5.2.1).

Každý odkazovaný soubor se

- předá ke zpracování, pokud ještě zpracován nebyl (kompletní zpracování – není-li vyžadován uživatelský vstup, mohou se dokonce soubory vygenerovat do cílového adresáře, v opačném případě se tento krok vynechává)
- připojí k výstupu (CodeDOM model) ve formě
    - Importu jeho jmenného prostoru (tj. názvu adresáře)
    - Vytvoření nové property, která se zároveň v konstruktoru třídy musí inicializovat
    - Volání metody ExecutePage()

### 5.2.4 Syntaktická analýza a konverze bloků ASP (Parsing, Converting)

Nejprve se analyzuje za pomoci zdrojové gramatiky vstupní ASP blok. Vytvořený derivační strom se zpracuje konverzními pravidly a metodami – vše se zapíše na výstup (CodeDOM model).

Detailněji se tímto problémem zabývá část 5.3 - Konverze jednoho ASP bloku.

### 5.2.5 Generování HTML bloků (HTML generating)

Generování probíhá tak, že se HTML kód z původního ASP souboru převede na řetězec, který se předá metodě Write objektu Response. Výpis se provádí po řádcích.

Příklad HTML kódu:

```
<P>this is "HTML"
code</p>
```

Výše uvedený HTML kód se převede na:

```
Response.WriteLine("<P>this is \"HTML\"");
Response.WriteLine("code</p>");
```

### 5.2.6 Analýza proměnných, metod (Names/Types analyzing)

Provádí se po dokončení generování CodeDOM výstupu. Během generování byly zaznamenávány výskyty deklarace proměnných/metod, stejně tak jako jejich volání. Nyní je tedy třeba upravit názvy, resp. odkázání volání (např. pokud je použita metoda z jiného – vloženého – souboru).

Příklad – v odkázaném souboru **GlobalMethods** je definována globální metoda **Dosomething**. Tato metoda je odkázána v právě zpracovávaném souboru. Na místě jejího volání v CodeDOM je nyní následující kód:

```
dosomething(myProperty);
```

Zároveň víme, že aktuální třída obsahuje deklaraci vlastnosti **Myproperty**. Kód tedy bude opraven následovně (oprava tučně a červeně):

```
this.GlobalMethodsInstance.Dosomething(this.Myproperty);
```

Ty výskyty (zejména odkazů), které nebyly rozpoznány, se zaznamenají a zobrazí se uživateli, jako „nevyřešené aspekty“. Více viz níže.

### 5.2.7 Interakce s uživatelem (USER INTERACTION)

Uživateli jsou zobrazeny aspekty, které nebyly vyřešeny. Tato interakce neprobíhá po analýze každého jednotlivého souboru, ale až jsou všechny soubory analyzovány.

Uživateli jsou předkládány k řešení především následující případy:

- Neznámá třída – uživatel je vyzván k opravě názvu
- Neznámá vlastnost třídy – uživatel je vyzván k opravě názvu
- Neznámá metoda – uživatel je vyzván k opravě názvu
- Nelze určit typ proměnné – uživatel je vyzván k zadání typu
- Nelze určit typ návratu metody – uživatel je vyzván k zadání typu

### 5.2.8 Úprava CodeDOM - zpětná vazba (Processing Changes)

Na základě uživatelem zadaných změn jsou tyto zapsány do výstupu (CodeDOM model).

### 5.2.9 Finální generování primárního výstupu (C# generating)

Na základě vytvořeného CodeDOM je tento za pomoci CodeDOMProvider třídy vygenerován v podobě C# kódu do cílového adresáře.

### 5.2.10 Generování sekundárních výstupů (ASPX generating)

Jako poslední krok je třída TransformManager vyzvána ke generování sekundárních souborů. Nyní se tedy vygeneruje soubor ASPX. Tento bude obsahovat popis ASPX stránky:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="FILENAME.aspx.cs"
    Inherits="FILE_CLASS_NAME" %>

<%@ Reference Page="INCLUDED_FILE_1.aspx" %>
…
<%@ Reference Page="INCLUDED_FILE_N.aspx" %>

<!-- file FILENAME.aspx; Copyrights GETMORE,s.r.o. (c) 1997-2006 All rights reserved -->
```

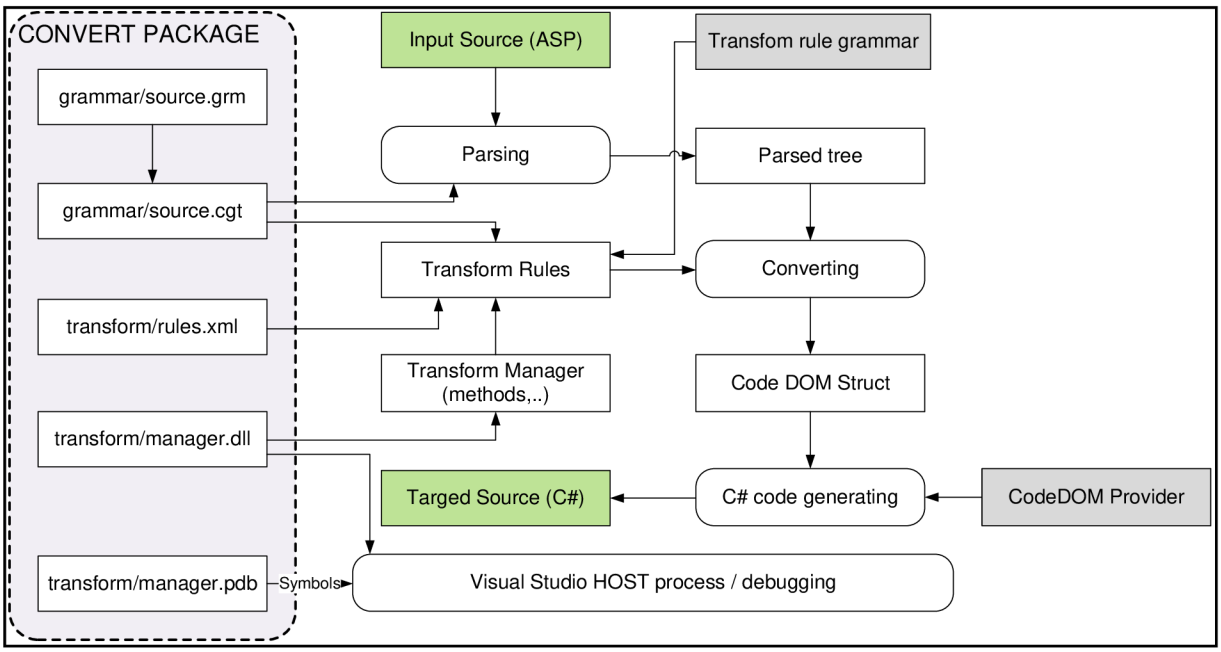
První řádek připojuje k dané stránce tzv. CodeBehind – tedy vlastní primární výstup.

Třetí až 5 řádek vypisuje odkazované soubory (původně INLCUDE).

Poslední sedmý řádek je pouze komentář s copyrightem.

## 5.3 Konverze jednoho ASP bloku

Celý proces konverze vyplývá z Obr 5.2 Schéma procesu konverze. Dále budou popsány jednotlivé části schématu.




**Obr 5.2 Schéma procesu konverze**

### 5.3.1 CONVERT PACKAGE: grammar/source.grm

Tento soubor z konverzního balíčku obsahuje gramatiku zdrojového jazyka. Tato gramatika je zapsaná v GOLD BNF syntaxi.

### 5.3.2 CONVERT PACKAGE: grammar/source.cgt

Tento soubor obsahuje kompilovanou tabulku redukčních pravidel zdrojového jazyka (Compiled Grammar Table), získanou z GRM souboru.

### 5.3.3 CONVERT PACKAGE: transform/rules.xml

Tento soubor obsahuje přiřazení derivačních pravidel ze zdrojové gramatiky a jejich transformace. Popis derivačních pravidel je v XML, transformační pravidla jsou zapsána ve vlastní syntaxi, viz 10.3 - Gramatika Transformace.

### 5.3.4 CONVERT PACKAGE: transform/manager.dll

Tento soubor obsahuje .NET kód (Assembly) s třídou Správce konverze (Transform Manager). Tato poskytuje zejména metody pro konverzi samotných pravidel.

### 5.3.5 CONVERT PACKAGE: transform/manager.pdb

Nepovinná část konverzního balíčku. Obsahuje symboly nutné pro ladění vykonávání metod Správce konverze.

### 5.3.6 Parsing

Proces, kdy se za pomoci redukčních pravidel zdrojové gramatiky převede zdrojový text na derivační strom.

### 5.3.7 Parsed Tree

Výstup procesu syntaktické analýzy. Je určen jako nejvyšší token dané gramatiky.

### 5.3.8 Transform Manager

Na základě atributů tříd a metod dynamicky nahraje ze souboru „manager.dll“ třídu, která následně poskytuje metody pro konverzi. Třída je označená atributem „`TransformClass`“, metody pak atributem „`TransformMethod`“. Volání metod probíhá při transformaci z třídy „Transform Rules“.

### 5.3.9 Transform Rule Grammar

Tato gramatika definuje syntaxi zápisu konverzních pravidel. Je součástí knihovny CodeConvertLib.Core (soubor CGT připojen jako „resource“). Také viz 10.3 - Gramatika Transformace.

### 5.3.10 Transform Rules

Třída řídící samotnou konverzi. Načte pravidla z XML souboru a každé transformační pravidlo spojí s původním redukčním pravidlem z tabulky redukčních pravidel (podle názvu pravidla a symbolů, které se redukují). Dále se již hledání pravidel řídí podle ID pravidla gramatiky.

Zároveň se jednotlivá transformační pravidla přeloží za pomoci gramatiky konverzních pravidel (Transform Rule Grammar) do jejich objektové podoby. Pokud je v pravidle volána konverzní metoda, tato se vyhledá ve třídě Transform Manager.

### 5.3.11 Converting

Výsledný nonterminál derivačního stromu se předá konverzním pravidlům. Po té je postupně shora dolů procházen celý strom – u každého symbolu tohoto stromu je uvedeno, jakým redukčním pravidlem byl symbol vytvořen. Pokud má toto redukční pravidlo asociováno transformaci, je tato spuštěna. Transformací myslíme zejména spuštění odpovídající metody/metod z třídy Transform Manager. Tyto metody pak vytvářejí struktury typu CodeObject, ze kterých je tvořen strom CodeCompileUnit.

### 5.3.12 Code DOM Struct

Je výsledkem konverze. Jedná se o strom objektů typu CodeObject. Struktura stromu je přesně definována a odpovídá DOM struktuře jazyků. Příklady objektů: CodeNameSpace, CodeTypeDeclaration,… Toto se týká zejména System.CodeDOM.

### 5.3.13 CodeDOM Provider

Poskytovatel služeb konkrétního jazyka pro generování a kompilaci z Code DOM struktur. Existují poskytovatelé např. pro jazyk C# (Microsoft.CSharp.CSharpCodeProvider), VB.NET (Microsoft.VisualBasic.VBCodeProvider) a další. Toto se týká například Microsoft.

### 5.3.14 C# Code Generating

Proces, kdy poskytovatel služeb generuje ze stromu CodeDOM syntaxi cílového jazyka.

## 5.4 Konverzní balíček

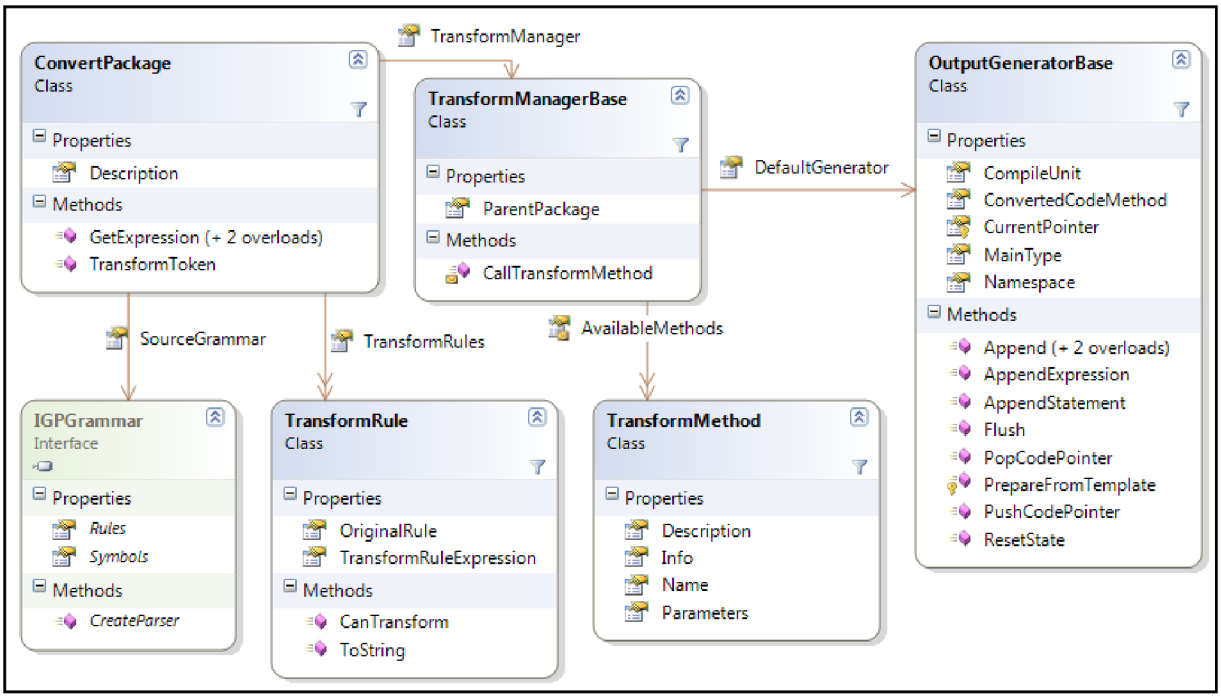
Jak již bylo v předešlém textu několikrát naznačeno, celý proces konverze je založen na konverzním balíčku.

Konverzní balíček je ZIP archív, obsahující následující soubory:

- grammar/source.grm
- grammar/source.cgt
- transform/rules.xml
- transform/manager.dll
- transform/manager.pdb

Význam a obsah jednotlivých souborů byl již popsán dostatečně v 5.3 - Konverze jednoho ASP bloku. Nyní se tedy zaměřme na jejich objektovou prezentaci.

Celý konverzní balíček je prezentován třídou ConvertPackage viz Obr 5.3 Diagram tříd Konverzního balíčku a závislostí. Pro zjednodušení jsou zobrazeny pouze relevantní metody a vlastnosti.




**Obr 5.3 Diagram tříd Konverzního balíčku a závislostí**

### 5.4.1 ConvertPackage

ConvertPackage zastřešuje celou konverzi. Poskytuje zejména zdrojovou gramatiku (**SourceGrammar**), konverzní pravidla (**TransformRulers**) a správce konverze (**TransformManager**) společně s primárním generátorem výstupu (**DefaultGenerator**).

Důležitá je zejména metoda **TransformToken**, která spustí konverzi nad předaným symbolem. Generování CodeDOM se následně provádí primárním generátorem výstupu.

Metoda GetExpression vrátí CodeDOM objekt konkrétního předaného symbolu a jeho pod-symbolů. Je použitá zejména při rekurzivním zpracování vstupu konverzními pravidly.

### 5.4.2 TransformManagerBase

Poskytuje zejména:

- Metody pro konverzi a jejich volání (CallTransformMethod). Metody jsou získávány přes reflection ze „sebe sama“.
- Primární generátor výstupu konverze (DefaultGenerator)

Pokud je v manager.dll přítomna třída s atributem „`TransformClass`“ a dědící z této třídy, je použita namísto této obecné implementace (vlastnost TransformManager). O toto se stará TransformManagerWrapper (viz dále).

Metody této (a bázové) třídy s atributem „`TransformMethod`“ jsou pak umožněny použít jako konverzní metody (jsou obsaženy v property AvailableMethods a přístupné voláním CallTransformMethod).

### 5.4.3 TransformRule, resp. TransformRules

Jsou odrazem uloženého rules.xml. Každé pravidlo (TransformRule) se váže na původní derivační pravidlo zdrojové gramatiky (property OriginalRule). Pokud je na toto pravidlo při procházení derivačního stromu naraženo, provedou se příkazy výrazu konverzního pravidla (TransformRuleExpression). O konverzi symbolů se „stará“ třída TransformRules a její metoda TransformToken (lze volat z třídy ConvertPackage).

### 5.4.4 OutputGeneratorBase – generování CodeDOM

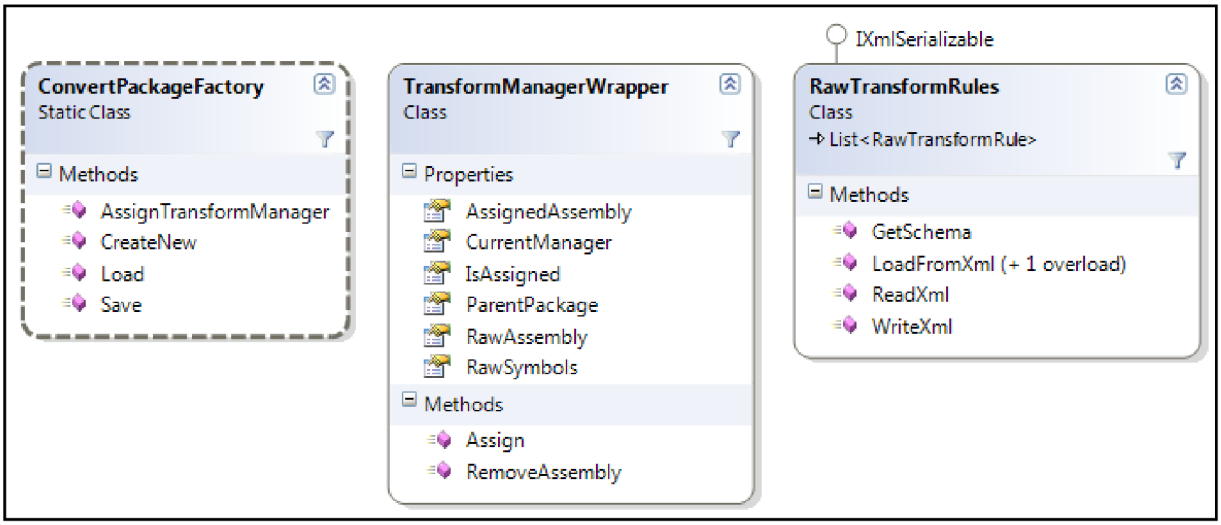
Třída je využívána zejména jako generátor CodeDOM. Instance v sobě obsahuje strukturu CodeDOM. Metody s prefixem **Append** připojují či nahrazují CodeDOM objekty na to místo, kam ukazuje property **CurrentPointer**. Tato ukazuje po vytvoření instance na konkrétní místo ve vytvořené šabloně (kam to přesně je, záleží právě na šabloně, která může být přepsána v potomcích této třídy – v bázové třídě je to nicméně property ConvertedCodeMethod). **CurrentPointer** se nicméně může během generování **měnit**! To umožňují metody PushCodePointer a PopCodePointer (na nastavení nového resp. obnovení starého CurrentPointer).

Generátor tak umožňuje následující:

- Generovat CodeDOM do **předem určeného** místa ve struktuře
- Generovat CodeDOM do **jiného místa** ve struktuře (deklarace globálních metod,…)
- Generovat CodeDOM **nezávisle na generované struktuře** (důležité zejména při generování „vnořených“ výrazů typu podmínka v IF cyklu,…)

### 5.4.5 Další důležité třídy

Na následujícím obrázku je diagram tříd, které jsou důležité při práci s konverzním balíčkem.




**Obr 5.4 Další důležité třídy v CodeConvertLib.Core**

**CodeConvertPackageFactory** – tato třída se kompletně stará o vytváření, ukládání, otevírání atd. konverzních balíčků. Smysl jednotlivých metod je zřejmý.

**TransformManagerWrapper** – je důležitý zejména pro asociaci, resp. odebrání externí DLL knihovny s balíčkem. Pokud není žádná externí DLL přiřazena, je v property Currentmanager přístupná instance třídy TransformManagerBase. V opačném případě je zde přístupná instance třídy nahrané z externí DLL knihovny (potomek třídy TransformManagerBase).

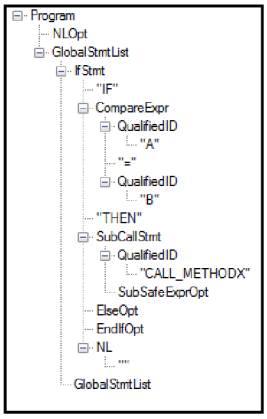
**RawTransformRules** – představuje soubor rules.xml. Slouží k nahrávání a ukládání z/do XML proudu (XmlReader/XmlWriter). Obsahuje pouze jednoduché textové vyjádření pravidel. Tuto třídu poté využívá jako datový zdroj třída TransformRules. Tato již má vše (včetně pravidel, parametrů,…) vyjádřeno odpovídajícím typem objektu.

## 5.5 Ukázka konverze – příklad

Na následujícím příkladu si ukážeme, jak vytvářet CodeDOM model z řetězce převedeného na derivační strom za pomoci konverzních pravidel a metod. Zejména bude naznačeno, jak je řešen **kontext** jazyka. V příkladu používám podmnožinu VBScript gramatiky a k ní odpovídající konverzní pravidla. Pro zápis konverzních pravidel používám syntaxi viz 4.1 - Zápis konverzních pravidel.

### 5.5.1 Zdrojový text, derivační strom

Uvažujme tedy zdrojový text „IF A=B THEN CALL_METHODX“. Tento text bude převeden na derivační strom viz následující obrázek (Obr 5.5 Derivační strom pro „IF A=B THEN CALL_METHODX“). Derivační strom není úplný dle gramatiky VBScript, slouží pouze pro ukázku. Ve skutečnosti obsahuje několik desítek nonterminálů navíc (vzniklých použitím jednoduchých přepisovacích pravidel).




**Obr 5.5 Derivační strom pro „IF A=B THEN CALL_METHODX“**

### 5.5.2 Konverzní pravidla

I bez znalosti původní VBScript gramatiky můžeme definovat následující konverzní pravidla[1]:

