

Universidade do Minho
Conselho de Cursos de Engenharia
Licenciatura em Engenharia Informática

Disciplina de Laboratórios de Informática IV

Ano Lectivo de 2007/08



Universidade do Minho



escola de engenharia



departamento de
informática

Construção do Grafo de Dependências de Classes C# Extensão do WebAppViewer para ASP .NET

Cristiano da Silva Castro 43114
Martinho Filipe Dias Fernandes 43170
Tiago Alves Veloso 43196
Márcio João Carvalho Coelho 43210

Supervisão:

Pedro Rangel Henriques
Daniela da Cruz

Junho, 2008

Data de Recepção	
Responsável	
Avaliação	
Observações	

Construção do Grafo de Dependências de Classes C# Extensão do WebAppViewer para ASP .NET

Cristiano da Silva Castro 43114
Martinho Filipe Dias Fernandes 43170
Tiago Alves Veloso 43196
Márcio João Carvalho Coelho 43210

Supervisão:
Pedro Rangel Henriques
Daniela da Cruz

Junho, 2008

Resumo

No projecto relatado neste documento pretendemos desenvolver uma aplicação para gerar o grafo de dependências de classes C# para facilitar a compreensão, manutenção e evolução de aplicações escritas nessa linguagem. Também é pretendido criar uma extensão para a aplicação WebAppViewer, desenvolvida por Rúben Fonseca, para que esta também reconheça a linguagem ASP .NET.

Área de Aplicação: Compreensão de Programas

Palavras-Chave: WebAppViewer, Compreensão de Programas, .NET, ASP .NET, C#, Web Application

Agradecimentos

Aos professores Pedro Rangel Henriques e Daniela da Cruz

...por nos introduzirem neste projecto e por nele nos guiarem na direcção certa.

Ao Rúben Fonseca

...por criar o WebAppViewer e por nos esclarecer dúvidas do mesmo sempre que necessário

Às nossas famílias

... por tudo

Conteúdo

Agradecimentos	i
Conteúdo	ii
Lista de Figuras	iv
1 Introdução	1
1.1 Contextualização	1
1.2 Apresentação do Caso de Estudo e Objectivos	2
1.2.1 Extensão do WebAppViewer para ASP .NET	2
1.2.2 Construção do Grafo de Dependências de Classes C#	2
1.3 Estrutura do Relatório	2
2 Enunciado do Problema	3
2.1 Extensão do WebAppViewer para ASP .NET	3
2.1.1 A nossa experiência com o WebAppViewer (WAV)	3
2.2 Construção do Grafo de Dependências de Classes C#	4
3 Arquitectura do Sistema	5
3.1 Construção do Grafo de Dependências de Classes C#	5
3.1.1 Análise do Código C#	6
3.2 Extensão do WAV	7
3.2.1 <i>Parser</i>	7
3.2.2 Análise do Código ASP .NET	7
3.2.3 Plugin	8
4 Gerador do Grafo de Dependências de Classes C#	10
4.1 Gerador de <i>Parsers</i> ANTLR	10
4.2 Análise Sintáctica	11
4.3 Travessia da Árvore	11
4.4 Serialização	12

5	Extensão do WebAppViewer para ASP .NET	13
5.1	Mono	13
5.2	<i>Parser</i>	14
5.3	Análise do Código ASP .NET	14
6	Resultados Obtidos	15
6.1	Detecção pelo WAV	15
6.2	Resultados da aplicação .NET	16
6.3	Resultados da extensão ao WAV	18
7	Conclusão	20
7.1	Síntese do Relatório	20
7.2	Análise Crítica dos Resultados	20
7.3	Trabalho Futuro	21
	Bibliografia	22
	Glossário	23

Lista de Figuras

3.1	Visão Global do Sistema	5
3.2	Parser	7
3.3	Análise do Código ASP .NET.	7
3.4	Plugin	8
4.1	Visão detalhada da aplicação .NET	10
4.2	Módulos onde o ANTLR foi usado	11
4.3	Visão mais detalhada do analisador de C#	11
5.1	Utilização do Mono no nosso sistema	13
6.1	Deteção dos ficheiros C# e ASP .NET	15
6.2	Grafo a ser gerado (C#)	16
6.3	Grafo gerado pelo WAV (C#)	17
6.4	Grafo a ser gerado (ASP .NET)	18
6.5	Grafo gerado pelo WAV (ASP .NET)	19

Capítulo 1

Introdução

Neste primeiro capítulo apresenta-se uma breve descrição do nosso projecto de Laboratórios de Informática 4. O projecto foi realizado no Departamento de Informática da Universidade do Minho em 2008, supervisionado pelos professores Pedro Rangel Henriques e Daniela Carneiro da Cruz. São mencionados neste primeiro capítulo os objectivos pretendidos com este projecto.

1.1 Contextualização

O desenvolvimento de linguagens continua a aumentar exponencialmente dia-a-dia e com a possibilidade de misturar linguagens numa aplicação web torna-se difícil perceber o que se está a usar. Devido a este problema tem-se vindo a dar maior importância a uma área recente das ciências da computação, a Compreensão de Programas (CP).

A plataforma .NET foi criada pela Microsoft e faz parte dos sistemas operativos Microsoft Windows. A plataforma .NET é executada sobre um Ambiente de Execução Comum conhecido como *Common Language Runtime* (CLR) interagindo com uma Coleção de Bibliotecas Unificadas. Ao serem compiladas as linguagens de programação na plataforma .NET, geram uma linguagem intermédia conhecida como *Common Intermediate Language* (CIL). Na implementação da linguagem intermédia pela Microsoft esta linguagem não é interpretada, mas sim compilada em *Just-in-Time* (JIT) em código máquina. A combinação destes conceitos é conhecida como a Infra-estrutura Comum de Linguagens, o *Common Language Infrastructure* (CLI). Isto significa que qualquer linguagem que se encontre implementada para o CLI pode ser usada no desenvolvimento de uma aplicação, ou seja, uma aplicação pode conter código fonte em C# e VB.NET sem que seja necessário algum cuidado na altura da compilação, pois ambas são compiladas para CLI, sendo possível misturar várias linguagens numa só aplicação.

