UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CUROS DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

GERAÇÃO DE CÓDIGO PHP A PARTIR DA FERRAMENTA CASE RATIONAL ROSE

SIEGFRIED KREUTZFELD NETO

BLUMENAU
2003

2003/2-38

SIEGFRIED KREUTZFELD NETO

GERAÇÃO DE CÓDIGO PHP A PARTIR DA FERRAMENTA CASE RATIONAL ROSE

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Everaldo Artur Grahl

BLUMENAU
2003

2003/2-38

GERAÇÃO DE CÓDIGO PHP A PARTIR DA FERRAMENTA CASE RATIONAL ROSE

Por

SIEGFRIED KREUTZFELD NETO

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Everaldo Artur Grahl, Orientador, FURB
Membro:	Prof. Marcel Hugo, FURB
Membro:	Prof. Fábio RafaelSegundo , FURB

Blumenau, 12 de novembro de 2003

Dedico este trabalho a todos os amigos, especialmente aqueles que me ajudaram diretamente na realização deste.

Os bons livros fazem “sacar” para fora o que a pessoa tem de melhor dentro dela.

Lina Sotis Francesco Moratti

AGRADECIMENTOS

À Deus, pelo seu imenso amor e graça.

Aos meus pais Evanir e Siegfried, que sempre me apoiaram, não somente no desenvolvimento deste trabalho, mas durante toda minha vivência como acadêmico.

Aos meus amigos, pelos empurrões e cobranças.

Ao meu orientador, Everaldo Artur Grahl, pelo incentivo e apoio.

À minha namorada Joyce, por me compreender e incentivar durante todo o período de desenvolvimento deste trabalho.

RESUMO

Este trabalho consiste no estudo da WAE (*Web Application Extension*), extensão da UML para desenvolver aplicações *web*; e da ferramenta CASE Rational Rose. Como resultado foi criado um protótipo de um *add-in* que tem como objetivo gerar código PHP a partir de uma modelagem WAE na ferramenta CASE Rational Rose. O código é gerado em nível estrutural e permite que a aplicação seja desenvolvida com base nos requisitos, assegurando uma maior consistência.

Palavras chaves: UML, WAE, Rational Rose, geração de código, PHP.

ABSTRACT

This work consists in the study about the WAE (Web Application Extension), an extension of UML created to develop web applications; and the CASE tool Rational Rose. As result it was created a prototype of an add-in that has the purpose of generate PHP code from a WAE model at Rational Rose. The code is generated in a design level and allow the application to be developed based in the requirements, assuring a better consistence.

Key-Words: UML, WAE, Rational Rose, Code Generation, PHP

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

HTML – *Hyper Text Markup Language*

OLE – *Object Linking and Embending*

PHP – PHP *Hypertext Preprocessor*

UML – *Unified Modeling Language*

WAE – *Web Application Extension*

SUMÁRIO

1 INTRODUÇÃO

Faz pouco tempo que os desenvolvedores vêm reconhecendo as vantagens e facilidades que um aplicativo *web* pode oferecer. Da mesma forma, menor ainda foi o tempo que se gastou para a questão do processo de desenvolvimento desses aplicativos. No entanto, qualquer sistema que apresenta uma complexidade um pouco mais elevada, necessita ser planejado e modelado. Baseado nesse contexto, Conallen (1999a) desenvolveu uma extensão para a *Unified Modeling Language* (UML), com o objetivo de modelar aplicações *web*.

Esta extensão da UML, desenvolvida e estudada por Conallen (1999a), está presente na ferramenta CASE Rational Rose Enterprise 2002. Assim, usando o Rational Rose, é possível modelar aplicações *web* utilizando os estereótipos criados por ele, como por exemplo Página Servidor, Página Cliente e Formulário HTML. Estereótipo é um mecanismo de extensão da UML, normalmente usado para indicar alguma especialização significativa nas classes do modelo. Umas das vantagens do Rational Rose é sua capacidade de gerar código a partir destas modelagens.

Geração de código é uma abordagem que reduz o tempo de desenvolvimento e permite que os engenheiros realizem um trabalho melhor, mais criativo e útil, reduzindo código redundante feito a mão (HERRINGTON, 2003). Utilizando-se técnicas de geração de código, o processo de desenvolvimento de um software torna-se mais produtivo, e permite uma manutenção mais fácil. Outra vantagem é que o software será construído com base na descrição dos requisitos, assegurando uma consistência maior. Entre as linguagens que se destinam a *web* suportadas na versão Rational Rose Enterprise 2002 estão *Active Server Pages* (ASP) e *Java Server Pages* (JSP).

ASP e JSP, juntamente com PHP *Hypertext Preprocessor* (PHP), são as linguagens mais comuns para desenvolvimento de aplicativos para *web*. O Rational Rose ainda não incorporou a capacidade de gerar código na linguagem PHP. Esta ferramenta possui características que tornam possível o desenvolvimento de extensões, como por exemplo na forma de *scripts* ou *add-ins*, que permitem customizar a ferramenta. Para o desenvolvimento de *scripts*, o Rational Rose incorpora um editor próprio, através do qual o usuário pode inserir seu código. Já um *add-in* normalmente é programado e compilado em outra linguagem, para depois ser acrescentado à ferramenta. Através destas extensões, é possível personalizar

menus, automatizar funções, integrar o Rational Rose com outras ferramentas e até mesmo adicionar novas funções.

O PHP é uma linguagem em forma de *script* (parte do código *Hyper Text Markup Language* – HTML, é interpretado pelo *browser* ou pelo servidor *web*) que interage no lado do servidor para a criação de páginas dinâmicas (ANSELMO, 2000). É uma linguagem de código fonte aberto, o que permite uma constante evolução e aperfeiçoamento, bem como qualquer customização da linguagem. Sua notação é simples, porém é uma linguagem de funcionamento otimizado, evitando que a execução de seus *scripts* interfiram na performance do *web*site.