```
1.  <Program>(<NLOpt><GlobalStmtList>) = DEFAULT
2.  <IfStmt>("IF"<CompareExpr>"THEN"<SubCallStmt><ElseOpt><EndIfOpt><NL>) = GenIF([1],[3],[4])
3.  <NL>() = NOT_SUPPORTED /*zde by se transformace nem2la nikdy ocitnout*/
4.  <CompareExpr>(<QualifiedID>"="<QualifiedID>) = GenBinOp([0],"Equal",[2])
5.  <SubCallStmt>(<QualifiedID><SubSafeExprOpt>) = GenMethodCall([0])
6.  <GlobalStmtList>() = NOTHING
7.  <QualifiedID>(ID) = GenReference([0])
```

Poznámka: V derivačním stromu je text bez uvozovek nonterminál, text s uvozovkami je terminál – ve všech případech se jedná o terminál definovaný jako **ID**.

### 5.5.3 Definice metody GenIF(condition, trueStmt, elseStmt):

```
1.  Vytvoř CodeConditionalObject.
CodeConditional.Condition = Vrať_výraz_z_nonterminálu(condition)
CodeConditional.TrueStatement = Vrať_výraz_z_nonterminálu(trueStmt)
Pokud elseStmt není prázdné
{
CodeConditional.ElseStatement = Vrať_výraz_z_nonterminálu(elseStmt)
}
Zapiš CodeConditionalObject na výstup
```

[1] Použitá syntaxe je jednoduchá – *derivovaný symbol* (*původní symboly*) = *konverzní výraz*

### 5.5.4 Definice metody Vraťˇ_výraz_z_nonterminálu(token):

```
1. Vytvoř Výraz
2. Přesměruj výstup na Výraz
3. Zavolej konverzi pro terminál token
4. Přesměruj výstup zpátky na předchozí
5. Návratová hodnota této metody bude Výraz
```

### 5.5.5 Definice metody GenBinOp(left, op, right):

```
1. Vytvoř CodeBinaryOperationObject
2. CodeBinaryOperationObject.Left = Vraťˇ_výraz_z_nonterminálu(left)
3. CodeBinaryOperationObject.Right = Vraťˇ_výraz_z_nonterminálu(right)
4. CodeBinaryOperationObject.Operator = op
5. Zapiš CodeBinaryOperationObject na výstup
```

### 5.5.6 Definice metody GenReference(ID)

```
1. Vytvoř CodeVarReferenceObject
2. CodeVarReferenceObject.Name = ID.Text
3. Zapiš CodeVarReferenceObject na výstup
```

### 5.5.7 Definice metody GenMethodCall(mName)