A CP nasceu da necessidade de melhorar os processos de manutenção sistemas aplicativos, especialmente as *Web Application* (WA), onde há muitas linguagens e onde a cooperação entre estas é muito grande. Assim é necessário compreender o código escrito de uma maneira rápida e clara, por exemplo, numa equipa de programadores onde entram novos membros é importante que estes fiquem a par da aplicação que estão a desenvolver

o mais rápido possível. A **CP** está assim no cerne da engenharia de software, e é necessária para a reutilização, inspecção, manutenção, engenharia reversa, reengenharia, migração e extensão dos sistemas de software existentes.

Uma das aplicações criadas com o intuito de ajudar a manter e compreender melhor uma **WA** é o **WAV**, que de momento apenas opera em processadores de 32 bits e em sistemas operativos Linux, e apenas permite analisar aplicações em linguagem PHP.

1.2 Apresentação do Caso de Estudo e Objectivos

1.2.1 Extensão do WebAppViewer para ASP .NET

Como já foi dito o **WAV** é uma ferramenta para compreender Aplicações Web que foi desenvolvida por Rúben Fonseca no âmbito do estágio da sua licenciatura. Infelizmente, a única linguagem de programação suportada até à data pelo **WAV** é o PHP. O nosso objectivo para este projecto é desenvolver uma extensão para o **WAV** para que este reconheça a linguagem ASP .NET.

1.2.2 Construção do Grafo de Dependências de Classes C#

C# é uma linguagem orientada a objectos desenvolvida por Anders Hejlsberg, criador do Turbo Pascal e arquitecto do Delphi, para a Microsoft, como parte da plataforma .NET. Pretendemos neste projecto criar uma aplicação que, dado código fonte em C#, gera automaticamente um grafo de dependências das classes presentes no mesmo.

1.3 Estrutura do Relatório

Além do presente capítulo introdutório, este relatório está organizado da seguinte forma:

Capítulo 2 Apresenta com maior detalhe os projectos desenvolvidos;

Capítulo 3 Contém a investigação realizada e as decisões tomadas;

Capítulos 4 e 5 Descrevem a implementação de cada um dos projectos: Construção do grafo de dependências de classes C# e Extensão do WebAppViewer para ASP .NET;

Capítulo 6 Observações sobre os resultados obtidos;

Capítulo 7 Conclusão do relatório onde se tiram as conclusões sobre o trabalho efectuado e se descreve o trabalho futuro.

Capítulo 2

Enunciado do Problema

Neste capítulo iremos descrever com maior detalhe as motivações e os objectivos deste projecto.

2.1 Extensão do WebAppViewer para ASP .NET

No capítulo anterior descrevemos o [WAV](#) como sendo uma ferramenta para compreender [WA](#) que apenas analisa aplicações em PHP.

Este projecto nasce da necessidade de criar o suporte para outras linguagens, nomeadamente o ASP .NET. Como o [WAV](#) foi desenvolvido com um objectivo de extensão futura, esta tarefa torna-se assim bastante mais simples do que seria numa situação contrária. Não existe qualquer necessidade de engenharia reversa, visto que a aplicação está bem documentada e funciona segundo um modelo de motor + *plugins*, também este bem documentado. Para além disso, e conforme sugerido em [\[Fon07\]](#) podemos usar a implementação de PHP como um ponto de referência.

Os objectivos deste projecto passam por analisar o modelo de extensibilidade do WebAppViewer e desenvolver então um plugin para que o ASP .NET se torne a segunda linguagem suportada.

2.1.1 A nossa experiência com o [WAV](#)

Após uma leitura cuidada do relatório sobre a aplicação e alguma discussão sobre a mesma, finalmente iríamos conhecê-la. Tínhamos em mente que iria ser uma aplicação de fácil utilização, pois foi essa a ideia com que ficamos após a leitura do relatório, e, como previsto, assim foi. Ao iniciar o [WAV](#) fomos confrontados com uma *Graphical User Interface (GUI)* bem organizada, de fácil compreensão e intuitiva. Clica-se no botão de *import*, selecciona-se o ficheiro compactado em formato *zip* contendo a [WA](#) que pretendemos analisar, e é-nos logo apresentada uma visão global da aplicação. Gerar o grafo de dependências e visualizá-lo está á distância de um clique, assim como a maioria das funções.

O único contratempo que tivemos na utilização do [WAV](#) foi ao nível da análise comportamental. Após instalação do programa requerido para a análise comportamental

dos ficheiros PHP, continuávamos a não conseguir aceder aos ficheiros através do *browser*, apesar deste não dar nenhuma mensagem de erro ao aceder ao Uniform Resource Locator (*URL*) indicado pela aplicação (*http://localhost:8000*). Após uma ajuda do criador da aplicação, Rúben Fonseca, este explicou-nos que o servidor do WAV, o *Apache Derby*, quando não lhe apontam para um recurso específico, não tenta alternativas *standard*, como fazem os *Apache* típicos ao *redireccionar* para *index.htm*, assim sendo, teríamos de colocar o caminho completo para o ficheiro que pretendemos visualizar. Por exemplo, supondo que importámos o ficheiro *xtdlPHP.zip* no WAV e queremos aceder ao *index.php* que se encontra na raiz do ficheiro, temos que colocar no endereço do *browser* *http://localhost:8000/xtdlPHP/index.php*. Achamos que com uma pequena alteração na mensagem exibida pelo WAV este precalço pode ser ultrapassado facilmente. Em vez de exibir uma mensagem *http://localhost:8000* exibir *http://localhost:8000/<nome do ficheiro importado>/<caminho para o ficheiro a aceder>*, ou adquirir a “inteligência” dos *Apache* típicos e procurar por alternativas *standard*. Após isto a análise comportamental decorreu sem problemas.

Podemos concluir que o WAV está desenhado para ser fácil e intuitivo de usar, excepto na parte de iniciar a análise comportamental, que, como dito acima, achamos que esse “defeito” pode ser facilmente corrigido.

2.2 Construção do Grafo de Dependências de Classes C#

C# é uma linguagem orientada a objectos desenvolvida por Anders Hejlsberg, criador do Turbo Pascal e arquitecto do Delphi, para a Microsoft, como parte da plataforma .NET. Neste projecto propomo-nos a criar um Analisador de C# que, dado o código fonte de um programa em C#, desenhe, usando ferramentas .NET, o grafo de dependências das classes nele contidas, evidenciando os vários tipos de relações existentes, tais como relações de herança e de implementação. De notar que aqui o termo classe é usado num sentido lato, englobando as classes propriamente ditas, *value types*, *interfaces* e *delegates*. É usado assim com o significado de tipo. Quando for necessário distinguir entre classes no sentido estrito e os outros tipos, será usado o termo tipo.