Durante o desenvolvimento deste trabalho, pode-se identificar a ferramenta Umbrello UML Modeller, que possui a capacidade de gerar código para as linguagens C++, Java e PHP, mas não utiliza a notação WAE. Um dos motivos da dificuldade em encontrar uma ferramenta com essa funcionalidade está no fato de que a utilização da UML para *web* é ainda uma área em desenvolvimento.

1.1 OBJETIVOS DO TRABALHO

O objetivo principal do trabalho é desenvolver uma extensão para a ferramenta CASE Rational Rose, que permita através da modelagem UML de um sistema *web*, gerar código em nívelestrutural na linguagem PHP.

Os objetivos secundários são:

a) pesquisar a extensão *Web Application Extension* (WAE) da UML, que permite a modelagem de sistemas *web*;
b) pesquisar a ferramenta Rational Rose, verificando a melhor forma de estender suas funcionalidades.

1.2 ESTRUTURA DO TRABALHO

O segundo capítulo apresenta a linguagem PHP, conceitos sobre aplicações *web* e sua classificação.

No terceiro capítulo, a UML é apresentada, tratando de forma mais detalhada sobre a *Web Application Extension* (WAE). A ferramenta CASE Rational Rose e as formas que ela possui para estender suas capacidades são apresentadas neste capítulo também.

No quarto capítulo é apresentado o protótipo de uma extensão para a ferramenta CASE Rational Rose, que gere código PHP a partir de uma modelagem UML.

O quinto capítulo apresenta as conclusões e resultados alcançados, bem como sugestões para continuação deste trabalho.

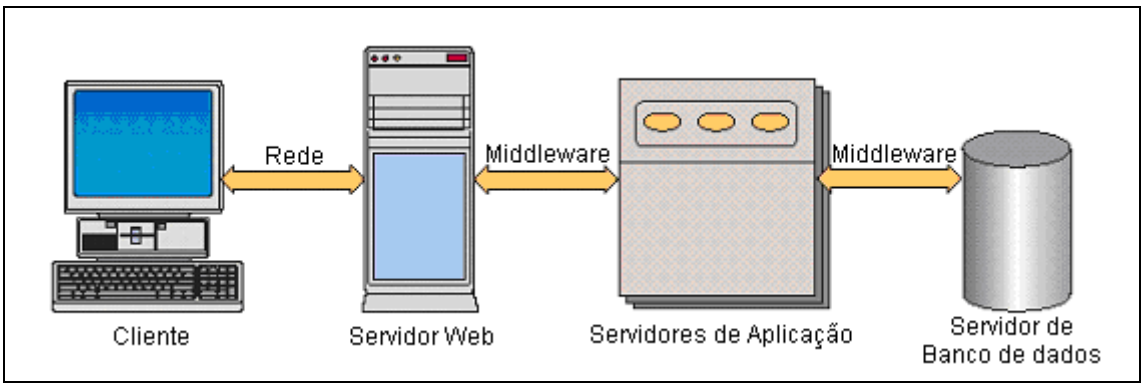
2 PHP

Este capítulo apresenta os conceitos de aplicações *web*, e apresenta a linguagem PHP e suas características principais.

2.1 APLICAÇÕES *WEB*

Segundo Offut (2002), aplicações *web* consistem na interação de diversos componentes, incluindo softwares tradicionais e não-tradicionais, linguagens de *script* interpretada, arquivos HTML, HTML misto com outros programas, banco de dados, imagens e interfaces complexas. "Uma aplicação *web* é um *web*site onde o usuário interfere no estado do sistema" (CONNALEN, 1999c).

O atual modelo da estrutura de um aplicativo *web*, como o exemplo mostrado na Figura 1, permite que este caminhe por várias camadas. Entre essas camadas estão distribuídos o aplicativo do cliente, o banco de dados e o servidor *web*. A antiga estrutura de um *web*site era composta apenas por duas camadas. Essas mudanças ocorreram para aumentar a qualidade de variáveis como qualidade, segurança e confiança dos aplicativos *web*. A nova geração de *web*sites permite também uma maior expansão dos aplicativos.



Fonte: OFFUT, 2002

FIGURA 1 - Geração de *web*sites modernos: modelo de n camadas

2.1.1 Aplicações *web* magro e *web* gordo

Aplicações *web* magro são aquelas onde todos os componentes são executados do lado do servidor. O padrão arquitetônico do cliente *web* magro é útil para aplicações baseadas na

Internet, para que apenas a configuração mínima do cliente possa ser assegurada (CONALLEN, 2003).

Esse tipo de aplicação exige do cliente menor quantidade de recursos. No caso de aplicações *web*, isso significa a capacidade de executar um navegador *web* HTML.

O padrão arquitetônico do cliente *web* gordo estende o padrão do cliente *web* magro com *scripts* e objetos personalizados do lado do cliente, como controles Active X e *applets* Java (CONALLEN, 2003). Esse tipo de aplicação permite que parte da lógica do problema seja executada do lado do cliente. A maior diferença entre estas duas arquiteturas esta no papel do navegador.

2.2 A LINGUAGEM

PHP, acrônimo de PHP *Hypertext Preprocessor* (Pré-processador de Hipertexto), é uma linguagem de elaboração embutida que opera do lado do servidor (CASCAGNETTO, 2001). Isso significa que ela é introduzida dentro de um documento HTML, entre as *tags* deste, conferindo-lhe a capacidade de gerar instruções específicas. Através destas funções do PHP é possível transformar pequenos arquivos HTML estáticos em páginas com conteúdo dinâmico que possam interagir com banco de dados, ou com o usuário.