```
1. Vytvoř CodeMethodInvoke
2. Vytvoř CodeVarReferenceObject
3. CodeVarReferenceObject = Vraťˇ_výraz_z_nonterminálu(mName)
4. CodeMethodInvoke.MethodName = CodeVarReferenceObject.Name
5. Zapiš CodeMethodInvoke na výstup
```

- **Poznámka k této metodě**: toto je ukázka, jak obejít nutnost znát kontext. Potřebujeme totiž při konverzi symbolu QualifiedID, aby právě v závislosti na kontextu jednou vrátila „referenci na proměnnou“ a podruhé „referenci na metodu“. V našem jednoduchém případě lze toto obejít tak, že při konverzi QualifiedID (voláním metody QualifiedID) se vždy vytvoří reference na proměnnou (tedy nezávisle na kontextu), ovšem při konverzi volání metody GenMethodCall se s tímto počítá – viz řádek 2 – 3. Reference na proměnnou se prostě nepoužije jako část výstupu, ale získá se z ní pouze její název (který odpovídá názvu metody).
- **Jiný způsob, jak získat kontext**: přesměrování výstupu generátoru (CodePointer) je ukládáno na zásobník. Je tedy relativně jednoduché zjistit, jaký je požadovaný výstup a podle toho vytvořit odpovídající typ objektu.

### 5.5.8 Proces konverze

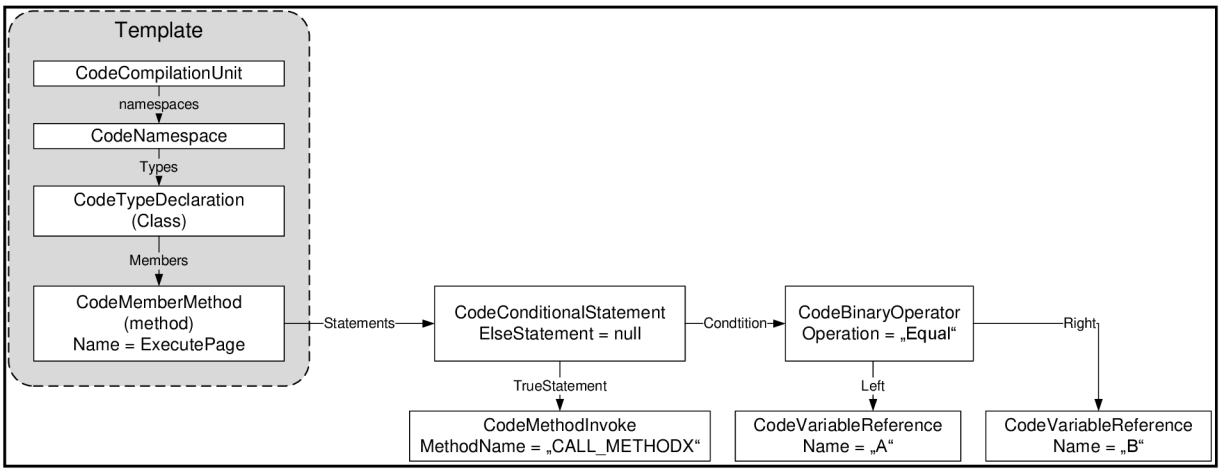
Po předložení prvního symbolu metodě pro konverzi bude proces probíhat následovně:

- Zjistí se, že pro derivaci <Program>(<NLOpt><GlobalStmtList>) existuje konverzní pravidlo (1). Toto má příznak nastaven na DEFAULT. Konverze se tedy chová, jakoby žádné pravidlo definováno nebylo – tedy zpracují se všechny původní symboly a provede se konverze pro každý jednotlivý symbol (konkrétně tedy <NLOpt><GlobalStmtList>).
- Volá se konverze pro derivaci <NLOpt>(). Pro tuto derivaci neexistuje konverzní pravidlo, neexistují žádné původní symboly, neprovede se tedy nic.
- Volá se konverze pro derivaci <GlobalStmtList>(<IfStmt><GlobalStmtList>). Pro tuto derivaci také neexistuje pravidlo (pozor, konverzní pravidlo číslo 8 se váže na jiné derivační pravidlo), konverze pokračuje pro každý původní symbol (<IfStmt><GlobalStmtList>).

- Nyní se pokračuje derivací <IfStmt>(…). Pro tu existuje konverzní pravidlo (2). Proto se nyní zavolá metoda **GenIF** s parametry <CompareExpr>, <SubCallStmt> a <ElseOpt> (předanými jako část stromu)
  - Řádek 1: vytvoří CodeDOM objekt typu podmíněný příkaz
  - Řádek 2: nejdříve se <CompareExpr> předá metodě **Vrať_výraz_z_nonterminálu**. Ta provede nejdříve přesměrování výstupu („ukradne si jej“), následně do svého výstupu (proměnná Výraz) nechá vygenerovat konverzi předaného parametru, obnoví původní výstup a vygenerovaný výraz vrátí. Proběhne tedy **rekurzivní volání procesu konverze** – v dalším bodě tedy pokračuji tímto rekurzivním voláním.
- Metoda **GenIF** zavolala na řádku 2 metodu **Vrať_výraz_z_nonterminálu**, která opětovně zavolala konverzi nad symbolem <CompareExpr>:
  - Pro tuto derivaci <CompareExpr>(<QualifiedID>"="<QualifiedID>) existuje konverzní pravidlo číslo 4. Proto se nyní zavolá metoda **GenBinOp**.
    - Řádek 1: vytvoří se CodeDOM objekt typu binární operace. Ten představuje operaci nad dvěma objekty. Operace může být například „rovno“, „nerovno“,…
    - Řádek 2: situace se opakuje a opět se volá metoda **Vrať_výraz_z_nonterminálu**.
- Metoda **GenBinOp** zavolala na řádku 2 metodu **Vrať_výraz_z_nonterminálu**, která opětovně zavolala konverzi nad symbolem <QualifiedID>:
  - Pro tuto derivaci <QualifiedID>(ID) existuje konverzní pravidlo číslo 7. Proto se nyní zavolá metoda **GenReference**.
    - Tato metoda z předaného terminálu („A“) vytvoří odkaz na proměnnou (CodeVarReferenceObject) a uloží ji na výstup
  - Metoda **GenReference** byla úspěšně dokončena, konverze <QualifiedID> byla dokončena.
  - **Vrať_výraz_z_nonterminálu** byla také úspěšně dokončena, výstup byl obnoven na předešlou hodnotu (což je ovšem v tuto chvíli pouze jiný objekt vytvořený jiným voláním metody **Vrať_výraz_z_nonterminálu**)
- Byl proveden návrat z **Vrať_výraz_z_nonterminálu** do metody **GenBinOp** na řádek 2. Do vlastnosti Left objektu CodeBinaryOperationObject byl přiřazen výraz CodeVarReferenceObject, obsahující odkaz na proměnnou „A“.
- Další příkaz je 3.řádek metody **GenBinOp**, přiřazení Right vlastnosti objektu CodeBinaryOperationObject z návratové hodnoty **Vrať_výraz_z_nonterminálu**. Vzhledem k tomu, že je postup naprosto stejný, jako u řádku 2, nebude zde tento postup vypsán. Z derivačního stromu následně vyplývá, že do vlastnosti Right bude přiřazen objekt CodeVarReferenceObject, obsahující odkaz na proměnnou „B“.
- GenBinOp, Řádek 4: byla předána textová hodnota „Equal“, která bude přiřazena do dané vlastnosti.
- GenBinOp, Řádek 4: CodeBinaryOperationObject bude zapsán na výstup.
  - Metoda GenBinOp byla úspěšně dokončena, konverze <CompareExpr> byla dokončena.
  - **Vrať_výraz_z_nonterminálu** byla také úspěšně dokončena, výstup byl obnoven na předešlou hodnotu (což je nyní zcela původní hodnota)
- Byl proveden návrat z **Vrať_výraz_z_nonterminálu** do metody **GenIF** na řádek 3. Do vlastnosti Condtition objektu CodeConditional byl přiřazen výraz CodeBinaryOperationObject, obsahující jako Left výraz odkaz na proměnnou „A“, jako Right výraz odkaz na proměnnou „B“ a jako operaci „Equal“.
- Podobně se v metodě postupuje dále, dokud tato není dokončena a na výstup zapsán CodeConditionalObject
- Nyní se pokračuje v konverzi další derivace v řadě a tou je <GlobalStmtList>(). Pro tuto derivaci je definované konverzní pravidlo číslo 6. Jeho typ je NOTHING. Neprovede se tedy nic.

- Konverze derivace <GlobalStmtList>(<IfStmt><GlobalStmtList>) je skončena, stejně jako derivace <Program>(<NLOpt><GlobalStmtList>). Konverze původního symbolu <Program> tak byla ukončena.

Po dokončení výše uvedeného procesu je vygenerován následující CodeDOM výstup:

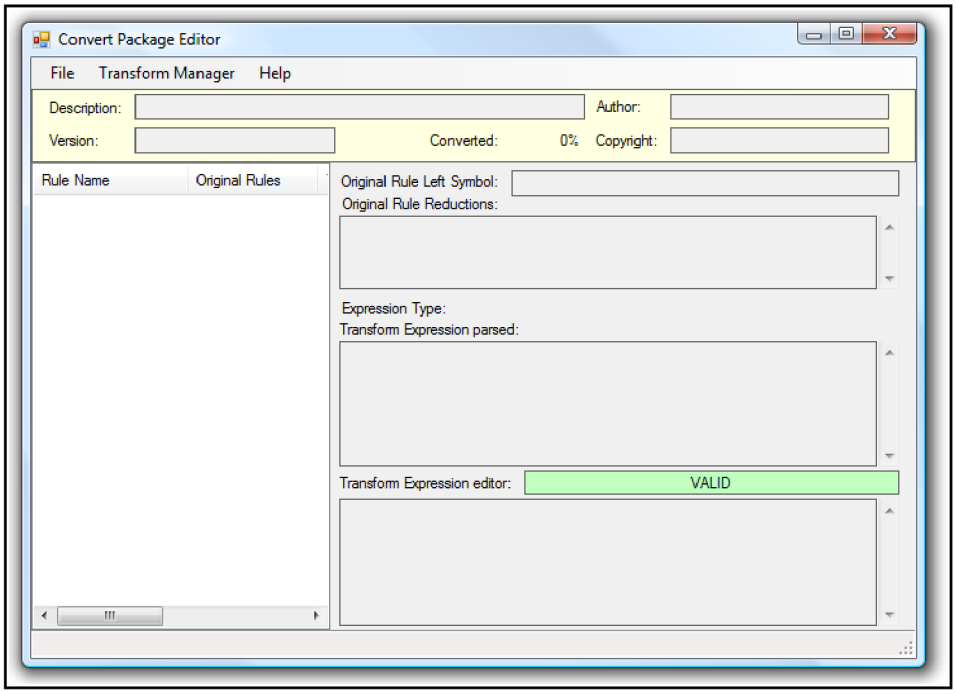



**Obr 5.6 ukázkový příklad - výsledná CodeDOM struktura**

Poznámka k obrázku: CodeDOM struktura šablony (šedá část „Template“) pouze naznačena – důležité jsou struktury, které byly vygenerovány – CodeConditionalStatement, aj.

## 5.6 Convert Package Editor

Aplikace slouží na editaci konverzních balíčků. Po spuštění se zobrazí prázdný formulář, viz Obr 5.7 Convert Package Editor - program po spuštění.

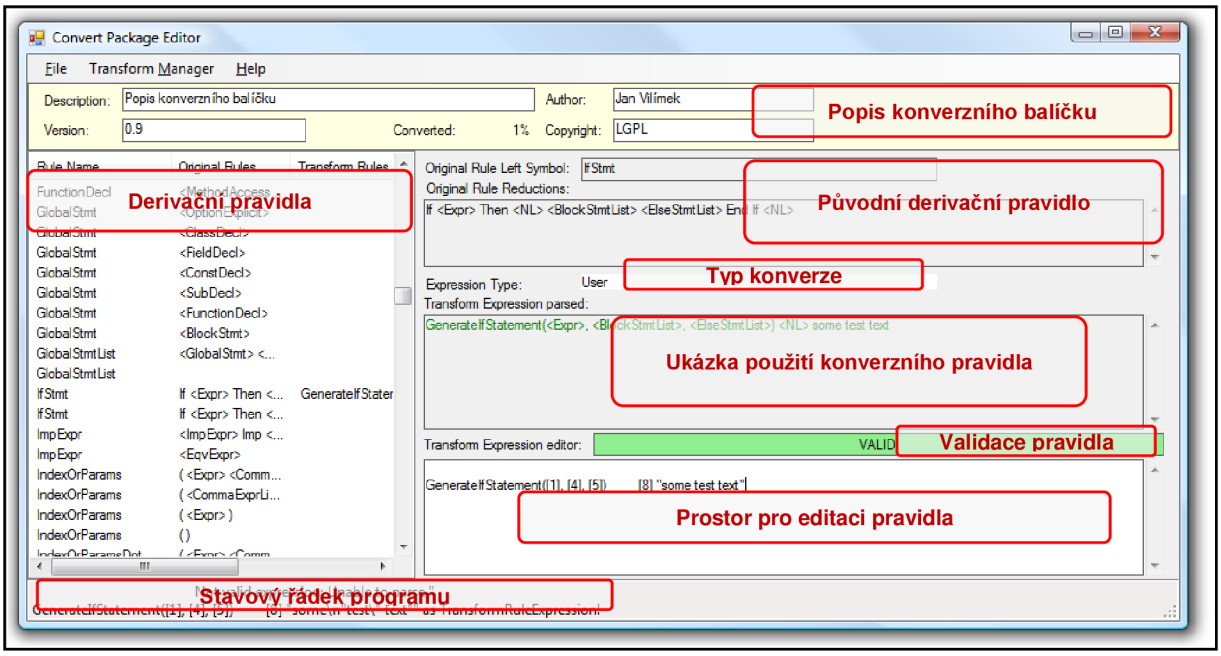



**Obr 5.7 Convert Package Editor - program po spuštění**

Je-li program spuštěn s přepínačem „**assign**“, je provedeno přiřazení DLL knihovny k balíčku (podle dalších parametrů příkazové řádky).

Balíček lze nahrát volbou „File – Open“, případně lze vytvořit nový balíček („File – Create New“). V takovém případě je ovšem nutné zadat zdrojovou gramatiku (která se do balíčku zkopíruje).

Po otevření konverzního balíčku vypadá aplikace stejně, jak lze vidět na následujícím obrázku (Obr 5.8 Convert Package Editor - otevřený balíček).




**Obr 5.8 Convert Package Editor - otevřený balíček**

### 5.6.1 Popis částí aplikace

„**Derivační pravidla**“ – zde se zobrazí přehled všech původních derivačních pravidel včetně (pokud existují) pravidel pro jejich konverzi.

„**Popis konverzního balíčku**“ – zde se zobrazí následující informace o konverzním balíčku:

- Popis
- Autor
- Verze
- Copyright
- Kolik % původních derivačních pravidel již obsahuje výraz pro konverzi (včetně výrazu typu DEFAULT)

„**Původní derivační pravidlo**“ – zde se zobrazí detail derivačního pravidla, pokud byla vybrána položka v části „Derivační pravidla“.

„**Typ konverze**“ – pokud byla v části „Derivační pravidla“ vybrána položka, zobrazí se zde typ konverze. Typy konverze mohou být:

- NoExpressionDefined – uživatel nezadal žádný výraz
- User – uživatel zadal jakýkoliv výraz kromě klíčového slova
- Default – bylo zadáno klíčové slovo DEFAULT
- NotSupported - zadáno klíčové slovo NOT_SUPPORTED
- Nothing - zadáno klíčové slovo NOTHING

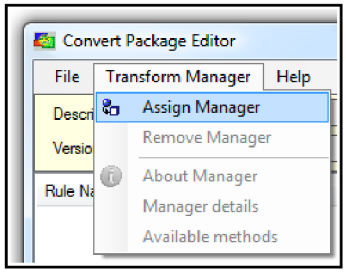
„**Ukázka použití konverzního pravidla**“ zde se text zadaný v prostoru pro editaci pravidla zobrazí tak, jak byl zpracován. Odkazované položky se nahradí původními symboly v derivačním pravidlu apod. Kontrola sémantické správnosti.

„**Validace pravidla**“ – zde se ukazuje syntaktická správnost editovaného pravidla.

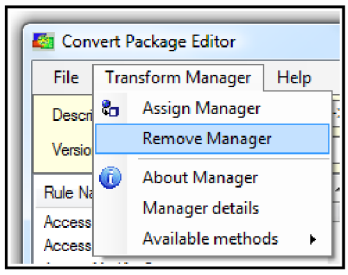
„**Prostor pro editaci pravidla**“ – zde lze editovat pravidlo. Veškeré změny jsou ihned prováděny (pokud je pravidlo syntakticky validní).

„**Stavový řádek programu**“ – zde se zobrazují eventuální chyby při validaci pravidla, jiné zprávy programu.

Program k otevřenému konverznímu balíčku umožní přiřadit (viz Obr 5.9 Convert Package Editor – přiřazení externí DLL) resp. odebrat (viz Obr 5.10 Convert Package Editor – odebrání externí DLL) externí DLL obsahující správce konverze.



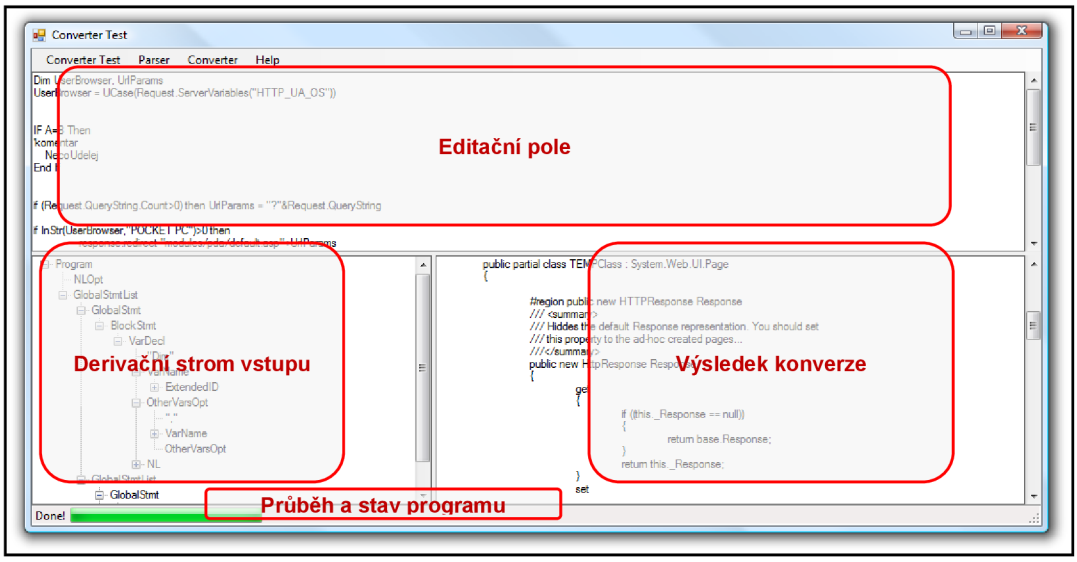

**Obr 5.9 Convert Package Editor – přiřazení externí DLL**




**Obr 5.10 Convert Package Editor – odebrání externí DLL**

## 5.7 Convert Package Tester

Aplikace slouží na testování konverzních balíčků. Po spuštění se zobrazí aplikace, viz Obr 5.11 Vzhled aplikace "Convert Package Tester".




**Obr 5.11 Vzhled aplikace "Convert Package Tester"**

### 5.7.1 Popis částí aplikace

„**Editační pole**“: zde lze zadat testovaný případ ve zdrojovém jazyce konverzního balíčku (tj. např. VBScript)

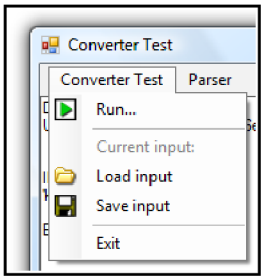
„**Derivační strom vstupu**“: po spuštění testu konverze (Converter Test – Run) nebo při spuštění testu parsování gramatiky (Parser – Run) se zde zobrazí derivační strom zdrojového textu (tedy textu z pole „editační pole“).

„**Výsledek konverze**“: na základě derivačního stromu se v tomto poli vygeneruje výsledný zdrojový kód výstupního jazyka (tj. např. C#).

„**Průběh a stav programu**“: V tomto stavovém řádku se zobrazují informace o průběhu analýzy/konverze,…

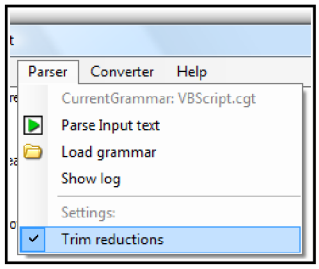