O SIGON.2 é um sistema de informação real, suportado na web, para submissão, avaliação/aprovação, acompanhamento e auditoria de candidaturas a programas de financiamento para o desenvolvimento regional do Norte de Portugal e será usado como caso de estudo para testar o Analisador a desenvolver.

Capítulo 3

Arquitectura do Sistema

Para cumprir os objectivos iniciais do projecto e para expansão do nosso conhecimento optámos por dividir o projecto em duas fases distintas, uma que será desenvolvida em tecnologia .NET e uma outra que será desenvolvida sobre plataforma JAVA, sendo estas a construção do grafo de dependências de classes C# e a extensão do WAV para ASP .NET, respectivamente. A visão global do sistema resultante é apresentada no diagrama 3.1:

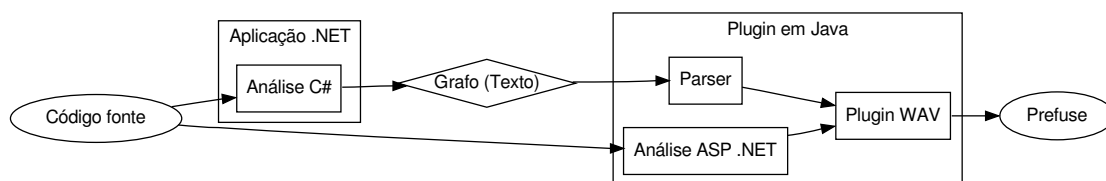


Figura 3.1: Visão Global do Sistema

A aplicação desenvolvida em .NET será independente do WAV, e criará um grafo com as dependências das classes de uma dada aplicação C#. A extensão para o WAV dependerá da aplicação .NET juntamente com um parser de ASP .NET para gerar o grafo de dependências de aplicações ASP .NET.

3.1 Construção do Grafo de Dependências de Classes C#

A construção do grafo de dependências C# será realizada por uma aplicação independente do WAV que dado código fonte em C# criará um grafo em formato de texto (syntaxe *dot*) sendo depois possível produzir o grafo em vários formatos incluindo o PDF e PS. Como foi visto na diagrama 3.1 para a aplicação .NET é necessário desenvolver um módulo que faça a devida análise do código C# e sua posterior conversão em *dot*.

3.1.1 Análise do Código C \sharp

Após ponderar sobre as relações que podem existir entre classes C \sharp obteve-se a listagem que se segue, onde X é um tipo que depende directamente de um outro tipo Y, C<Y>, T<Y> e I<Y> são, respectivamente, uma classe, um tipo ou um *interface* genéricos que têm Y como um dos seus argumentos de tipo.

Herança X deriva de Y ou de C<Y>;

Membro Existe um membro de X que é do tipo Y ou T<Y>;

Implementação X implementa Y ou I<Y>;

Atributos Uma classe depende dos atributos que a decoram, ou que decoram qualquer um dos seus membros;

Valor de Saída Existe um método em X que devolve um Y ou T<Y>;

Valor de Entrada Existe um método em X que recebe um Y ou T<Y>;

Variável Local Existe um método de X com uma variável local do tipo Y ou T<Y>

Valor Intermédio Existe uma expressão ou subexpressão num método de X que devolve um Y ou T<Y>. Por outras palavras, X usa um método de outra classe com Y ou T<Y> na assinatura;

Exceptuando esta última forma, todas estas relações, são facilmente identificáveis através de uma análise sintáctica/semântica simples. A última requer um grau de compreensão do programa mais elevado, podendo ser necessário fazer inferência de tipos. Por questões de tempo e também do âmbito e da escala deste projecto, foi decidido que esta última forma de relacionamento não será estudada. Por razões semelhantes não serão consideradas as novidades introduzidas com a versão 3.0 da linguagem.

3.2 Extensão do WAV

A extensão do WAV irá construir o grafo na estrutura utilizada pelo *prefuse* a partir do grafo em texto gerado pela aplicação descrita na secção 3.1.

3.2.1 Parser

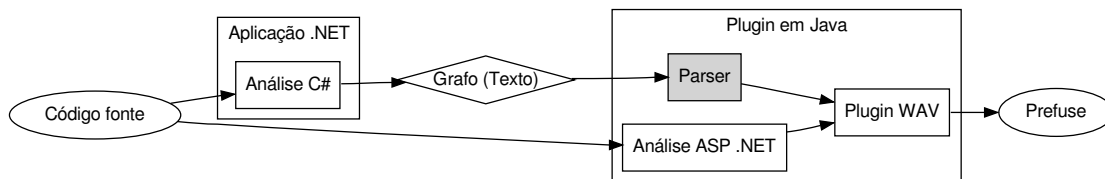


Figura 3.2: Parser

Nesta componente foi usado um subconjunto da linguagem *dot* (a sintaxe do grafo em formato de texto) para gerar um pequeno e simples *parser* para ler os nodos (classes) do grafo resultante do processamento da aplicação descrita anteriormente. Do ficheiro *.dot* apenas queremos manter informação sobre o nome dos nodos e o tipo de dependência associado a cada nodo, sendo o resto desnecessário nesta fase. A gramática após criada foi compilada pela ferramenta *ANother Tool for Language Recognition* (*ANTLR*) para criar o *parser* em linguagem *JAVA*.

3.2.2 Análise do Código ASP .NET

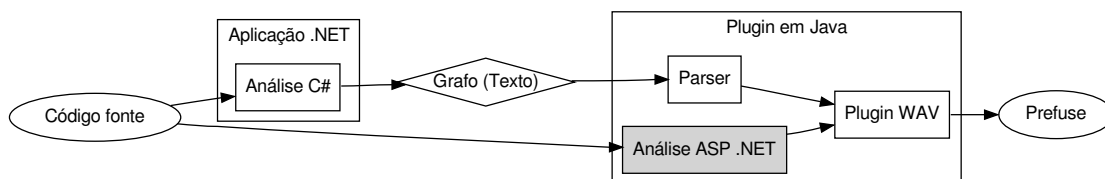


Figura 3.3: Análise do Código ASP .NET.