O exemplo apresentado no Quadro 1 é o clássico programa Hello World na linguagem PHP. As linhas em negrito representam o código PHP embutido dentro do HTML.

```
<HTML>
<HEAD>
<TITLE> Hello World em PHP </TITLE>
</HEAD>
<BODY>
<?
// Exemplo do programa "Hello world"  em PHP
 print("Hello World");
?>
</BODY>
</HTML>
```

QUADRO 1 – Em negrito, exemplo de código PHP

Várias são as vantagens que fazem do PHP uma boa opção para o desenvolvimento de aplicativos *web*. Primeiramente, a simplicidade pela qual o PHP trata funções como por exemplo o acesso a um banco de dados; que pode ser realizado em poucas linhas. A forma

como os *scripts* são processados é bem-otimizada, permitindo um tempo de resposta adequado ao necessário em aplicativos *web*.

O PHP é portável, possui funções específicas para tecnologias como *Extensible Markup Language* (XML) e documentos Adobe *Portable Document Format* (PDF) (SOARES, 2000). Apesar de possuir uma notação simples, pode ser usada em aplicações de grande complexidade.

O PHP é uma linguagem de código-fonte aberto, o que torna possível uma evolução e aperfeiçoamento muito mais rápidos. É comum a criação de bibliotecas para funções específicas, como por exemplo para acesso a um determinado banco de dados ainda não suportado. Atualmente o PHP encontra-se na versão 4.3, chamada de PHP4.

2.2.1 Sintaxe

Para o desenvolvimento deste trabalho, levou-se em consideração a sintaxe de alguns dos elementos da linguagem PHP. As variáveis no PHP são representadas por um cifrão ($) seguido pelo nome da variável. Os nomes de variável no PHP fazem distinção entre maiúsculas e minúsculas.

As funções são declaradas conforme a sintaxe mostrada no Quadro 2.

```
<?php
function Funcao($argumento_1, $argumento_2, ..., $argumento_n)
{
    return $valor;
}
?>
```

QUADRO 2 – Sintaxe para declaração de funções em PHP

A declaração de classes na linguagem PHP está representada no Quadro 3.

```
<?
class Classe
{
    var $Variavel;

    function adiciona ($arg1, $num)
    {
        $this->Variavel[$arg1] += $num;
    }

    function remove ($arg1, $num)
    {
        if ($this-> Variavel [$arg1] > $num) {
            $this-> Variavel [$arg1] -= $num;
            return true;
        } else {
            return false;
        }
    }
}
?>
```

QUADRO 3 – Sintaxe para declaração de classes na linguagem PHP

3 UNIFIED MODELING LANGUAGE (UML)

"UML é uma linguagem destinada a visualizar, especificar, construir e documentar os artefatos de um sistema complexo de software" (BOOCH, 2000). É um padrão que auxilia o desenvolvimento de softwares baseado em objeto. Através da UML, é possível capturar a estrutura de sistemas orientados a objeto e expressá-las através de diagramas (PAGE-JONES, 2001). Dentre as características mais relevantes da UML, pode-se citar a existência de mecanismos de extensibilidade e especialização, que permitem ampliar os conceitos centrais da linguagem.

3.1 UML PARA *WEB*

A popularização das aplicações *web* faz crescer a necessidade de estudos sobre a metodologia de desenvolvimento das mesmas. Pensando nisso, Conallen (1999a) desenvolveu uma extensão para a linguagem UML, que permite a modelagem de aplicações *web* como qualquer outro software.

Excluindo-se detalhes que não afetam diretamente a lógica do sistema, como por exemplo, detalhes visuais do *web*site, Conallen (1999a) pode identificar os principais componentes que compõem um aplicativo *web*. A partir do desenvolvimento deste trabalho, Conallen definiu a *Web Application Extension* (WAE).

3.2 *WEB APPLICATION EXTENSION* (WAE)

É uma extensão que permite representar páginas *web* e outros elementos significativos do ponto de vista arquitetônico no modelo, paralelamente às classes "normais" do modelo (CONALLEN, 2003).

Uma extensão para UML é definida através de estereótipos, com valores com *tags* e restrições. Através da combinação desses elementos, é possível representar outros modelos de negócios na notação UML.

Estereótipo é uma extensão ao vocabulário da linguagem que permite anexar um novo significado de semântica a um elemento do modelo. Um valor com *tag* é uma propriedade que será associada a um elemento do modelo. Restrições são extensões semânticas da linguagem

que impõem condições sob as quais o modelo pode ser considerado bem formado (CONALLEN, 2003).

Uma extensão para UML contém uma breve descrição, a lista e características de todos os estereótipos que a compõe. O Anexo A contém a especificação dos itens mais comuns da WAE. A visão lógica do modelo é composta em sua maioria pelas classes e seus relacionamentos.

Entre os estereótipos que compõe a WAE estão os de associação: *link*, *build*, *submit*, *redirect*, *forward*, *object* e *include*. Os principais estereótipos de classes são Página do Servidor, Página do Cliente e Formulário HTML. Estes são os estereótipos básicos para se montar a estrutura de uma aplicação *web*.