### 5.7.2 Menu aplikace

V první volbě hlavního menu lze spustit test konverze („Run…“), eventuálně nahrát soubor do vstupu editačního pole (volba „Load Input“, jakákoliv přípona, implicitně TXT), případné provedené změny lze uložit volbou „Save input“. Program lze ukončit volbou „Exit“.



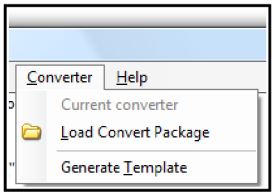

**Obr 5.12 Convert Package Tester - menu Converter Test**

Druhá volba hlavního menu se týká syntaktické analýzy a gramatiky použité při této analýze. Tato část je oddělena od samotného testování konverze. Lze nahrát libovolnou gramatiku (volba „Load grammar“), lze spustit analýzu textu z editačního pole (volba „Parse Input text“), lze zobrazit log zapsaný v průběhu analýzy („Show log“) a lze nastavit, zda se mají v derivačním stromu zkracovat pravidla, pokud pouze přepisují jedno na druhé („Trim reductions“).




**Obr 5.13 Convert Package Tester - menu Parser**

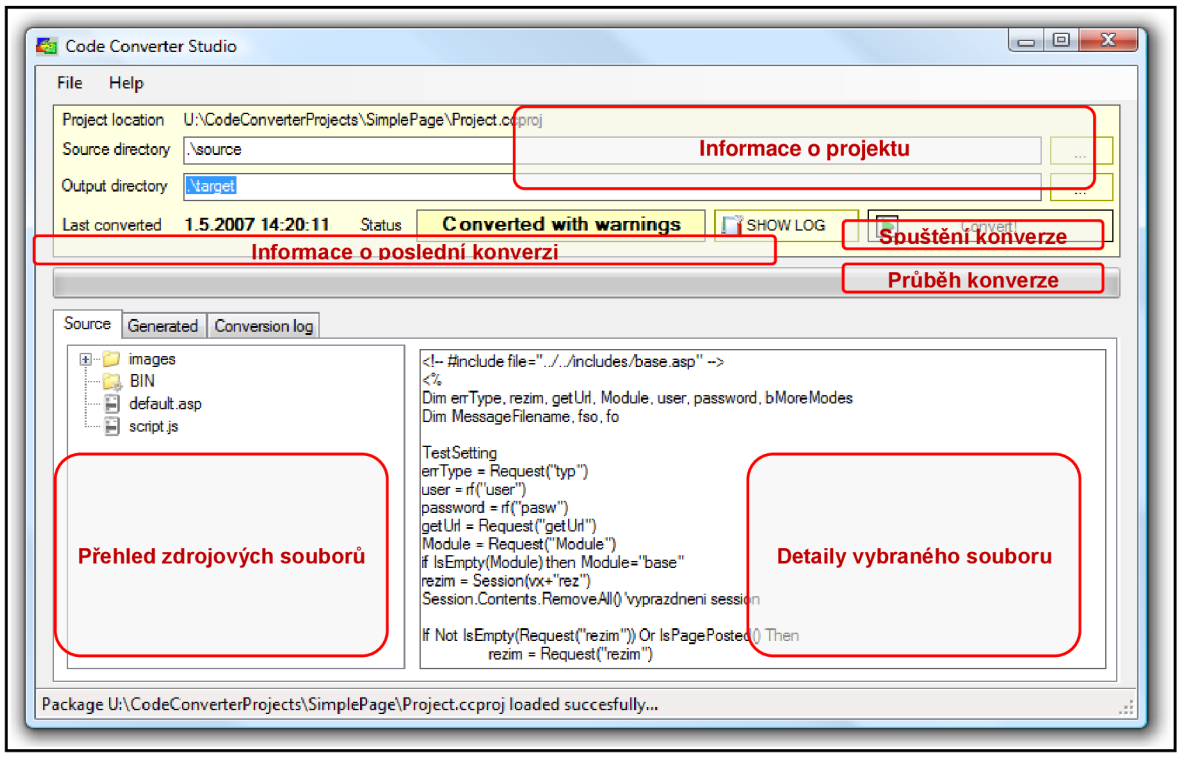
Poslední důležitá položka hlavního menu v sobě skrývá možnost nahrání konverzního balíčku (menu „Load Convert Package“). Při startu celé aplikace je automaticky nahrán konverzní balíček se jménem asp2aspx.zip. Volbou „Generate Template“ lze vygenerovat prázdnou výstupní šablonu.




**Obr 5.14 Convert Package Tester - menu Converter**

## 5.8 Code Converter Studio

Aplikace slouží na vytváření projektů a spouštění (opakované) konverze za pomoci konverzních balíčků. Po otevření/vytvoření projektu se zobrazí formulář, viz Obr 5.15 Code Converter Studio - hlavní formulář.




**Obr 5.15 Code Converter Studio - hlavní formulář**

### 5.8.1 Popis částí aplikace

„**Informace o projektu**“: zde se zobrazí přehled informací o projektu – tedy, kde se projekt nachází, jaký je cílový a jaký zdrojový adresář projektu (relativně od umístění projektu, nebo absolutní cesta).

„**Informace o poslední konverzi**“: Informace, kdy a s jakým výsledkem (včetně logu) byla konverze provedena.

„**Spuštění konverze**“: tlačítko, po jehož aktivace vyvolá spuštění konverze. Všechny editovatelné části jsou poté zneaktivněny.

„**Průběh konverze**“: zobrazení průběhu konverze formou „progress bar“ komponenty.

„**Přehled zdrojových souborů**“: zobrazení adresářové struktury a souborů ve zdrojovém adresáři pro konverzi. Po označení souborů se zobrazí detail daného souboru v panelu „detaily vybraného souboru“.

„**Detaily vybraného souboru**“: zobrazí detailní informace o vybraném souboru. U textových položek (ASP, CS, CSS…) zobrazí zdrojový kód, u obrázků a jiných binárních souborů zobrazí pouze základní informace (název, velikost,…).

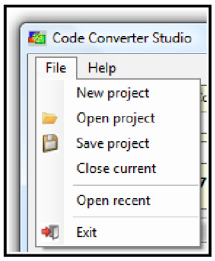
„**Záložka source**“: informace o zdrojovém adresáři – viz Přehled zdrojových souborů a Detaily vybraného souboru.

„**Záložka generated**“: zobrazí ty samé informace ale o vygenerovaných souborech.

„**Záložka conversion log**“: při konverzi zde zobrazí probíhající činnosti.

### 5.8.2 Menu aplikace

V první volbě hlavního menu lze zejména vytvořit nový projekt, otevřít nový projekt, uložit změněný projekt a uzavřít právě otevřený projekt. Menu také obsahuje přehled naposledy otevřených souborů a možnost aplikaci ukončit.

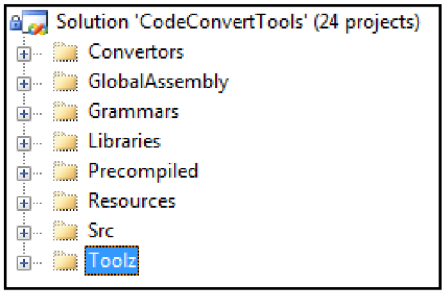



**Obr 5.16 Code Converter Studio – menu File**

Tato kapitola popisuje, jak byl návrh implementován. Popisuje struktury vytvořených projektů, jejich výstupy, závislosti a další postupy při implementaci.

## 6.1 Popis projektů

Celá implementace byla řešena v rámci vývojového prostředí Visual Studio 2005. Bylo vytvořeno „solution“, které obsahovalo další podprojekty.

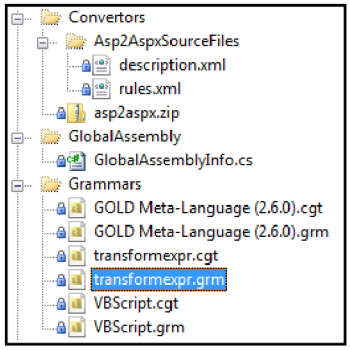



**Obr 6.1 Přehled částí Solution CodeCOnvertTools**

„**Convertors**“ – zde se nacházejí konverzní balíčky, zejména tedy asp2aspx.zip.

„**GlobalAssembly**“ – zde je informace o verzi celého řešení, copyright,…

„**Grammars**“ – obsahuje použité gramatiky (zdroj i přeloženou tabulku).



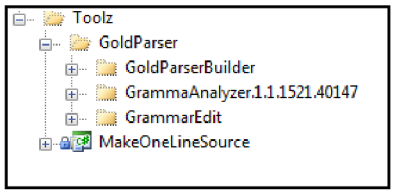

**Obr 6.2 Solution CodeCOnvertTools: část "Convertors", „GlobalAssembly“ a „Grammars“**

Libraries a Precompiled – zde se nacházejí knihovny (zdrojové soubory, resp. předkompilované DLL soubory), použité v projektu (více viz 6.3 - Knihovny použité v projektu).

Resources – obsahuje globálně použitelné ikony, obrázky,... (např. programico-source.gif a další)

Src – nejdůležitější část – zde se nachází vlastní zdrojové kódy řešení. Popis jednotlivých projektů viz následující části.

Toolz – obsahuje zejména podpůrné aplikace pro práci na projektu.




**Obr 6.3 Solution CodeCOnvertTools: část "Toolz"**

### 6.1.1 Asp2AspxTransformManager

Projekt **Asp2AspxTransformManager** obsahuje cílovou třídu použitou pro konverzi z ASP do ASPX. Tento projekt je knihovna s cílem překladu „transformmanager.dll“. Po úspěšném překladu se tato DLL připojí ke konverznímu balíčku /convertors/ asp2aspx.zip (spuštěním /AspToDotNetConverter/Src/Asp2AspxTransformManager/postbuildCommand.cmd). Toto je provedeno za pomocí projektu **ConvertPackageEditor**. Pokud tento není přeložený, nic se neprovede! Pokud se při překladu objeví chybová hláška „Error while assigning assembly to convert package…“, pak je třeba provést překlad ještě jednou (dáno zejména tím, že se překlad projektu **ConvertPackageEditor** „zpožďuje“ o jeden cyklus). Konkrétně se jedná o spuštění programu s parametrem „assign“ – také viz Classes/program.cs v tomto projektu.

### 6.1.2 GoldParser.Core

Tento projekt obsahuje zejména definici rozhraní a dalších prvků, které se používají pro sjednocení různých „strojů“ pro práci s GOLD parserem. Na všech místech v aplikaci se tedy musí pracovat s rozhraními a ne s konkrétními třídami!

Na základě tohoto projektu byly také mírně upraveny použité knihovny, zejména knihovna Calitha.

### 6.1.3 GoldParser.Wrapper

Slouží jako spoj mezi konkrétními implementacemi „strojů“ pro práci s GOLD gramatikami (zejména Calitha a Mozorov). V současné době je implementována „obsluha“ pouze stroje Calitha.

### 6.1.4 CodeConvertLib.Core

**Základ pro konverzi** mezi různými gramatikami. Obsahuje definice bázových typů, ale i konkrétní třídy pro práci s pravidly, odkazy na metody, symboly, generátory kódů,… Předpoklad je, že se později rozdělí na 2 DLL knihovny – CodeConvertLib.Core a CodeConvertLib.Engine.

### 6.1.5 AssemblyHelper

Záležitost pro sledování změn v assembly, definice verzí,… Pomocný projekt.

### 6.1.6 AspTools

V současné době „zastaralé“. Bude postupně přeneseno do **Asp2AspxTransformManager**.

### 6.1.7 ConvertPackageTester

Testovací aplikace pro konverzní balíčky. Po úspěšném překladu se výstup zkopíruje do adresáře /bin v kořenovém adresáři solution.

### 6.1.8 ConvertPackageEditor

Editor konverzních balíčků. Po úspěšném překladu se výstup zkopíruje do adresáře /bin v kořenovém adresáři solution.

### 6.1.9 Code Converter Studio

Spouštění konverze za pomoci konverzních balíčků. Po úspěšném překladu se výstup zkopíruje do adresáře /bin v kořenovém adresáři solution.

## 6.2 Konverzní balíček Asp2Aspx.Zip

Konverzní balíček byl vytvořen na základě gramatiky VBScript autora Vladimira Morozova. Konkrétní konverze dané gramatiky viz kapitola 4 - VBScript gramatika a její konverze.

S konkrétní úpravou a testování konverzních pravidel mi pomohla firma GETMORE (Ing. Martin Košut).

Cílem úprav balíčku asp2aspx bylo upravit veškerá pravidla tak, aby vracela instanci typu CodeObject, případně aby zpracovávala jiné pravidla. Toho docílíme tak, že pro pravidla, zpracovávající určitý aspekt zdrojového jazyka (např. porovnání dvou operandů), konvertujeme za použití určité metody (např. v případě výše uvedeném metodou GenBinaryOp). Tato metoda pak do generátoru výstupu zapíše instanci odpovídajícího typu (tedy např. CodeBinaryOperator).

> Cílem konverzních pravidel je vytvořit legitimní strukturu CodeDOM

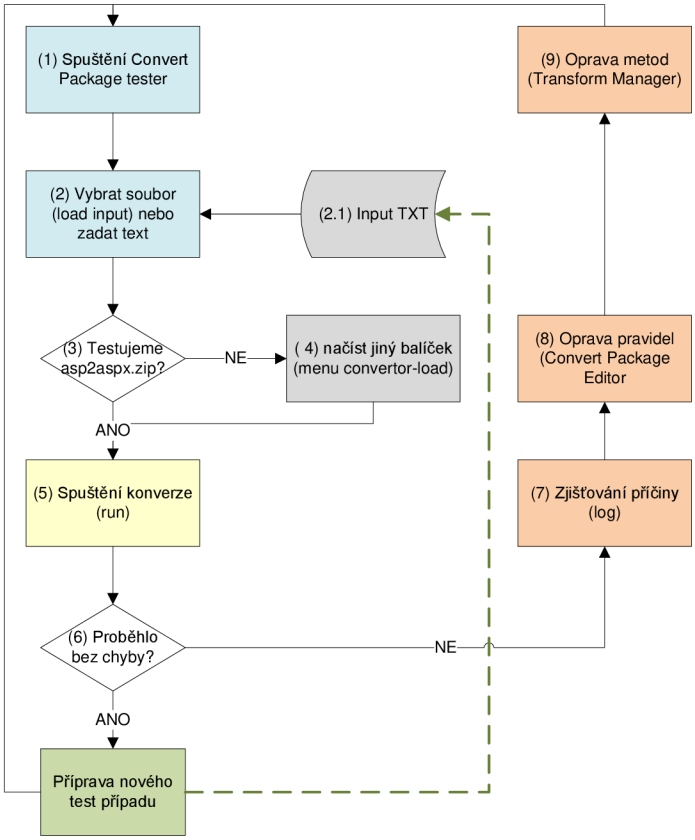
Pro dosažení tohoto cíle je třeba upravit:

- Konverzní pravidla aby volala metody správce konverze
- Vytvořit odpovídající metody ve správci konverze

Konkrétní testovací proces viz následující text.

### 6.2.1 Testování konverzního balíčku

Následující kroky byly opakovány tak dlouho, dokud nebyl konverzní balíček zcela připraven:




**Obr 6.4 Testování Asp2Aspx - vývojový diagram postupu**

Poznámka k (3): balíček asp2aspx.zip se hledá v adresářích ..\..\..\..\convertors, ..\convertors a v aktuálním adresáři.

Poznámka k (6):

- Bezchybný stav:
  - Vlevo zobrazený derivační strom
  - Vpravo zobrazený vygenerovaný zdrojový kód

- Chybový stav:
  - Program zobrazí chybovou hlášku
  - V menu Parser – log bude popsaná chyba
  - Pokud není zobrazen derivační strom a chybí generovaný kód – toto znamená, že je chyba v syntaxi vstupu. Je třeba zkusit opravit vstup. Pokud je přesto vstup zapsán v souladu se syntaxí jazyka VBScript, je možné, že je chyba v gramatice VBScript.
  - Derivační strom zobrazen je, ale chybí generovaný kód – toto je indikace chyby, která vznikla před nebo po generování kódu. Znamená to, že „něco“ shodilo proces generování. Toto je chyba aplikace a je třeba opravit Convert Package Tester.
  - Pokud je zobrazen jak derivační strom, tak vygenerovaný kód, bude vygenerovaný kód nekompletní – chyba nastala při konverzi.

### 6.2.2 Další informace

Hlavním zdrojem chyb byla zejména nedokončená transformační pravidla. Tedy například konverzní pravidlo, které chtělo vygenerovat kód podmínky, dostalo na vstup pro podmínku více výrazů. Toto nastávalo zejména v případech, kdy se konverze řešila „přímo“ vypisováním (bez použití metod).

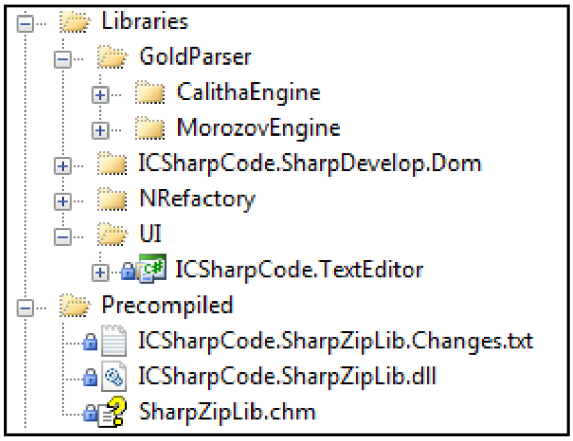
Konkrétně lze při testování postupovat následovně:

- Nastavit „startup“ projekt na ConvertPackageTester
- Upravit asp2aspxTransaformManager (zejména třídu TransformManager)