Após investigar melhor as relações existentes entre páginas ASP .NET encontramos as mencionadas na lista abaixo:

CodeFile Especifica o caminho referente ao ficheiro do código da página;

CodeBehind Especifica o tipo da classe para a página;

ErrorPage Define a página de redireccionamento em caso de excepções não tratadas;

MasterPageFile Contém informação sobre o caminho para a página mestre;

Src Especifica o caminho do ficheiro que contém o código que está ligado á página;

TargetSchema Especifica o nome de um XML Schema Definition ([XSD](#)) que valida o conteúdo da página.

Theme Especifica um tema para ser usado na página

StyleSheetTheme Especifica um tema com controlos especiais para ser usado na página

Achamos que todos estes componentes são de fácil implementação e portanto iremos incluí-los na totalidade no nosso plugin. Por razões de tempo e dimensões do projecto a análise comportamental da [WA](#) não irá ser implementada.

3.2.3 Plugin

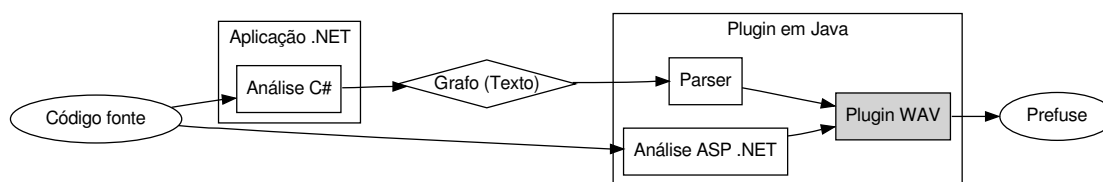


Figura 3.4: Plugin

Este módulo não é mais do que a junção dos dois anteriores de modo a que os resultados obtidos da análise dos ficheiros de código fonte sejam construídos para uma estrutura de representação de grafos usada pelo *prefuse*.

Contudo, nesta fase surgiu um problema relacionado com o módulo DETECT (ver [\[Fon07\]](#)) do plugin. Ao contrário de ASP .NET, em C# não existem “tokens mágicos” que estejam sempre presentes, o que torna difícil a detecção deste tipo de ficheiro usando este método. O detector de *Multipurpose Internet Mail Extensions* ([MIME](#))-types do WAV não é muito bom a reconhecer estes ficheiros, impondo mais uma dificuldade neste processo. A única solução suficientemente fiável a que chegámos foi recorrer ao uso do parser da linguagem para verificar a sua sintaxe. Esta solução, apesar da sua fiabilidade, é bastante pesada em termos computacionais, o que nos levou a usar uma estratégia composta por três etapas:

- 1) Verificação do [MIME](#)-type do ficheiro. Esta operação é a mais leve de todas, por isso é a primeira a ser efectuada. Apenas se esta validação falhar e o [MIME](#)-type for *text/plain*, *text/x-cpp* ou nulo é que se efectuam os passos seguintes.

- 2) Procura por tokens específicos da linguagem que não são mandatórios. Esta operação, apesar de não apresentar resultados positivos em todos os ficheiros evita algumas passagens desnecessárias pelo parser. Se esta procura não apresentar resultados passa-se à última fase.
- 3) Como último recurso, corre-se o parser da linguagem, apenas para verificar se a sua sintaxe. Esta última fase é muito pesada, daí ser a última a ser efectuada, mas é a mais fiável.

Capítulo 4

Gerador do Grafo de Dependências de Classes C#

Modulámos esta aplicação em três componentes mais relevantes:

Análise Sintáctica Processa o código fonte e gera uma *Abstract Syntax Tree* (AST).

Travessia da Árvore Percorre a AST gerada anteriormente e descobre as dependências.

Serialização Transforma o grafo de dependência num ficheiro em sintaxe *dot*.

Neste capítulo iremos descrever com maior detalhe estes três componentes. O diagrama 4.1 representa uma visão mais detalhada da aplicação .NET.

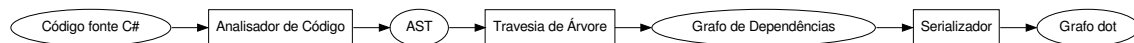


Figura 4.1: Visão detalhada da aplicação .NET

Mas antes de proceder com a descrição destas três fases iremos antes descrever uma ferramenta que foi o nosso pilar central no desenvolvimento deste projecto, o ANTLR.

4.1 Gerador de *Parsers* ANTLR

O ANTLR é um gerador de parsers que usa um *parser* do tipo LL criado por Terence Parr em Fevereiro de 1992 como sucessor do *Purdue Compiler Construction Tool Set* (PCCTS). O ANTLR usa regras do tipo *Extended Backus–Naur Form* (eBNF) ao invés da sintaxe de expressões regulares normalmente utilizada pelos outros geradores de *parsers*. A versão do ANTLR usada (2.7.7) apenas suporta geração de *parsers* em C, C# e JAVA.

Em geral gostamos desta ferramenta pois permitiu-nos criar facilmente parsers para as linguagens C#, ASP .NET e *dot*. O *parser* de C# foi gerado na própria linguagem C# para utilização na aplicação .NET. Os *parsers* de ASP .NET e de *dot* foram gerados em JAVA para inclusão no WAV, como mostrado na figura 4.2.

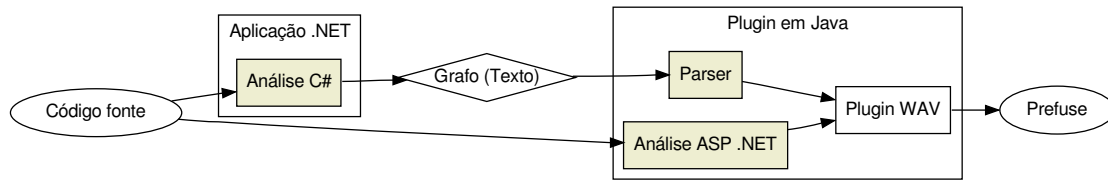


Figura 4.2: Módulos onde o ANTLR foi usado

4.2 Análise Sintáctica

Para este módulo decidimos aproveitar a gramática de C# fornecida com os exemplos do ANTLR 2.7. Com esta gramática gerou-se uma classe `CSharpParser` que efectua o *parsing* de código fonte e gera uma AST. A Árvore Sintáctica Abstracta, AST, contém em todas as suas raízes uma construção da linguagem processada, e nas folhas contém os *tokens* da linguagem. A AST é usada como passo intermédio do nosso “compilador” de C# para *dot*. A figura 4.3 representa uma visão mais detalhada do analisador de código C#.