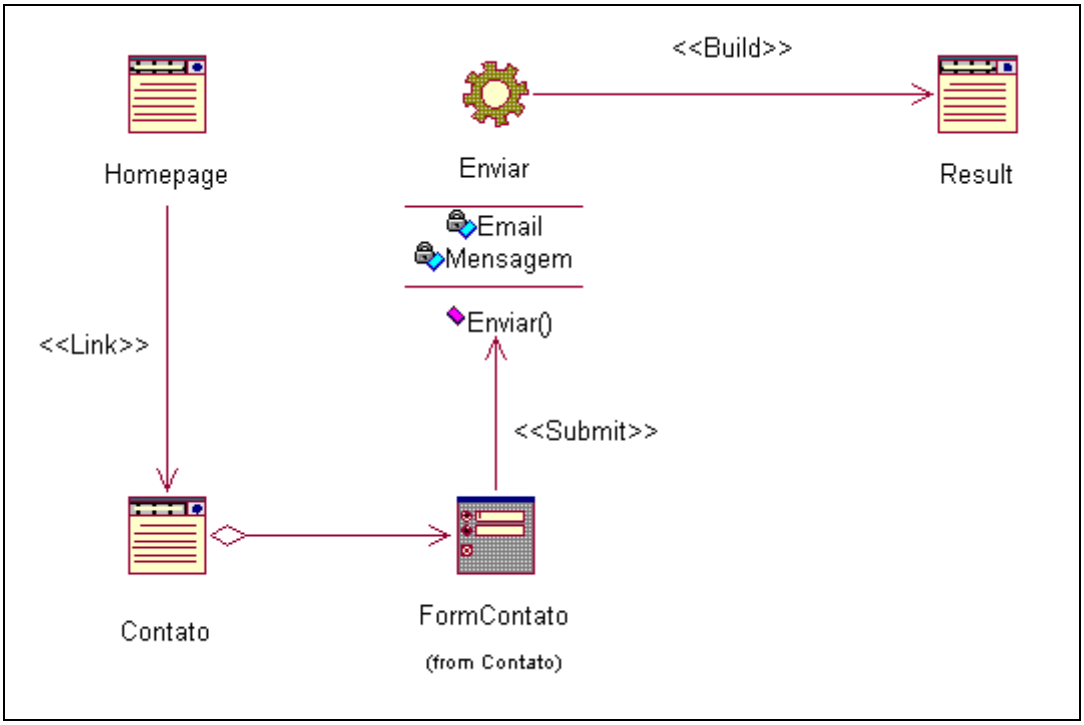


FIGURA 2 – Exemplo de modelagem utilizando a WAE

O exemplo apresentado na Figura 2 é uma modelagem simples utilizando os principais elementos da WAE. Nele encontra-se três Páginas Clientes: a *Homepage*, a Contato e a *Result*. Entre as classes Contato e *Homepage* existe uma associação *Link*, indicando a presença de uma âncora de uma página para a outra.. A classe Contato possui um Formulário HTML, o *form* Contato, que por sua vez, é tratado por uma Página Servidor, a Enviar. Este elemento possui dois atributos principais, a variável Email e a variável Mensagem. Possui

também um único método, Enviar. A Página Servidor apresenta um relacionamento *Build* com a classe *Result* Esse r elacionamento identifica a saída HTML de uma execução da página servidor.

3.3 A FERRAMENTA RATIONAL ROSE

O mercado oferece uma vasta quantidade de ferramentas CASE com a capacidade de modelar sistemas utilizando UML, mas poucas apresentam a capacidade de modelar aplicativos *web*. Dentre as ferramentas CASE capazes de modelar aplicativos *web* está o Rational Rose.

> O Rational Rose é uma ferramenta de desenho baseada no conceito de modelo de negócios. Ou seja, trata-se de uma ferramenta robusta, com o propósito de desenvolver soluções que atendam às necessidades do negócio do cliente que vão desde uma simples solução localizada até soluções complexas baseadas em ambientes distribuídos (MATOS, 2002).