- Eventuálně upravit (v Convert Package Editoru) pravidla
- Stisknout F5 pro přeložení a spuštění s´testeru
- Spustit konverzi klávesami ALT+T a následně klávesou enter
- Nyní lze eventuálně zjistit chyby a postupovat opět od druhého bodu

## 6.3 Knihovny použité v projektu

Projekt využívá následujících knihoven:

- CalithaEngine
- MozorovEngine
- ICSharpCode.SharpDevelop.Dom
- NRefactory
- ICSharpCode.TextEditor
- ICSharpCode.SharpZipLib




**Obr 6.5 Solution CodeCOnvertTools: část "Libraries", „Precompiled“**

### 6.3.1 CalithaEngine

Knihovna sloužící jako prostředník k uloženým gramatikám GOLD Parser. Umožňuje na základě přeložených tabulek gramatik (CGT soubory) vytvářet parsery. Za pomoci této knihovny (a uložených gramatik) jsou pak zpracovávány zdrojové soubory, ale i zápis konverzních pravidel. Výstup této knihovny jsou derivační stromy.

### 6.3.2 MozorovEngine

Stejná funkcionalita jako knihovna CalithaEngine. Tato knihovna nebyla v konečné fázi použita, i když se o jejím použití uvažovalo a je relativně jednoduché v budoucnu podporu této knihovny implementovat.

Nutné kroky k implementaci podpory této knihovny:

- Implementovat rozhraní z GoldParser.Core (např. ISymbol,…) na třídy, které knihovna používá
- Upravit GoldParser.Wrapper pro podporu této knihovny (wrapper třída pro parser a úprava enum výčtu knihoven)

### 6.3.3 ICSharpCode.SharpDevelop.Dom

Tato knihovna je zde pouze kvůli knihovně NRefactory, která je na této knihovně závislá.

### 6.3.4 NRefactory

Podpora pro syntaktickou analýzu a generování kódu jazyků C# a Visual Basic.

### 6.3.5 ICSharpCode.TextEditor

Komponenta pro zobrazování zdrojového textu (včetně zobrazení čísel řádků, zvýraznění syntaxe,…). Knihovna přebrána z SharpDevelop aplikace.

### 6.3.6 ICSharpCode.SharpZipLib

Knihovna pro podporu formátu ZIP. Umožňuje číst, upravovat a ukládat soubory v ZIP formátu. Podporuje také jiné formáty (GZ, ...).

## 6.4 Statistické informace

Výše uvedené projekty obsahují celkem cca dvanáct tisíc řádků zdrojového kódu (pouze nově vytvořené projekty, bez převzatých knihoven).

# Závěr (7)

V první části diplomové práce, jsem se věnoval rešerši současného stavu. Byly analyzovány dostupné programy pro konverzi ASP stránek (NetCoole a Microsoft Migrační Asistent), které byly shledány jako nevyhovující. Protože nebylo možné použít již existující řešení, bylo nutné začít vyvíjet řešení nové. Nastudoval jsem problematiku jazyků, gramatik a překladačů. Dané znalosti pak byly využity při analýze prostředků pro konverzi ASP na ASPX. Zejména pak byla zanalyzována gramatika jazyka VBScript a připraveny pravidla pro konverzi podle gramatiky jazyka C#. Pro zápis konverzních pravidel byla vytvořena gramatika zcela nová (Transform expression grammar). Rešerší současného stavu se zabývá druhá kapitola, analýzou a popisem konverzních pravidel VBScript gramatiky pak čtvrtá kapitola.

Byly taktéž zvoleny vhodné nástroje pro vývoj konvertoru. Byl zvolen implementační nástroj (Visual Studio 2005) a další prostředky, zejména knihovny pro analýzu textů (GOLD Parser a jeho parsery). Této problematice jsem se podrobně věnoval ve třetí kapitole.

Na základě zjištěných informací bylo navrhnuto řešení. Byl navrhnut princip automatického překladu, postup konverze zdrojového adresáře, jednotlivých souborů i konkrétních bloků souboru. Detailní popis nalezneme v páté kapitole.

Navrhnuté řešení bylo implementováno. Mezi nově vyvinuté nástroje patří:

- Editor konverzních balíčků
- Aplikace pro testování konverzních balíčků
- Aplikace pro opakované spouštění konverze

Součástí implementace byly i úpravy stávajících knihoven pro práci s GOLD Parser gramatikami (CalithaEngine, Mozorov), zejména vytvoření unifikovaného rozhraní.

Navrhnuté řešení přesahuje původní zadání práce, neboť umožňuje generovat nejenom jazyk C#, ale prakticky jakýkoliv z jazyků, podporující model CodeDOM.

Při implementaci byl kladen velký důraz na další rozšiřitelnost a obecné řešení problému konverze. Rozhodně se dá říci, že za pomoci implementovaných nástrojů lze s relativně malou námahou velice snadno implementovat konverzní balíčky pro konverzi libovolného jazyka do jazyka cílového, podporující model CodeDOM. Teoreticky nejsme omezeni ani modelem CodeDOM, protože je možné „přepsat“ také třídu cílového generátoru. Celý proces konverze je tak zcela pod kontrolou programátora konverzního balíčku.

Implementovaný nástroj je v současné době používán společností GETMORE k převodu aplikace GetmoreSystem na platformu .NET. Zřejmou výhodou automatické konverze je možný současný vývoj jak nad platformou ASP, tak platformou .NET a s tím související oddělený proces testování. V okamžiku, kdy bude aplikace GetmoreSystem na platformě .NET kompletně otestována, lze během velmi krátké doby na tuto platformu přejít. Díky této automatické konverzi tak proces změny platformy téměř neovlivnil průběžný vývoj aplikace!

Během práce na tomto projektu jsem zhodnotil své dosavadní zkušenosti s jazykem VBScript, C# a prostředím .NET obecně. Objevil jsem možnosti i nevýhody CodeDOM modelu. Zvýšil jsem své znalosti z oblasti práce s .NET assembly, obecně jsem více pronikl do syntaxe obou jazyků (VBScript i C#).

Další vývoj produktu by mohl zahrnout následující aspekty:

- Rozšíření podpory GOLD Parser strojů (např. Mozorov)
- Přidání možnosti vybrat cílový jazyk pro generování
- Rozšíření podpory generování i jiných, než CodeDOM jazyků
- Rozšíření možnosti konverzní gramatiky, například umožnit vnořené volání funkcí
- Umožnit editovat konverzní metody přímo v editoru konverzních balíčků (za pomoci rekompilace DLL, eventuálně umožnění připojení zdrojového kódu)

Implementované nástroje lze určitě využít na mnoha místech, neboť i v současnosti, nedbajíc mohutnému nástupu platformy .NET, je mnoho aplikací psáno v „zastaralém“ ASP. Automatická konverze umožní hladký a bezproblémový přechod na novou platformu, což mnoho společností jistě ocení.

Řešení je v současné době využíváno interně firmou GETMORE pro konverzi vlastních projektů, o komerční nabídce nástroje se v současné chvíli neuvažuje.

Další informace o projektu budou umístěny na **http://www.codeconverttools.com/**.

## 8.1 Seznam použité literatury


[1] COOK, Devin D. *DESIGN AND DEVELOPMENT OF A GRAMMAR*. 2004th rev. edition. Sacramento: B.S., California State University, 1997. 527 s. Dostupný z WWW: <http://www.devincook.com/GOLDParser/misc/Devin-Cook-Masters-Project-GOLD.zip>.

[2] MEDUNA, Alexander, LUKÁŠ, Libor. Formal Languages and Compilers. *Přednášky předmětu IFJ: Formální jazyky a překladače* [online]. 2007 [cit. 2007-05-10]. Dostupný z WWW: <https://www.fit.vutbr.cz/study/courses/IFJ/private/>, založeno na [3].

[3] MEDUNA, Alexander. *Automata and Languages: Theory and Applications*. [s.l.]: Springer, 2000. 920 s. ISBN 978-1852330743.

[4] PARSONS, T. W. *Introduction to Compiler Construction*. New York: Freeman, 1922. 359 s. ISBN 978-0716782612.

[5] HASSAN, Ahmed E., HOLT, Richard C. *Migrating Web Applications* [online]. ASERC Workshop on Software Architecture 2003, Banff, Alberta, Canada: Feb 18-19, 2003 [cit. 2007-05-10]. Dostupný z WWW: <http://plg.uwaterloo.ca/~aeehassa/home/pubs/aserc2003.pdf>.

[6] KORZENIOWSKI, Brian J. Microsoft .NET CodeDom Technology: Part 1. *15 seconds* [online]. 2007 [cit. 2007-05-10]. Dostupný z WWW: <http://www.15seconds.com/issue/020917.htm>.

[7] KORZENIOWSKI, Brian J. Microsoft .NET CodeDom Technology: Part 2. *15 seconds* [online]. 2007 [cit. 2007-05-10]. Dostupný z WWW: <http://www.15seconds.com/issue/021113.htm>.

[8] PERRY, Nigel. *A Definition of the CodeDom Abstract Language* [online]. Version: 4 November 2002. University of Canterbury : 2002 [cit. 2007-05-10]. Dostupný z WWW: <http://kahu.zoot.net.nz/codedom/CodeDom%20Grammar.html>.

[9] SCHÖNECKER, Rudolf. *Semestrální práce do předmětu Vybrané problémy informačních systémů*. [s.l.]: [s.n.], 2006. 15 s. Dostupný z WWW: <http://www.fit.vutbr.cz/study/courses/VPD/public/0506VPD-Schonecker.pdf>.

[10] BONANSEA, Gustavo. Compiling with CodeDom. *The Code Project : Your Visual Studio and .NET resource* [online]. 2004 [cit. 2007-05-10]. Dostupný z WWW: <http://www.codeproject.com/dotnet/CompilingWithCodeDom.asp>.

[11] HARRISON, Nick. Using the CodeDOM. *O\'Reilly Network* [online]. 2003 [cit. 2007-05-10]. Dostupný z WWW: <http://www.ondotnet.com/pub/a/dotnet/2003/02/03/codedom.html>.

[12] NOVOTNY, Oren. Next Generation Tools for Object-Oriented Development. *The Architecture Journal : Input for Better Outcomes* [online]. 2005, vol. 4 [cit. 2007-05-10]. Dostupný z WWW: <http://msdn2.microsoft.com/en-us/library/aa480062.aspx>.


## 8.2 WWW odkazy

Na tomto místě bych rád uvedl několik portálů a jiných webových stránek, ve kterých jsem souhrnně hledal informace týkající se této diplomové práce. Všechny odkazy byly k datu 18. 5. 2007 funkční.


[13] **http://www.asp.net/migrationassistants/asp2aspnet.aspx** Migrační asistent od společnosti Microsoft

[14] **http://www.netcoole.com/asp2aspx.htm** Nástroj společnosti NetCoole.

[15] **http://www.antlr.org/** nástroj pro generování kódu podle gramatik.

[16] **http://rainbow.cs.unipi.gr/projects/aspa** ASP to PHP, používá ANTLR.

[17] **http://www.google.com/Top/Computers/Programming/Compilers/Lexer_and_Parser_Generators/** Přehled nástrojů pro parsování, překlad,… gramatik.

[18] **http://www.microsoft.com/msdn** Microsoft Developer Network.

[19] **http://www.devincook.com/goldparser/** GOLD Parser

## 8.3 Citace

**8**

## 9.1 Rejstřík



## 9.2 Seznam obrázků



## 9.3 Seznam tabulek

## 10.1 Problémy při použití NetCoole

Jeden příklad problémů při konverzi ASP na ASPX při použití aplikace NetCoole.

Triviální kód (default.asp)