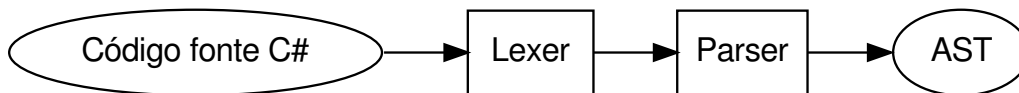


Figura 4.3: Visão mais detalhada do analisador de C#

4.3 Travessia da Árvore

Após a geração da AST é necessário percorrer essa árvore e encontrar as dependências entre os vários tipos definidos. Para isso recorreu-se mais uma vez ao ANTLR, e escrevemos um *tree walker*. Durante a travessia da árvore construímos um grafo com a seguinte estrutura para os nodos e arcos:

$$TypeReference = \begin{cases} Kind\ kind^1 & (1) \text{ O tipo de dependência} \\ string\ typeName^2 & (2) \text{ O identificador do tipo} \\ string\ namespace^3 & (3) \text{ O namespace a que pertence} \\ IList<Dependency>\ deps^4 & (4) \text{ A colecção de dependências} \end{cases}$$

$$Dependency = \begin{cases} \text{DependencyType type}^1 \\ \text{TypeReference dependsOn}^2 \end{cases}$$

(1) O tipo de dependência

(2) O “pai” da relação de dependência

4.4 Serialização

Depois de gerado o grafo de dependências apenas resta serializá-lo para um ficheiro de texto com a sintaxe *dot* para ser *parsado* posteriormente pelo plugin do [WAV](#). Não esquecer que a informação dos nodos representa o nome do tipo e os arcos a informação representa o tipo da relação de dependência.

Capítulo 5

Extensão do WebAppViewer para ASP .NET

Esta parte do projecto foi pensada em três módulos, um parser da linguagem *dot*, um analisador de código ASP .NET e o plugin que não passa da “fusão” dos dois anteriores para que o plugin funcione correctamente. Antes de avançarmos com a descrição da implementação do plugin para o WAV iremos descrever uma das tecnologias utilizadas na realização deste plugin, o Mono.

5.1 Mono

O Mono é um projecto liderado pela Novell (antigamente pela Ximian) para criar uma implementação open-source da plataforma .NET, obedecendo ao standard da *European Computer Manufacturer Association* (ECMA). Esta implementação inclui entre outros um compilador de C# e um CLR. Actualmente existem implementações de Mono para os seguintes sistemas operativos: Linux, BSD, UNIX, Mac OS X, Solaris e Windows.

O projecto Mono nasceu da necessidade de criar um conjunto de ferramentas .NET que não tivessem a mesma limitação que o Shared Source CLI (Rotor) da Microsoft que não permite o uso da mesma para fins comerciais. Um objectivo em comum com o projecto dotGNU.

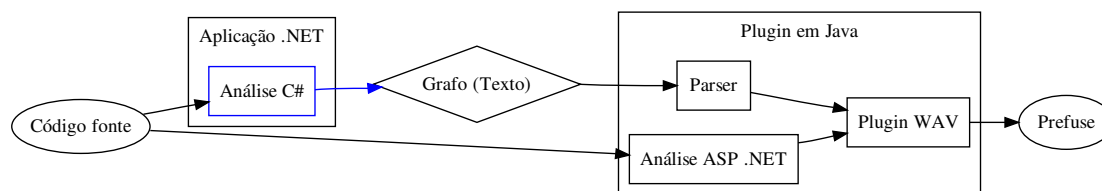


Figura 5.1: Utilização do Mono no nosso sistema

Como evidenciado na figura 5.1 o trajecto a azul representa a utilização do Mono no nosso sistema.

5.2 *Parser*

Como já foi referido a nossa aplicação .NET produz como *output* um ficheiro em sintaxe *dot* onde os nodos do grafo contêm informação sobre os tipos e os arcos a informação sobre o tipo de dependência detectada. Para a realização do *parser* escrevemos uma gramática no ANTLR para reconhecer e *parsar* a informação necessária do ficheiro *dot* para o WAV. Isto é alcançado através do Mono. Basta fazer uma chamada da nossa aplicação a partir do WAV. Assim o WAV invoca a nossa aplicação e *parsa* o ficheiro resultante através do *parser* construído para o efeito, preenchendo as suas estruturas com a informação necessária para a construção do grafo de dependências através do *prefuse*.

5.3 Análise do Código ASP .NET

A análise do código ASP .NET é conseguida mais uma vez com o auxílio da ferramenta ANTLR. Foi escrita uma pequena gramática com a intenção de capturar as dependências de um ficheiro em ASP .NET de acordo com as dependências mencionadas no capítulo 3. De seguida bastou tomar como exemplo a implementação da linguagem PHP feita por Rúben Fonseca como a primeira linguagem a ser suportada pelo WAV e criar uma implementação semelhante mas para a linguagem ASP .NET utilizando os recursos criados por nós.

Capítulo 6

Resultados Obtidos

Após todos os módulos estarem devidamente implementados como descrito nos capítulos 4 e 5 podemos observar os resultados obtidos. Neste capítulo iremos apresentá-los e discuti-los.

6.1 Detecção pelo WAV

Antes da geração dos grafos de dependências, o plugin tem de detectar correctamente os ficheiros C# e ASP .NET, tal pode ser observado na figura seguinte.