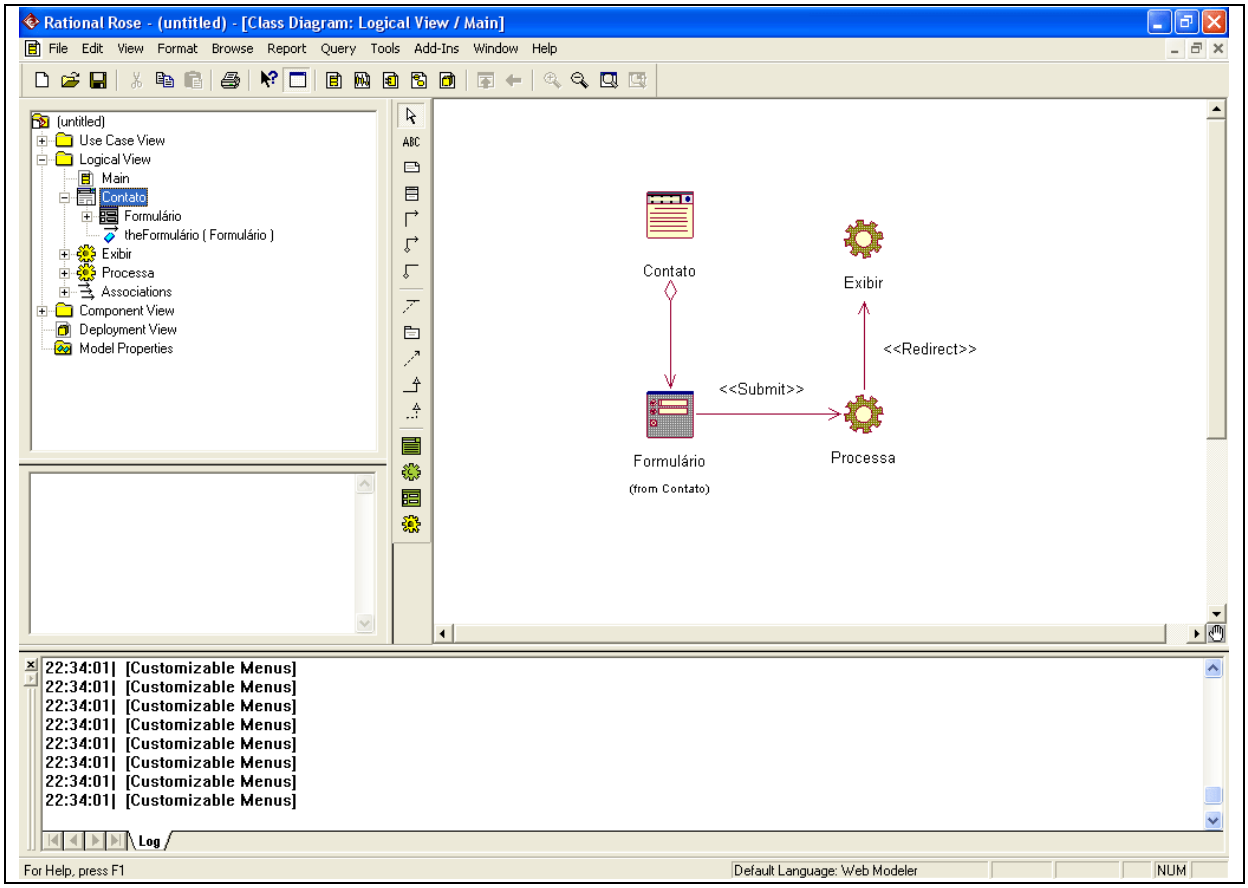


FIGURA 3 – Ambiente de desenvolvimento do Rational Rose

O produto Rational Rose é destinado ao desenvolvedor de software, apresentando um conjunto amplo de ferramentas de modelagem visual que auxiliam o processo de desenvolvimento. Sua utilização acarreta em softwares mais eficientes, robustos e de maior qualidade. A Figura 3 mostra o ambiente de desenvolvimento da ferramenta. No *web*site www.rational.com.br/rose pode ser encontrado mais informações sobre a ferramenta.

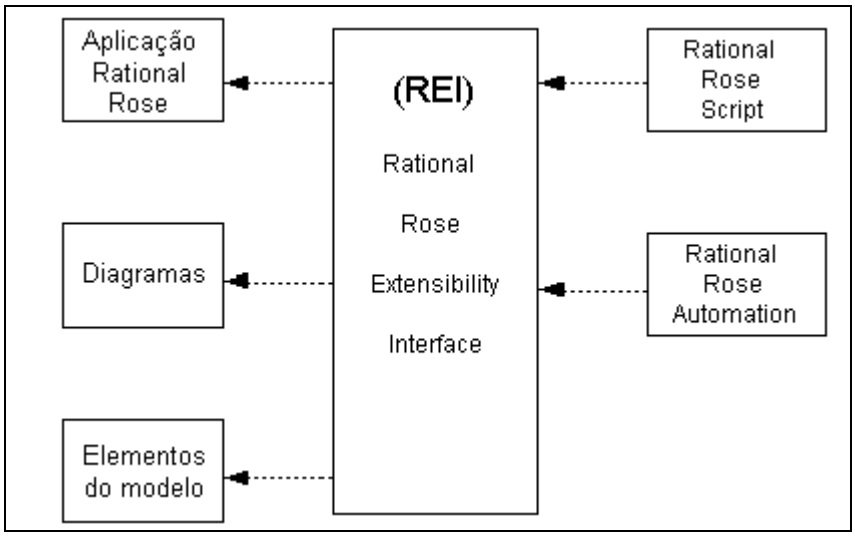
3.3.1 Criando extensões para o Rational Rose

O Rational Rose apresenta várias maneiras através das quais é possível estender e customizar suas capacidades:

a) customizar os menus;
b) automatizar funções manuais do Rational Rose através de *scripts*, como por exemplo, a criação de modelos e geração de documentação;
c) executar funções do Rational Rose a partir de outras aplicações usando o objeto Rational Rose *Automation* (RoseApp);
d) acessar as classes do Rational Rose, propriedades e métodos através de *scripts* ou da inclusão da Biblioteca de Estensão do Rational Rose;
e) ativar *add-ins* utilizando o Gerenciador de *Add-in*.

O propósito da Rational é permitir que sejam desenvolvidos aplicações baseadas em componentes (RATIONAL, 2000a). Esses componentes são definidos na Rational Rose *Extensibility Interface* (REI) *Model*. Este modelo é o conjunto dos componentes (classes, métodos, propriedades) que definem e controlam a aplicação Rational Rose e todas as suas funções. É possível se comunicar com a REI através de *scripts* ou através do Rational Rose *Automation*.

A REI está disponível através de uma API, Rational Rose *Extensibility Interface Reference*, que contém uma breve explicação sobre cada classe, lista de propriedades e métodos e descrições sobre os mesmos. Esta API está disponível no *Help* da ferramenta Rational Rose Enterprise.



Fonte: RATIONAL, 2000a

FIGURA 4 – Componentes do Rational Rose e a interface de extensão.

A Figura 4 apresenta os componentes do Rational Rose e a interface de extensão do Rational Rose, ilustrando os relacionamentos entre eles.

A linguagem de *script* do Rational Rose é uma versão estendida da linguagem BasicScript da Summit. Este tipo de extensão permite que você automatize funções do Rational Rose, e em alguns casos, crie funções que ainda não estão disponíveis através da interface do Rose com o usuário. O Quadro 4 apresenta um exemplo de código na linguagem BasicScript.

O Rational Rose Automation permite que se integre com outros softwares de duas maneiras. A primeira delas é utilizando o Rose como um controle automático, chamando um objeto OLE (*Object Linking and Embedding*) utilizando um *script*. Por exemplo, através de um *script*, poderia ser criado um controlador que utilizasse um automatismo OLE que executasse as funções do Microsoft Word.