```
<%
'/*****************************************************************************
' * dle typu IE provede presmerovani do logovaci obrazovky PC nebo DATA verze *
' *  ke strance se pridaji i vsechny GET parametry - kvuli WF mailum [JAN]    *
' *****************************************************************************/

Dim UserBrowser, UrlParams
UserBrowser = UCase(Request.ServerVariables("HTTP_UA_OS"))
if (Request.QueryString.Count>0) then UrlParams = "?"&Request.QueryString

if InStr(UserBrowser,"POCKET PC")>0 then
    response.redirect "modules/pda/default.asp"+UrlParams
else
    response.redirect "modules/base/default.asp"+UrlParams
end if
%>
```

Byl zkonvertován jako (default.aspx)

```
<%--
     Generated by ASP2ASPX 4.1.
     ASP2ASPX home page: http://www.netcoole.com
     Original File Name: [..removed..]\default.asp
--%>

<%@    Page    language="C#"    CodeFile="default.aspx.cs"    Inherits="GM.default_aspx_cs"
Debug="true"%>
<%@ Import Namespace = "Microsoft.VisualBasic" %>
<%
    ///*****************************************************************************
    //* dle typu IE provede presmerovani do logovaci obrazovky PC nebo DATA verze    *
    //*     ke strance se pridaji i vsechny GET parametry - kvuli WF mailum [JAN]    *
    //*****************************************************************************/
    UserBrowser = (Request.ServerVariables["HTTP_UA_OS"]).ToUpper();
    if ((Request.QueryString.Count > 0))
    {
        UrlParams = "?" + Convert.ToString(Request.QueryString);
    }
    if (Strings.InStr(1, UserBrowser, "POCKET PC", CompareMethod.Binary) > 0)
    {
        Response.Redirect("modules/pda/default.aspx" + UrlParams);
    }
    else
    {
        Response.Redirect("modules/base/default.aspx" + UrlParams);
    }
%>
```

Vygenerovaná stránka (default.aspx) pak při zadání do prohlížeče vykazovala chybu, popsanou na Obr 10.1 Chyba při zkonvertování stránky default.asp. Chyba byla odstraněna použitím statické metody Strings.UCase (jmenný prostor Microsoft.VisualBasic) spíše než volání metody objektu, který může být null. Ukázka opravené části (řádek 14) následuje.

```
14.     UserBrowser = Strings.UCase(Request.ServerVariables["HTTP_UA_OS"]);
```

Server Error in '/test' Application.

*Object reference not set to an instance of an object.*

**Description:** An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

**Exception Details:** System.NullReferenceException: Object reference not set to an instance of an object.

**Source Error:**

```
Line 14:     UserBrowser = (Request.ServerVariables["HTTP_UA_OS"]).ToUpper();
```

[..zkráceno..]

**Version Information:** Microsoft .NET Framework Version:2.0.50727.42; ASP.NET Version:2.0.50727.210

**Obr 10.1 Chyba při zkonvertování stránky default.asp**

## 10.2 Šablona pro generování ASP C# Code Behind

Zde je výpis kódu pro šablonu ASPX.CS stránek. Červeně jsou uvedeny komentáře.

```
1.  //------------------------------------------------------------------------------
2.  // <auto-generated>
3.  //     Tento kód byl generován nástrojem.
4.  //     Verze modulu runtime:2.0.50727.312
5.  //
6.  //     Změny tohoto souboru mohou způsobit nesprávné chování a budou ztraceny,
7.  //     dojde-li k novému generování kódu.
8.  // </auto-generated>
9.  //------------------------------------------------------------------------------
10.
11. namespace TEMP
12. {
13.       using System;
14.       using System.Data;
15.       using System.Configuration;
16.       using System.Collections;
17.       using System.Web;
18.       using System.Web.Security;
19.       using System.Web.UI;
20.       using System.Web.UI.WebControls;
21.       using System.Web.UI.WebControls.WebParts;
22.       using System.Web.UI.HtmlControls;
23.
24.       //Zde se vygenerují eventuální odkazované namespace ostatních vkládaných souborů („included" stránek)
25.
26.       public partial class TEMPClass : System.Web.UI.Page
27.       {
28.
29.                 #region public new HTTPResponse Response
30.                 /// <summary>
31.                 /// Hiddes the default Response representation. You should set
32.                 /// this property to the ad-hoc created pages...
33.                 ///</summary>
34.                 public new HttpResponse Response
35.                 {
36.                           get
37.                           {
38.                                     if ((this._Response == null))
39.                                     {
40.                                               return base.Response;
41.                                     }
42.                                     return this._Response;
43.                           }
44.                           set
45.                           {
46.                                     this._Response = value;
47.                           }
48.                 }
49.
50.                 private HttpResponse _Response;
51.                 #endregion
52.
53.                 //Zde se vygenerují deklarace „included" stránek (každá stránka jako „property").
54.
55.                 #region protected void Page_Load(...)
56.                 /// <summary>
57.                 /// Fires on page load - calls this.ExecutePage();
```

```
58.                 ///</summary>
59.                 protected void Page_Load(object sender, EventArgs e)
60.                 {
61.                         //Zde se provede volání metody Page_Load „included" stránek
62.                         this.ExecutePage();
63.                 }
64.                 #endregion
65.
66.                 #region public void ExecutePage()
67.                 /// <summary>
68.                 /// executes the page content as it was declared in the ASP file...
69.                 ///</summary>
70.                 public void ExecutePage()
71.                 {
72.                         //Zde se vygeneruje globální kód dané ASP stránky
73.                 }
74.                 #endregion
75.
76.
77.                 //Zde se vygenerují globální proměnné dané ASP stránky
78.
79.                 //Zde se vygenerují metody dané ASP stránky
80.         }
81. }
```

## 10.3 Gramatika Transformace

Příloha je výpisem gramatiky, použité při zápisu každého jednotlivého konverzního pravidla.

```
1.   !==============================
2.   ! Transform expression grammar.
3.   !
4.   ! This grammar is used for parsing transform expressions
5.   !
6.   ! Jan Vilimek   jan at vilimek.cz
7.   !
8.   ! 1.1.0.0 :
9.   !    - added NOT_SUPPORTED, DEFAULT and NOTHING keywords
10.  ! 1.0.1.0 :
11.  !    - changed terminal name "Text" for "Terminal"
12.  !    - reversed items output...
13.  !    - changed text terminal expressions from '...' to "..\".."
14.  !    - removed "line based" grammar spec.
15.  !
16.  ! 1.0.0.0 :
17.  !    - Initial version
18.  !
19.  ! USE GOLD PARSER BUILDER VERSION 2.1 AND LATER TO COMPILE THIS GRAMMAR.
20.  !==============================
21.
22.  "Name"            = 'Transform expression '
23.  "Author"          = 'Jan Vilimek'
24.  "Version"         = '1.1.0.0'
25.  "About"           = 'Transform expression grammar.'
26.  "Case Sensitive"  = True
27.  "Start Symbol"    = <TransformExpression>
28.
29.  ! ===================================================================
30.  ! Special Terminals
31.  ! ===================================================================
32.
33.  {Literal Ch}             = {Printable}    - ["]       !Basically anything, DO NOT CHANGE!
34.  {ID Tail}               = {Alphanumeric} + [_]
35.
36.  Text                    = '"'  ( {Literal Ch} | '\"' )* '"'
37.  TokenRef                = '[' {Digit}+ ']'
38.  MethodName              = {Letter} {ID Tail}*
39.
40.
41.  ! ===================================================================
42.  ! Whitespaces,... Grammar Declarations
43.  ! ===================================================================
44.
45.  {Whitespace Ch} = {Whitespace} !- {CR} - {LF}
46.
47.  Whitespace = {Whitespace Ch}+
48.
49.  !Newline    = {CR}{LF} | {CR} | {LF}
50.
51.  ! ===================================================================
52.  ! Comments
53.  ! ===================================================================
54.
55.  Comment Line  = '//'
56.  Comment Start = '/*'
57.  Comment End   = '*/'
58.
59.
60.  ! ===================================================================
61.  ! Rules
62.  ! ===================================================================
63.
64.  <TransformExpression>  ::= <GlobalKeyword>
65.                           | <Content>
66.
67.  <Content>               ::= <SymbolRef> <Content>
```

```
68.                          | <SymbolRef>
69.
70.
71.  <SymbolRef>              ::= Text                                ! the text terminal, for example "if" ...
72.                          | TokenRef                              ! reference to original reduced terminal/nonterminal
73.                          | <MethodRef>
74.
75.
76.  <MethodRef>              ::= MethodName '(' <FunctionParamsOpt> ')'       !
77.
78.  <FunctionParamsOpt>      ::= <ArgList>
79.                          |
80.
81.  <ArgList>                ::= <Param> ',' <ArgList>
82.                          | <Param>
83.
84.  <Param>                  ::= <Keyword>
85.                          | TokenRef
86.                          | Text
87.
88.  <Keyword>                ::= 'Null'
89.                          | 'True'
90.                          | 'False'
91.
92.  <GlobalKeyword>          ::= 'NOT_SUPPORTED'
93.                          | 'DEFAULT'
94.                          | 'NOTHING'
```

## 10.4 Gramatika VBScript

Příloha je výpisem zápisu gramatiky, používaného programem GOLD parser. Autorem gramatiky je Vladimír Morozov. Gramatika byla revidována Nathanem Baulchem.