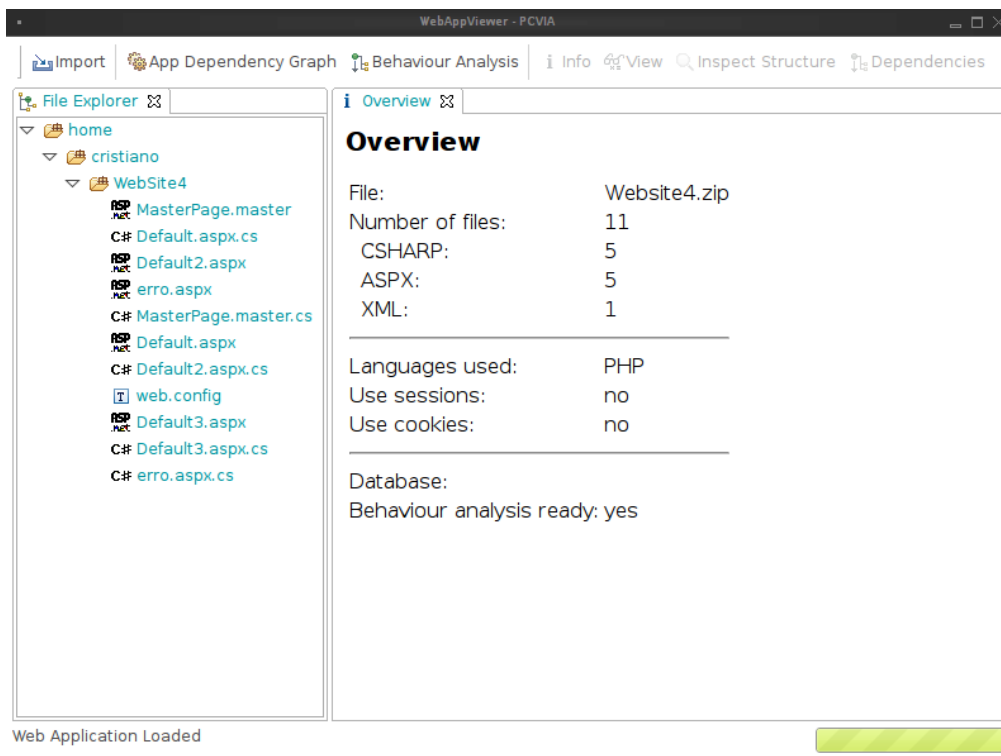


Figura 6.1: Detecção dos ficheiros C# e ASP .NET

6.2 Resultados da aplicação .NET

Segundo a diagrama de classes do nosso exemplo, a extensão que nós criamos para o WAV deveria criar um grafo de dependências idêntico ao apresentado em baixo.

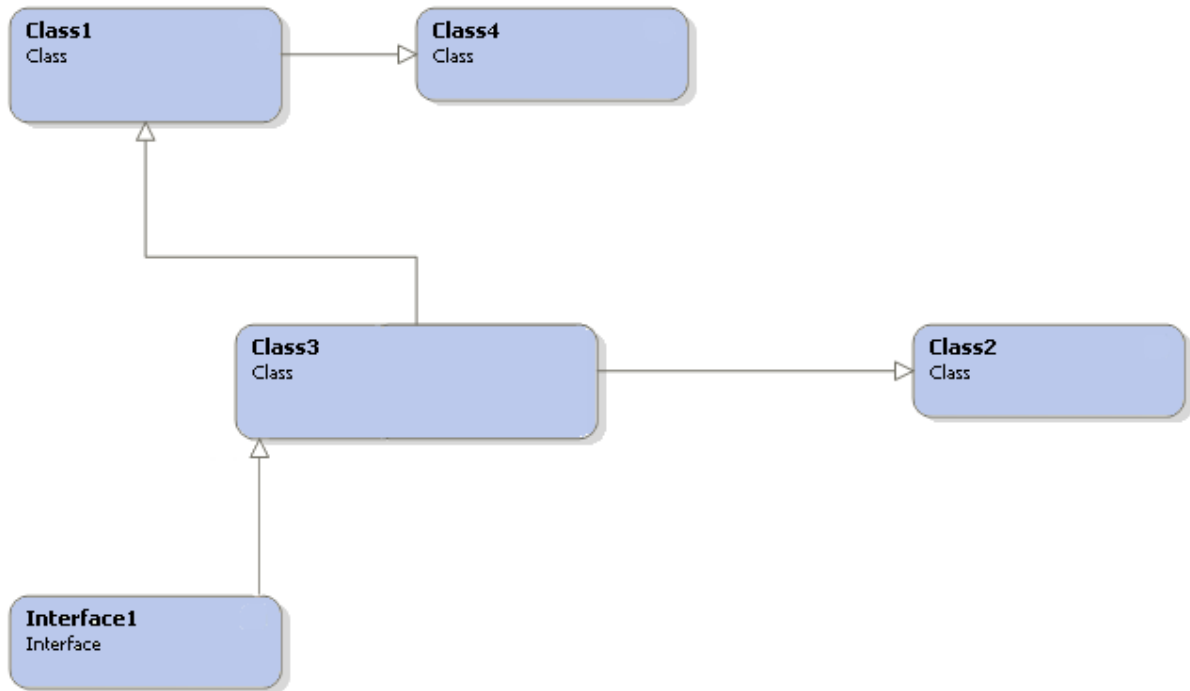


Figura 6.2: Grafo a ser gerado (C#)

Vamos de seguida apresentar o grafo gerado pela aplicação para o ficheiro em causa.