A outra maneira é utilizar o Rose como um servidor OLE, podendo este ser chamado a partir de outros aplicativos que comportem este tipo de objetos. Como exemplo, um *script* do Microsoft Excel poderia chamar as funções do Rational Rose e interagir com este.

```
'Cria a string referente ao link
Function CreateLinkString (theAssociation As Association, theClass As
Class) As String
'Set theRole = theAssociation.GetCorrespondingRole (theClass)
      Dim theRole As Role
      Dim thisClass As Class
      Set theRole = theAssociation.GetOtherRole (theClass)

      Set thisClass = theRole.Class
      FileName$ = thisClass.GetPropertyValue ("WAE","File Name")

      If theRole.Navigable Then
            'Cria o comentário
            LinkCode$ = LinkCode$  + "          <!-- Link para a classe: " +
_
                  therole.Class.Name + " -->" + NewLine
            'Cria o link
            LinkCode$ = LinkCode$ + "           <a href=""" + FileName$
            If Len(theRole.Constraints) > 0 Then
                  LinkCode$ = LinkCode$ + "?" + theRole.Constraints
            End If
            LinkCode$ = LinkCode$ + """>" + theAssociation.Name + "</a>" +
NewLine
      End If
      CreateLinkString = LinkCode$
End Function

'Checa a existência de alguma associação com o estereótipo link
Function CheckLinkExists (theClass As Class) As String
      Dim allAssociations As AssociationCollection
      Dim theAssociation As Association

    Set allAssociations = theClass.GetAssociations ()

      If allAssociations.Count > 0 Then
            For AssID% = 1 To allAssociations.Count
                  Set theAssociation = allAssociations.GetAt(AssID%)
                  If theAssociation.Stereotype = "aLink" Then
                        LinkCode$ = LinkCode$ + CreateLinkString
(theAssociation, theClass)
                  End If
            Next AssID%
      End If

      CheckLinkExists = LinkCode$
End Function
```

QUADRO 4 – Exemplo de código *script* do Rational Rose

3.3.2 Add-Ins

*Add-in*s permitem empacotar customizações e automatizações de várias funções do Rational Rose através da Interface de Extensão (REI) em um único pacote. *Add-in* é uma coleção de alguns objetos:

a) ítens de menu;
b) ítens de atalho;
c) propriedades;
d) especificação de tipos de dados;
e) estereótipos;
f) arquivos de ajuda (help);
g) tratamento de eventos;
h) funções através de *scripts*.

Existem dois tipos de *add-ins*: básicos e de linguagem.

*Add-in*s básicos fornecem seus próprios tratamentos para eventos através de executáveis ou *scripts*. Ele não usa a visão de componentes para geração de código. *Add-ins* de linguagem utilizam a visão de componentes para definir uma linguagem alvo. Ele também tem responsabilidades que tratam eventos específicos de geração de código como *OnGenerateCode*.

Como a Rational tem a intenção de que sejam desenvolvidos extensões para o Rose, criou toda uma metodologia para que estas sejam desenvolvidas. Para se adicionar novas propriedades e ferramentas aos modelos do Rose são criados arquivos de propriedades que possuem a extensão .pty. O arquivo de propriedades possui um formato próprio apresentado no Anexo B. Cada *add-in* pode ter seu próprio arquivo de propriedades. Através da criação de novas propriedades e ferramentas, automaticamente são criadas novas abas na janela de propriedades (Figura 5) de elementos do Rational Rose. É através desta nova aba que é possível a interação do usuário com as propriedades criadas para o *add-in*, podendo assim, alterar seus valores. Os arquivos de propriedades são automaticamente habilitados ou desabilitados na medida que você habilita ou desabilita o *add-in*.

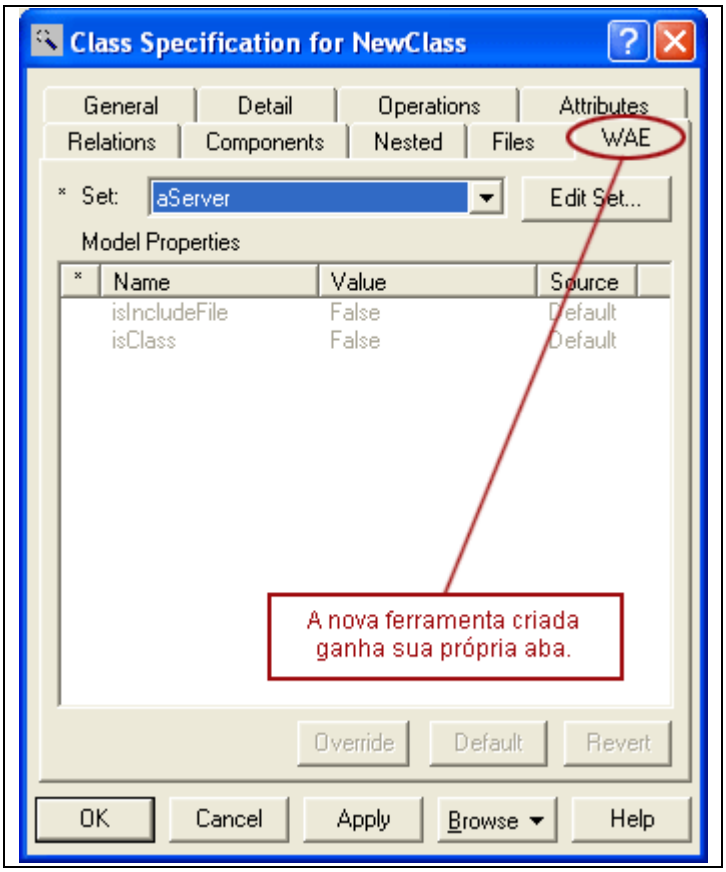


FIGURA 5 – Janela de propriedades do Rational Rose

Um arquivo de propriedades é criado a partir de um arquivo texto, utilizando qualquer editor de texto comum.

Um *add-in* pode oferecer também um conjunto de novos estereótipos, bem como um conjunto de ícones para representá-los no modelo. Os novos estereótipos a serem adicionados são definidos por um arquivo de configuração, que apresenta a extensão .ini. Nele estão contidas informações como a ferramenta a qual pertencem, localização de ícones e tipo de estereótipo.

Para que um *add-in* seja adicionado ao Gerenciador de *Add-in*s do Rational Rose, devem ser feitas as devidas inclusões no arquivo de Registro do Windows. Para isso, pode ser criado um arquivo com a extensão .reg, que fará todas as inclusões, ou criar cada inclusão separadamente, através do aplicativo regedit (Figura 6). Entre os dados que são armazenados no registro estão o diretório principal do *add-in*, nome do arquivo de propriedades, nome do arquivo de menu e forma de tratamento dos eventos.