```
1.   !================================
2.   ! VB Script grammar.
3.   !
4.   ! To create the grammar I was using Microsoft's VB Script documentation
5.   ! available from http://msdn.microsoft.com/scripting,
6.   ! VB Script parser from ArrowHead project http://www.tripi.com/arrowhead/,
7.   ! and Visual Basic .Net grammar file written by Devin Cook.
8.   !
9.   ! This grammar cannot cover all aspects of VBScript and may have some errors.
10.  ! Feel free to contact me if you find any flaws in the grammar.
11.  !
12.  ! Vladimir Morozov   vmoroz@hotmail.com
13.  !
14.  ! Special thanks to Nathan Baulch for the grammar updates.
15.  !
16.  ! USE GOLD PARSER BUILDER VERSION 2.1 AND LATER TO COMPILE THIS GRAMMAR.
17.  !================================
18.
19.  "Name"            = 'VB Script'
20.  "Author"          = 'John G. Kemeny and Thomas E. Kurtz'
21.  "Version"         = '5.0'
22.  "About"           = 'VB Script grammar.'
23.  "Case Sensitive"  = False
24.  "Start Symbol"    = <Program>
25.
26.  !================================
27.  ! Character sets
28.  !================================
29.
30.  {String Char}    = {All Valid} - ["]
31.  {Date Char}     = {Printable} - [#]
32.  {ID Name Char}  = {Printable} - ['['']']
33.  {Hex Digit}     = {Digit} + [abcdef]
34.  {Oct Digit}     = [01234567]
35.  {WS}            = {Whitespace} - {CR} - {LF}
36.  {ID Tail}       = {Alphanumeric} + [_]
37.
38.  !================================
39.  ! Terminals
40.  !================================
41.
42.  NewLine        = {CR} {LF}
43.                | {CR}
44.                | {LF}
45.                | ':'
46.
47.  ! Special white space definition. Whitespace is either space or tab, which
48.  ! can be followed by continuation symbol '_' followed by new line character
49.  Whitespace     = {WS}+
50.                | '_' {WS}* {CR}? {LF}?
51.
52.  ! Special comment definition
53.  Comment Line   = ''
54.                | 'Rem'
55.
56.  ! Literals
57.  StringLiteral  = '"' ( {String Char} | '""' )* '"'
58.  IntLiteral     = {Digit}+
59.  HexLiteral     = '&H' {Hex Digit}+ '&'?
60.  OctLiteral     = '&' {Oct Digit}+ '&'?
61.  FloatLiteral   = {Digit}* '.' {Digit}+ ( 'E' [+-]? {Digit}+ )?
62.                | {Digit}+ 'E' [+-]? {Digit}+
```

```
63.  DateLiteral    = '#' {Date Char}+ '#'
64.
65.  ! Identifier is either starts with letter and followed by letter,
66.  ! number or underscore, or it can be escaped sequence of any printable
67.  ! characters ([] and [_$% :-) @] are valid identifiers)
68.  ID             = {Letter} {ID Tail}*
69.                 | '[' {ID Name Char}* ']'
70.
71.  ! White space is not allowed to be before dot, but allowed to be after it.
72.  IDDot          = {Letter} {ID Tail}* '.'
73.                 | '[' {ID Name Char}* ']' '.'
74.                 | 'And.'
75.                 | 'ByRef.'
76.                 | 'ByVal.'
77.                 | 'Call.'
78.                 | 'Case.'
79.                 | 'Class.'
80.                 | 'Const.'
81.                 | 'Default.'
82.                 | 'Dim.'
83.                 | 'Do.'
84.                 | 'Each.'
85.                 | 'Else.'
86.                 | 'ElseIf.'
87.                 | 'Empty.'
88.                 | 'End.'
89.                 | 'Eqv.'
90.                 | 'Erase.'
91.                 | 'Error.'
92.                 | 'Exit.'
93.                 | 'Explicit.'
94.                 | 'False.'
95.                 | 'For.'
96.                 | 'Function.'
97.                 | 'Get.'
98.                 | 'GoTo.'
99.                 | 'If.'
100.                | 'Imp.'
101.                | 'In.'
102.                | 'Is.'
103.                | 'Let.'
104.                | 'Loop.'
105.                | 'Mod.'
106.                | 'New.'
107.                | 'Next.'
108.                | 'Not.'
109.                | 'Nothing.'
110.                | 'Null.'
111.                | 'On.'
112.                | 'Option.'
113.                | 'Or.'
114.                | 'Preserve.'
115.                | 'Private.'
116.                | 'Property.'
117.                | 'Public.'
118.                | 'Redim.'
119.                | 'Rem.'
120.                | 'Resume.'
121.                | 'Select.'
122.                | 'Set.'
123.                | 'Step.'
124.                | 'Sub.'
125.                | 'Then.'
126.                | 'To.'
127.                | 'True.'
128.                | 'Until.'
129.                | 'WEnd.'
130.                | 'While.'
131.                | 'With.'
132.                | 'Xor.'
133.
134. ! The following identifiers should only be used in With statement.
135. ! This rule must be checked by contextual analyzer.
136. DotID          = '.' {Letter} {ID Tail}*
137.                | '.' '[' {ID Name Char}* ']'
138.                | '.And'
139.                | '.ByRef'
140.                | '.ByVal'
141.                | '.Call'
142.                | '.Case'
143.                | '.Class'
144.                | '.Const'
145.                | '.Default'
146.                | '.Dim'
147.                | '.Do'
148.                | '.Each'
149.                | '.Else'
150.                | '.ElseIf'
151.                | '.Empty'
152.                | '.End'
153.                | '.Eqv'
154.                | '.Erase'
155.                | '.Error'
156.                | '.Exit'
157.                | '.Explicit'
158.                | '.False'
159.                | '.For'
160.                | '.Function'
161.                | '.Get'
162.                | '.GoTo'
163.                | '.If'
```

```
164.                | '.Imp'
165.                | '.In'
166.                | '.Is'
167.                | '.Let'
168.                | '.Loop'
169.                | '.Mod'
170.                | '.New'
171.                | '.Next'
172.                | '.Not'
173.                | '.Nothing'
174.                | '.Null'
175.                | '.On'
176.                | '.Option'
177.                | '.Or'
178.                | '.Preserve'
179.                | '.Private'
180.                | '.Property'
181.                | '.Public'
182.                | '.Redim'
183.                | '.Rem'
184.                | '.Resume'
185.                | '.Select'
186.                | '.Set'
187.                | '.Step'
188.                | '.Sub'
189.                | '.Then'
190.                | '.To'
191.                | '.True'
192.                | '.Until'
193.                | '.WEnd'
194.                | '.While'
195.                | '.With'
196.                | '.Xor'
197.
198. DotIDDot       = '.' {Letter}{ID Tail}* '.'
199.                | '.' '[' {ID Name Char}* ']' '.'
200.                | '.And.'
201.                | '.ByRef.'
202.                | '.ByVal.'
203.                | '.Call.'
204.                | '.Case.'
205.                | '.Class.'
206.                | '.Const.'
207.                | '.Default.'
208.                | '.Dim.'
209.                | '.Do.'
210.                | '.Each.'
211.                | '.Else.'
212.                | '.ElseIf.'
213.                | '.Empty.'
214.                | '.End.'
215.                | '.Eqv.'
216.                | '.Erase.'
217.                | '.Error.'
218.                | '.Exit.'
219.                | '.Explicit.'
220.                | '.False.'
221.                | '.For.'
222.                | '.Function.'
223.                | '.Get.'
224.                | '.GoTo.'
225.                | '.If.'
226.                | '.Imp.'
227.                | '.In.'
228.                | '.Is.'
229.                | '.Let.'
230.                | '.Loop.'
231.                | '.Mod.'
232.                | '.New.'
233.                | '.Next.'
234.                | '.Not.'
235.                | '.Nothing.'
236.                | '.Null.'
237.                | '.On.'
238.                | '.Option.'
239.                | '.Or.'
240.                | '.Preserve.'
241.                | '.Private.'
242.                | '.Property.'
243.                | '.Public.'
244.                | '.Redim.'
245.                | '.Rem.'
246.                | '.Resume.'
247.                | '.Select.'
248.                | '.Set.'
249.                | '.Step.'
250.                | '.Sub.'
251.                | '.Then.'
252.                | '.To.'
253.                | '.True.'
254.                | '.Until.'
255.                | '.WEnd.'
256.                | '.While.'
257.                | '.With.'
258.                | '.Xor.'
259.
260. !==============================
261. ! Rules
262. !==============================
263.
264. <NL>                  ::= NewLine <NL>
```

```
265.                          | NewLine
266.
267. <Program>              ::= <NLOpt> <GlobalStmtList>
268.
269. !================================
270. ! Rules : Declarations
271. !================================
272.
273. <ClassDecl>            ::= 'Class' <ExtendedID> <NL> <MemberDeclList> 'End' 'Class' <NL>
274.
275. <MemberDeclList>       ::= <MemberDecl> <MemberDeclList>
276.                          |
277.
278. <MemberDecl>           ::= <FieldDecl>
279.                          | <VarDecl>
280.                          | <ConstDecl>
281.                          | <SubDecl>
282.                          | <FunctionDecl>
283.                          | <PropertyDecl>
284.
285. <FieldDecl>            ::= 'Private' <FieldName> <OtherVarsOpt> <NL>
286.                          | 'Public' <FieldName> <OtherVarsOpt> <NL>
287.
288. <FieldName>            ::= <FieldID> '(' <ArrayRankList> ')'
289.                          | <FieldID>
290.
291. <FieldID>              ::= ID
292.                          | 'Default'
293.                          | 'Erase'
294.                          | 'Error'
295.                          | 'Explicit'
296.                          | 'Step'
297.
298. <VarDecl>              ::= 'Dim' <VarName> <OtherVarsOpt> <NL>
299.
300. <VarName>              ::= <ExtendedID> '(' <ArrayRankList> ')'
301.                          | <ExtendedID>
302.
303. <OtherVarsOpt>         ::= ',' <VarName> <OtherVarsOpt>
304.                          |
305.
306. <ArrayRankList>        ::= <IntLiteral> ',' <ArrayRankList>
307.                          | <IntLiteral>
308.                          |
309.
310. <ConstDecl>            ::= <AccessModifierOpt> 'Const' <ConstList> <NL>
311.
312. <ConstList>            ::= <ExtendedID> '=' <ConstExprDef> ',' <ConstList>
313.                          | <ExtendedID> '=' <ConstExprDef>
314.
315. <ConstExprDef>         ::= '(' <ConstExprDef> ')'
316.                          | '-' <ConstExprDef>
317.                          | '+' <ConstExprDef>
318.                          | <ConstExpr>
319.
320. <SubDecl>              ::= <MethodAccessOpt> 'Sub' <ExtendedID> <MethodArgList> <NL> <MethodStmtList> 'End' 'Sub' <NL>
321.                          | <MethodAccessOpt> 'Sub' <ExtendedID> <MethodArgList> <InlineStmt> 'End' 'Sub' <NL>
322.
323. <FunctionDecl>             ::= <MethodAccessOpt> 'Function' <ExtendedID> <MethodArgList> <NL> <MethodStmtList> 'End'
     'Function' <NL>
324.                          | <MethodAccessOpt> 'Function' <ExtendedID> <MethodArgList> <InlineStmt> 'End' 'Function' <NL>
325.
326. <MethodAccessOpt>      ::= 'Public' 'Default'
327.                          | <AccessModifierOpt>
328.
329. <AccessModifierOpt>    ::= 'Public'
330.                          | 'Private'
331.                          |
332.
333. <MethodArgList>        ::= '(' <ArgList> ')'
334.                          | '(' ')'
335.                          |
336.
337. <ArgList>              ::= <Arg> ',' <ArgList>
338.                          | <Arg>
339.
340. <Arg>                  ::= <ArgModifierOpt> <ExtendedID> '(' ')'
341.                          | <ArgModifierOpt> <ExtendedID>
342.
343. <ArgModifierOpt>       ::= 'ByVal'
344.                          | 'ByRef'
345.                          |
346.
347. <PropertyDecl>               ::= <MethodAccessOpt> 'Property' <PropertyAccessType> <ExtendedID> <MethodArgList> <NL>
     <MethodStmtList> 'End' 'Property' <NL>
348.
349. <PropertyAccessType>   ::= 'Get'
350.                          | 'Let'
351.                          | 'Set'
352.
353. !================================
354. ! Rules : Statements
355. !================================
356.
357. <GlobalStmt>           ::= <OptionExplicit>
358.                          | <ClassDecl>
359.                          | <FieldDecl>
360.                          | <ConstDecl>
361.                          | <SubDecl>
362.                          | <FunctionDecl>
363.                          | <BlockStmt>
```