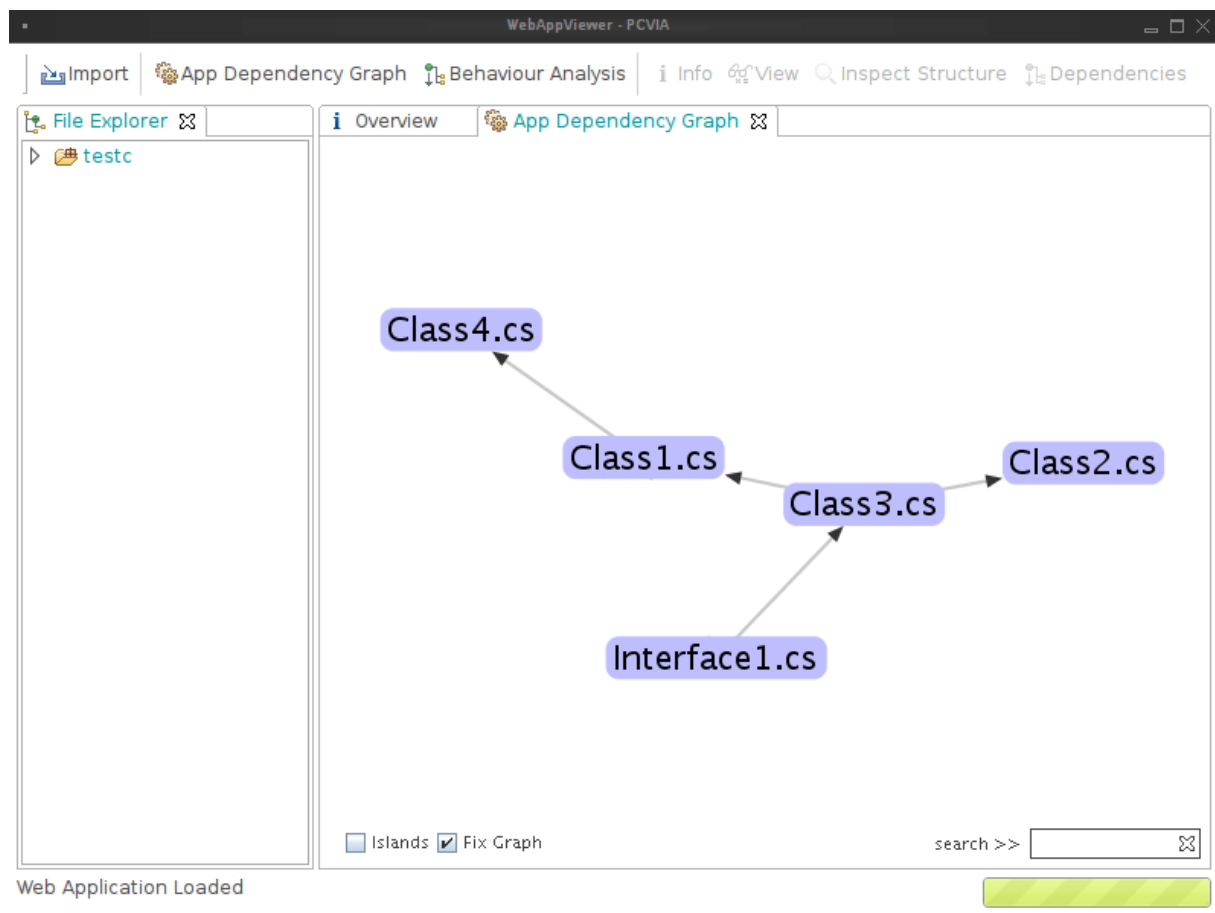


Figura 6.3: Grafo gerado pelo WAV (C#)

Podemos ver que os grafos são idênticos e assim concluir que a aplicação está a gerar as dependências correctamente.

6.3 Resultados da extensão ao WAV

Segundo o nosso ficheiro de teste, a extensão que nós criamos para o WAV deveria criar um grafo de dependências idêntico ao apresentado em baixo.

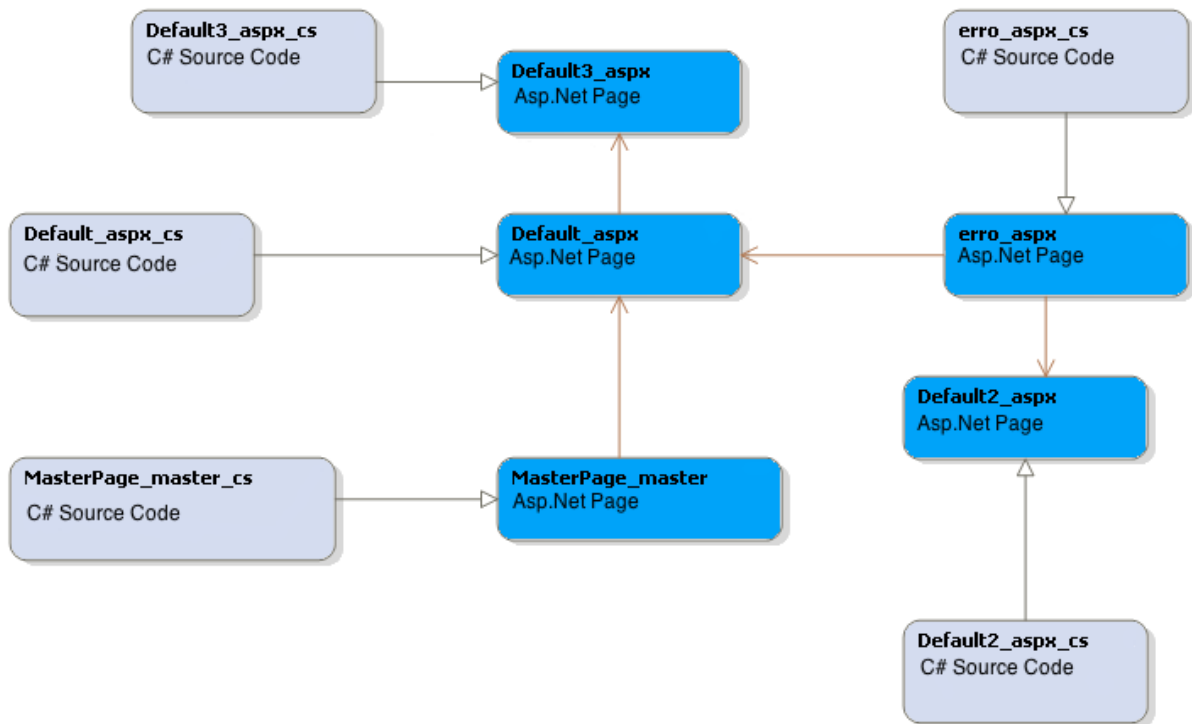


Figura 6.4: Grafo a ser gerado (ASP .NET)

Vamos de seguida apresentar o grafo gerado pela aplicação para o ficheiro em causa.