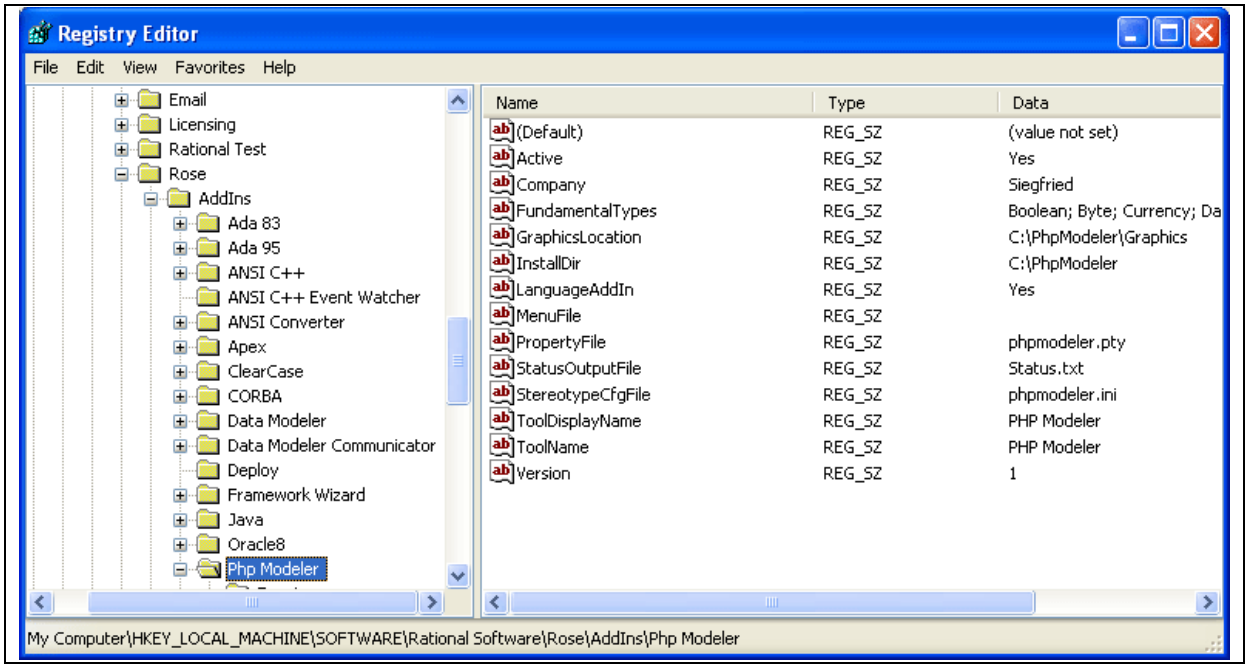


FIGURA 6 – Aplicativo regedit

Alguns dos eventos que compõe a REI podem ser tratados por *scripts*; outros precisam de um servidor OLE. Um servidor OLE é uma dll que contém funções que tratarão os eventos em questão, como por exemplo, os responsáveis pelos menus de atalho do Rational Rose.

4 DESENVOLVIMENTO DO TRABALHO

Para o desenvolvimento deste trabalho utilizou-se a ferramenta CASE Rational Rose Enterprise, onde foram implementados os *scripts* que fazem a conversão dos diagramas da UML para o código em PHP. Também fez-se necessário a criação de arquivos de configuração de *add-ins*, como arquivos de extensão *ini*, *pty* e *reg*.

4.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Os requisitos funcionais do software são:

a) interagir com a ferramenta Rational Rose Enterprise através da interface fornecida por ela, utilizando a modelagem desenvolvida;
b) permitir gerar código PHP a partir da UML;
c) gerar código para a versão 4 do PHP;
d) utilizar a notação WAE.

Como requisito não funcional, pode-se citar que o software deve utilizar a versão Rational Rose Enterprise 2002.

4.2 ESPECIFICAÇÃO

Para a especificação do protótipo foi utilizado a UML, apresentando-se um diagrama de caso de uso e o diagrama de classes – nativas da ferramenta Rational Rose – utilizadas pelos *scripts* criados. Também foi utilizado pseudo-código para especificar a lógica dos *scripts*.

4.2.1 Diagramas de Casos de Uso

No desenvolvimento deste protótipo foram identificados dois casos de uso. O primeiro refere-se à interação entre o analista e a construção dos diagramas UML. O segundo refere-se à interação do analista com a geração de código.

O analista é responsável por criar os diagramas UML dentro da notação definida pela WAE. Depois de criada modelagem, se desejado, o analista pode dar início ao processo de geração de código. Os dois casos de uso estão representados na Figura 7.

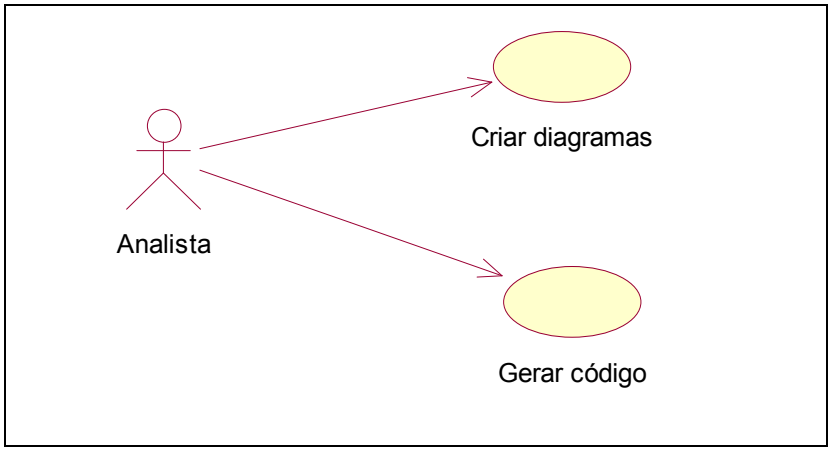


FIGURA 7 – Diagrama de casos de uso

4.2.2 Diagrama de Classes

Um *add-in* interage com as classes nativas do Rational Rose através da REI (*Rose Extensibility Interface*). Para o desenvolvimento deste protótipo foi necessária a utilização de algumas, apresentadas no diagrama de classes na Figura 8.