```
364.
365. <MethodStmt>           ::= <ConstDecl>
366.                          | <BlockStmt>
367.
368. <BlockStmt>            ::= <VarDecl>
369.                          | <RedimStmt>
370.                          | <IfStmt>
371.                          | <WithStmt>
372.                          | <SelectStmt>
373.                          | <LoopStmt>
374.                          | <ForStmt>
375.                          | <InlineStmt> <NL>
376.
377. <InlineStmt>            ::= <AssignStmt>
378.                          | <CallStmt>
379.                          | <SubCallStmt>
380.                          | <ErrorStmt>
381.                          | <ExitStmt>
382.                          | 'Erase' <ExtendedID>
383.
384. <GlobalStmtList>        ::= <GlobalStmt> <GlobalStmtList>
385.                          |
386.
387. <MethodStmtList>        ::= <MethodStmt> <MethodStmtList>
388.                          |
389.
390. <BlockStmtList>         ::= <BlockStmt> <BlockStmtList>
391.                          |
392.
393. <OptionExplicit>        ::= 'Option' 'Explicit' <NL>
394.
395. <ErrorStmt>             ::= 'On' 'Error' 'Resume' 'Next'
396.                          | 'On' 'Error' 'GoTo' IntLiteral  ! must be 0
397.
398. <ExitStmt>              ::= 'Exit' 'Do'
399.                          | 'Exit' 'For'
400.                          | 'Exit' 'Function'
401.                          | 'Exit' 'Property'
402.                          | 'Exit' 'Sub'
403.
404. <AssignStmt>            ::= <LeftExpr> '=' <Expr>
405.                          | 'Set' <LeftExpr> '=' <Expr>
406.                          | 'Set' <LeftExpr> '=' 'New' <LeftExpr>
407.
408. ! Hack: VB Script allows to have construct a = b = c, which means a = (b = c)
409. ! In this grammar we do not allow it in order to prevent complications with
410. ! interpretation of a(1) = 2, which may be considered as array element assignment
411. ! or a subroutine call: a ((1) = 2).
412. ! Note: VBScript allows to have missed parameters: a ,,2,3,
413. ! VM: If somebody knows a better way to do it, please let me know
414. <SubCallStmt>           ::= <QualifiedID> <SubSafeExprOpt> <CommaExprList>
415.                          | <QualifiedID> <SubSafeExprOpt>
416.                          | <QualifiedID> '(' <Expr> ')' <CommaExprList>
417.                          | <QualifiedID> '(' <Expr> ')'
418.                          | <QualifiedID> '(' ')'
419.                          | <QualifiedID> <IndexOrParamsList> '.' <LeftExprTail> <SubSafeExprOpt> <CommaExprList>
420.                          | <QualifiedID> <IndexOrParamsListDot> <LeftExprTail> <SubSafeExprOpt> <CommaExprList>
421.                          | <QualifiedID> <IndexOrParamsList> '.' <LeftExprTail> <SubSafeExprOpt>
422.                          | <QualifiedID> <IndexOrParamsListDot> <LeftExprTail> <SubSafeExprOpt>
423.
424. ! This very simplified case - the problem is that we cannot use parenthesis in aaa(bbb).ccc (ddd)
425. <SubSafeExprOpt>        ::= <SubSafeExpr>
426.                          |
427.
428. <CallStmt>              ::= 'Call' <LeftExpr>
429.
430. <LeftExpr>              ::= <QualifiedID> <IndexOrParamsList> '.' <LeftExprTail>
431.                          | <QualifiedID> <IndexOrParamsListDot> <LeftExprTail>
432.                          | <QualifiedID> <IndexOrParamsList>
433.                          | <QualifiedID>
434.                          | <SafeKeywordID>
435.
436. <LeftExprTail>          ::= <QualifiedIDTail> <IndexOrParamsList> '.' <LeftExprTail>
437.                          | <QualifiedIDTail> <IndexOrParamsListDot> <LeftExprTail>
438.                          | <QualifiedIDTail> <IndexOrParamsList>
439.                          | <QualifiedIDTail>
440.
441. ! VB Script does not allow to have space between Identifier and dot:
442. ! a . b - Error ; a. b or a.b - OK
443. <QualifiedID>           ::= IDDot <QualifiedIDTail>
444.                          | DotIDDot <QualifiedIDTail>
445.                          | ID
446.                          | DotID
447.
448. <QualifiedIDTail>       ::= IDDot <QualifiedIDTail>
449.                          | ID
450.                          | <KeywordID>
451.
452. <KeywordID>             ::= <SafeKeywordID>
453.                          | 'And'
454.                          | 'ByRef'
455.                          | 'ByVal'
456.                          | 'Call'
457.                          | 'Case'
458.                          | 'Class'
459.                          | 'Const'
460.                          | 'Dim'
461.                          | 'Do'
462.                          | 'Each'
463.                          | 'Else'
464.                          | 'ElseIf'
```

```
                          | 'Empty'
                          | 'End'
                          | 'Eqv'
                          | 'Exit'
                          | 'False'
                          | 'For'
                          | 'Function'
                          | 'Get'
                          | 'GoTo'
                          | 'If'
                          | 'Imp'
                          | 'In'
                          | 'Is'
                          | 'Let'
                          | 'Loop'
                          | 'Mod'
                          | 'New'
                          | 'Next'
                          | 'Not'
                          | 'Nothing'
                          | 'Null'
                          | 'On'
                          | 'Option'
                          | 'Or'
                          | 'Preserve'
                          | 'Private'
                          | 'Public'
                          | 'Redim'
                          | 'Resume'
                          | 'Select'
                          | 'Set'
                          | 'Sub'
                          | 'Then'
                          | 'To'
                          | 'True'
                          | 'Until'
                          | 'WEnd'
                          | 'While'
                          | 'With'
                          | 'Xor'

<SafeKeywordID>        ::= 'Default'
                          | 'Erase'
                          | 'Error'
                          | 'Explicit'
                          | 'Property'
                          | 'Step'

<ExtendedID>           ::= <SafeKeywordID>
                          | ID

<IndexOrParamsList>    ::= <IndexOrParams> <IndexOrParamsList>
                          | <IndexOrParams>

<IndexOrParams>        ::= '(' <Expr> <CommaExprList> ')'
                          | '(' <CommaExprList> ')'
                          | '(' <Expr> ')'
                          | '(' ')'

<IndexOrParamsListDot> ::= <IndexOrParams> <IndexOrParamsListDot>
                          | <IndexOrParamsDot>

<IndexOrParamsDot>     ::= '(' <Expr> <CommaExprList> ').'
                          | '(' <CommaExprList> ').'
                          | '(' <Expr> ').'
                          | '(' ').'

<CommaExprList>        ::= ',' <Expr> <CommaExprList>
                          | ',' <CommaExprList>
                          | ',' <Expr>
                          | ','

!========= Redim Statement

<RedimStmt>            ::= 'Redim' <RedimDeclList> <NL>
                          | 'Redim' 'Preserve' <RedimDeclList> <NL>

<RedimDeclList>        ::= <RedimDecl> ',' <RedimDeclList>
                          | <RedimDecl>

<RedimDecl>            ::= <ExtendedID> '(' <ExprList> ')'

!========= If Statement

<IfStmt>               ::= 'If' <Expr> 'Then' <NL> <BlockStmtList> <ElseStmtList> 'End' 'If' <NL>
                          | 'If' <Expr> 'Then' <InlineStmt> <ElseOpt> <EndIfOpt> <NL>

<ElseStmtList>         ::= 'ElseIf' <Expr> 'Then' <NL> <BlockStmtList> <ElseStmtList>
                          | 'ElseIf' <Expr> 'Then' <InlineStmt> <NL> <ElseStmtList>
                          | 'Else' <InlineStmt> <NL>
                          | 'Else' <NL> <BlockStmtList>
                          |

<ElseOpt>              ::= 'Else' <InlineStmt>
                          |

<EndIfOpt>             ::= 'End' 'If'
                          |

!========= With Statement

```

```
566. <WithStmt>             ::= 'With' <Expr> <NL> <BlockStmtList> 'End' 'With' <NL>
567.
568. !========= Loop Statement
569.
570. <LoopStmt>              ::= 'Do' <LoopType> <Expr> <NL> <BlockStmtList> 'Loop' <NL>
571.                          | 'Do' <NL> <BlockStmtList> 'Loop' <LoopType> <Expr> <NL>
572.                          | 'Do' <NL> <BlockStmtList> 'Loop' <NL>
573.                          | 'While' <Expr> <NL> <BlockStmtList> 'WEnd' <NL>
574.
575. <LoopType>              ::= 'While'
576.                          | 'Until'
577.
578. !========= For Statement
579.
580. <ForStmt>               ::= 'For' <ExtendedID> '=' <Expr> 'To' <Expr> <StepOpt> <NL> <BlockStmtList> 'Next' <NL>
581.                          | 'For' 'Each' <ExtendedID> 'In' <Expr> <NL> <BlockStmtList> 'Next' <NL>
582.
583. <StepOpt>               ::= 'Step' <Expr>
584.                          |
585.
586. !========= Select Statement
587.
588. <SelectStmt>            ::= 'Select' 'Case' <Expr> <NL> <CaseStmtList> 'End' 'Select' <NL>
589.
590. <CaseStmtList>          ::= 'Case' <ExprList> <NLOpt> <BlockStmtList> <CaseStmtList>
591.                          | 'Case' 'Else' <NLOpt> <BlockStmtList>
592.                          |
593.
594. <NLOpt>                 ::= <NL>
595.                          |
596.
597. <ExprList>              ::= <Expr> ',' <ExprList>
598.                          | <Expr>
599.
600. !==============================
601. ! Rules : Expressions
602. !==============================
603.
604. <SubSafeExpr>           ::= <SubSafeImpExpr>
605.
606. <SubSafeImpExpr>        ::= <SubSafeImpExpr> 'Imp' <EqvExpr>
607.                          | <SubSafeEqvExpr>
608.
609. <SubSafeEqvExpr>        ::= <SubSafeEqvExpr> 'Eqv' <XorExpr>
610.                          | <SubSafeXorExpr>
611.
612. <SubSafeXorExpr>        ::= <SubSafeXorExpr> 'Xor' <OrExpr>
613.                          | <SubSafeOrExpr>
614.
615. <SubSafeOrExpr>         ::= <SubSafeOrExpr> 'Or' <AndExpr>
616.                          | <SubSafeAndExpr>
617.
618. <SubSafeAndExpr>        ::= <SubSafeAndExpr> 'And' <NotExpr>
619.                          | <SubSafeNotExpr>
620.
621. <SubSafeNotExpr>        ::= 'Not' <NotExpr>
622.                          | <SubSafeCompareExpr>
623.
624. <SubSafeCompareExpr>    ::= <SubSafeCompareExpr> 'Is' <ConcatExpr>
625.                          | <SubSafeCompareExpr> 'Is' 'Not' <ConcatExpr>
626.                          | <SubSafeCompareExpr> '>=' <ConcatExpr>
627.                          | <SubSafeCompareExpr> '=>' <ConcatExpr>
628.                          | <SubSafeCompareExpr> '<=' <ConcatExpr>
629.                          | <SubSafeCompareExpr> '=<' <ConcatExpr>
630.                          | <SubSafeCompareExpr> '>'  <ConcatExpr>
631.                          | <SubSafeCompareExpr> '<'  <ConcatExpr>
632.                          | <SubSafeCompareExpr> '<>' <ConcatExpr>
633.                          | <SubSafeCompareExpr> '='  <ConcatExpr>
634.                          | <SubSafeConcatExpr>
635.
636. <SubSafeConcatExpr>     ::= <SubSafeConcatExpr> '&' <AddExpr>
637.                          | <SubSafeAddExpr>
638.
639. <SubSafeAddExpr>        ::= <SubSafeAddExpr> '+' <ModExpr>
640.                          | <SubSafeAddExpr> '-' <ModExpr>
641.                          | <SubSafeModExpr>
642.
643. <SubSafeModExpr>        ::= <SubSafeModExpr> 'Mod' <IntDivExpr>
644.                          | <SubSafeIntDivExpr>
645.
646. <SubSafeIntDivExpr>     ::= <SubSafeIntDivExpr> '\' <MultExpr>
647.                          | <SubSafeMultExpr>
648.
649. <SubSafeMultExpr>       ::= <SubSafeMultExpr> '*' <UnaryExpr>
650.                          | <SubSafeMultExpr> '/' <UnaryExpr>
651.                          | <SubSafeUnaryExpr>
652.
653. <SubSafeUnaryExpr>      ::= '-' <UnaryExpr>
654.                          | '+' <UnaryExpr>
655.                          | <SubSafeExpExpr>
656.
657. <SubSafeExpExpr>        ::= <SubSafeValue> '^' <ExpExpr>
658.                          | <SubSafeValue>
659.
660. <SubSafeValue>          ::= <ConstExpr>
661.                          | <LeftExpr>
662. !                          | '(' <Expr> ')'
663.
664. <Expr>                  ::= <ImpExpr>
665.
666. <ImpExpr>               ::= <ImpExpr> 'Imp' <EqvExpr>
667.                          | <EqvExpr>
```

```
668.
669. <EqvExpr>              ::= <EqvExpr> 'Eqv' <XorExpr>
670.                          | <XorExpr>
671.
672. <XorExpr>              ::= <XorExpr> 'Xor' <OrExpr>
673.                          | <OrExpr>
674.
675. <OrExpr>               ::= <OrExpr> 'Or' <AndExpr>
676.                          | <AndExpr>
677.
678. <AndExpr>              ::= <AndExpr> 'And' <NotExpr>
679.                          | <NotExpr>
680.
681. <NotExpr>              ::= 'Not' <NotExpr>
682.                          | <CompareExpr>
683.
684. <CompareExpr>          ::= <CompareExpr> 'Is' <ConcatExpr>
685.                          | <CompareExpr> 'Is' 'Not' <ConcatExpr>
686.                          | <CompareExpr> '>=' <ConcatExpr>
687.                          | <CompareExpr> '=>' <ConcatExpr>
688.                          | <CompareExpr> '<=' <ConcatExpr>
689.                          | <CompareExpr> '=<' <ConcatExpr>
690.                          | <CompareExpr> '>'  <ConcatExpr>
691.                          | <CompareExpr> '<'  <ConcatExpr>
692.                          | <CompareExpr> '<>' <ConcatExpr>
693.                          | <CompareExpr> '='  <ConcatExpr>
694.                          | <ConcatExpr>
695.
696. <ConcatExpr>            ::= <ConcatExpr> '&' <AddExpr>
697.                          | <AddExpr>
698.
699. <AddExpr>               ::= <AddExpr> '+' <ModExpr>
700.                          | <AddExpr> '-' <ModExpr>
701.                          | <ModExpr>
702.
703. <ModExpr>               ::= <ModExpr> 'Mod' <IntDivExpr>
704.                          | <IntDivExpr>
705.
706. <IntDivExpr>            ::= <IntDivExpr> '\' <MultExpr>
707.                          | <MultExpr>
708.
709. <MultExpr>              ::= <MultExpr> '*' <UnaryExpr>
710.                          | <MultExpr> '/' <UnaryExpr>
711.                          | <UnaryExpr>
712.
713. <UnaryExpr>             ::= '-' <UnaryExpr>
714.                          | '+' <UnaryExpr>
715.                          | <ExpExpr>
716.
717. <ExpExpr>               ::= <Value> '^' <ExpExpr>
718.                          | <Value>
719.
720. <Value>                 ::= <ConstExpr>
721.                          | <LeftExpr>
722.                          | '(' <Expr> ')'
723.
724. <ConstExpr>             ::= <BoolLiteral>
725.                          | <IntLiteral>
726.                          | FloatLiteral
727.                          | StringLiteral
728.                          | DateLiteral
729.                          | <Nothing>
730.
731. <BoolLiteral>           ::= 'True'
732.                          | 'False'
733.
734. <IntLiteral>            ::= IntLiteral
735.                          | HexLiteral
736.                          | OctLiteral
737.
738. <Nothing>               ::= 'Nothing'
739.                          | 'Null'
740.                          | 'Empty'
```