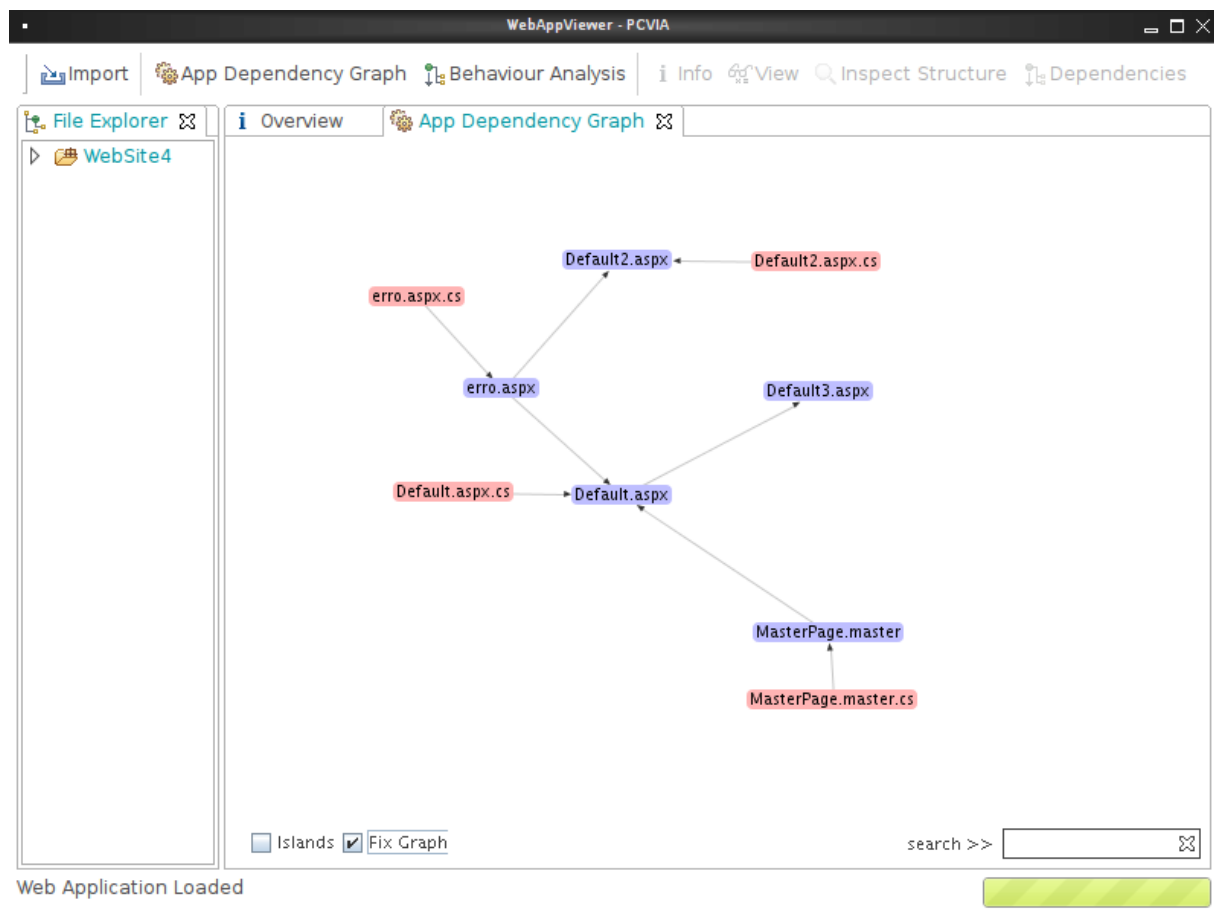


Figura 6.5: Grafo gerado pelo WAV (ASP .NET)

Podemos ver que os grafos são idênticos e assim concluir que a aplicação está a gerar as dependências correctamente.

Capítulo 7

Conclusão

7.1 Síntese do Relatório

Este documento começou por contextualizar o leitor em alguns conceitos relevantes para este projecto, a [CP](#), a plataforma [.NET](#) e o [WAV](#). sendo estes apresentados com maior detalhe no segundo capítulo.

No terceiro capítulo apresentamos a arquitectura pensada para o sistema explicando com algum detalhe todos os módulos necessários para o devido funcionamento deste.

O quarto e quinto capítulo descrevem com todo o detalhe a implementação do sistema proposto e o sétimo os resultados obtidos.

7.2 Análise Crítica dos Resultados

A aplicação [.NET](#) funciona como pretendido, parsando código [C#](#), tratando-o devidamente e produzindo como *output* um ficheiro em sintaxe *dot*, podendo este ficheiro ser utilizado para gerar uma visualização do grafo num formato suportado pela ferramenta *Graphviz*¹, como por exemplo [PDF](#) e [PS](#).

O plugin para o [WAV](#) funciona, mas com algumas limitações. Por o [WAV](#) apenas funcionar em [Linux](#) e por a plataforma [.NET](#) estar apenas completamente suportada em sistemas [Windows](#), tivemos de recorrer a uma implementação *open source* da mesma, mas esta ainda não está tão solidamente implementada como a plataforma [.NET](#) em [Windows](#), logo o nosso sistema pode sofrer de alguns eventuais decréscimos de performance.

O [WAV](#) têm um erro no *overview* de um aplicação que faz com que seja sempre apresentada a linguagem [PHP](#) como a única linguagem usada, independentemente das linguagens usadas na aplicação. No entanto a distribuição dos ficheiros pelas linguagens mostra os resultados correctos, tal como pode ser visto na figura [6.1](#). Este erro foi reportado ao autor do projecto para ser resolvido futuramente.

¹O *Graphviz* encontra-se disponível para os três principais sistemas operativos, [Microsoft Windows](#), [Linux](#) e [Mac OS X](#)

7.3 Trabalho Futuro

Para trabalho futuro pode ficar a correcção ou a melhor implementação dos problemas mencionados no plugin do [WAV](#), podendo a nossa aplicação .NET ser implementada totalmente em [JAVA](#) e embutida directamente no [WAV](#). Para trabalho futuro pode também ser implementada a análise comportamental para o [WAV](#) das linguagens [ASP .NET](#) e [C#](#). Para a aplicação .NET pode ser criada uma [GUI](#) e estudadas maneiras de implementar novas linguagens.

Bibliografia

- [Fon07] Rúben Fonseca.
Implementação do WebAppViewer: uma ferramenta para compreender aplicações Web.
Technical report, Departamento de Informática, Universidade do Minho, 2007.
Relatório de Estágio.

Glossário

ANTLR *ANother Tool for Language Recognition* - Uma ferramenta para geração de *parsers*

AST *Abstract Syntax Tree* - Árvore Sintática Abstracta

CIL *Common Intermediate Language* - Linguagem Intermédia Comum

CLI *Common Language Infrastructure* - Infraestrutura de Linguagens Comum

CLR *Common Language Runtime*

CP Compreensão de Programas

DI Departamento de Informática

eBNF *Extended Backus–Naur Form* - Forma Backus-Naur Extendida

ECMA *European Computer Manufacturer Association* - Associação Europeia de Fabricantes de Computadores

gEPL grupo de Especificação e Processamento de Linguagens

GUI *Graphical User Interface* - Interface Gráfica com o Utilizador

JIT *Just-in-Time* Pode se referir a qualquer acção que seja efectuada enquanto o código está a correr, como por exemplo compilação e interpretação

MIME *Multipurpose Internet Mail Extensions*

PCCTS *Purdue Compiler Construction Tool Set*

URL *Uniform Resource Locator* - Localizador Uniforme de Recursos

WA *Web Application*

WAV WebAppViewer

XSD XML Schema Definition