A classe base utilizada é a Element, é ela quem fornece a interface com as propriedades do modelo. Todo objeto numa modelagem do Rational Rose é um elemento. Cada elemento possui um único ID, e através deste ID é possível chegar a qualquer elemento do modelo e pegar suas propriedades. Para isto é utilizado o método GetUniqueID() (RATIONAL, 2000b).

RoseItem é uma especialização da Classe Element. Usa-se as propriedades e métodos desta classe para manipular documentações e estereótipos. Todas as outras classes, com exceção da classe Collection, são especializações da classe RoseItem e por isso herdam suas propriedades.

A classe Class permite retornar ou atribuir os valores às características e relacionamentos de outras classes específicas do modelo.

A classe Attributes define as características de uma classe. No protótipo foram utilizadas os atributos Type e InitValue desta classe, através dos quais se conseguia atingir as características das classes desejadas.

A classe Property define as propriedades do modelo. É utilizada no protótipo para retornar o valor de propriedades criadas na Tool (ferramenta) indicada.

A classe Role define o propósito de uma classe em cada lado de uma associação. DefaultModelProperties é a classe que contém todas as propriedades do modelo. Só existe um objeto deste por modelo.

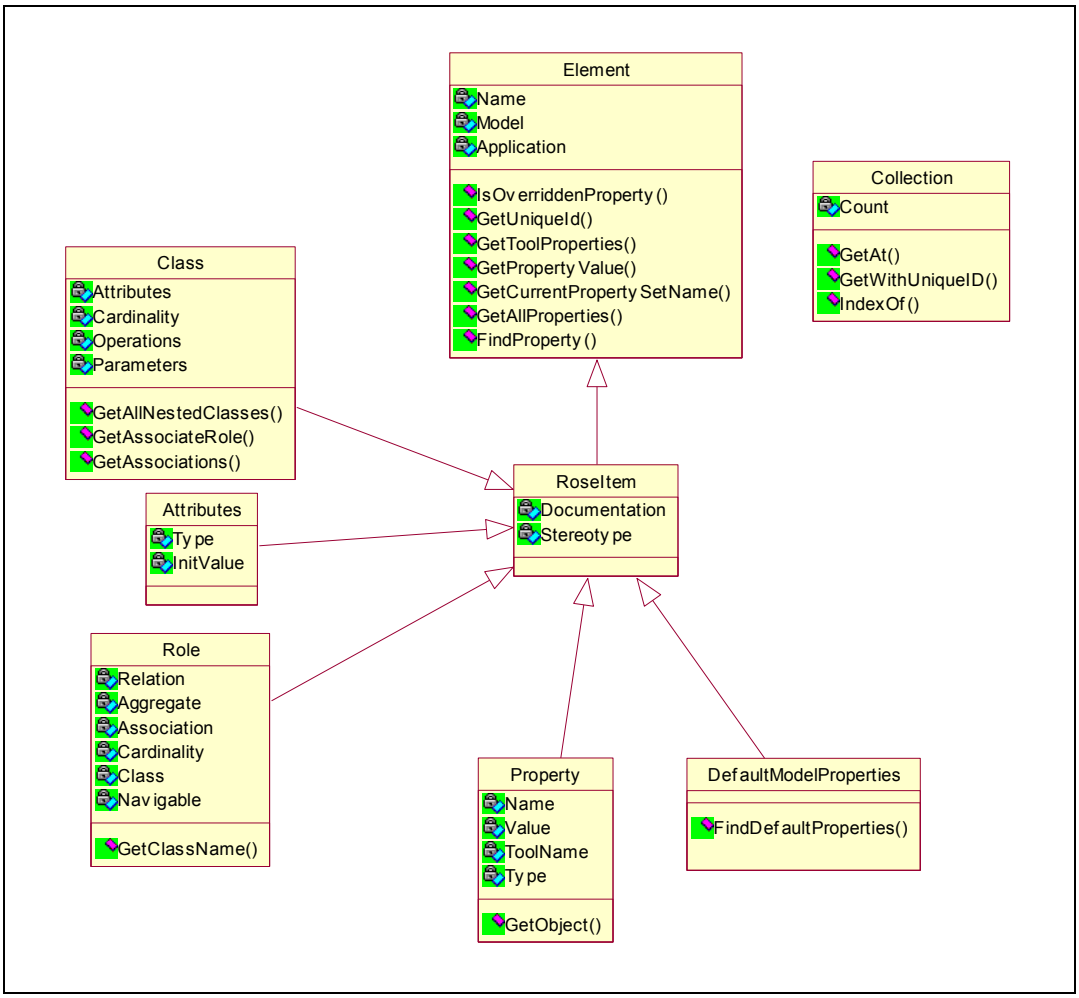


FIGURA 8 – Diagrama de classes

4.2.3 Descrição do Caso Gerar código

Paralelamente ao estudo e identificação das classes, desenvolveu-se – em pseudo-código – a lógica para tratar a modelagem e gerar código. O Quadro 5 apresenta um algoritmo inicial, mostrando a lógica de como deveria funcionar os *scripts* do *add-in*.

```
PARA cada classe do modelo FAÇA
     CASO estereótipo da classe
          "página servidor" :
               criar cabeçalho
               SE existir atributos ENTÃO
                  criar código declaração de variáveis
               SE existir operações ENTÃO
                  criar código métodos
          //se for uma classe em PHP, deve gerar o construtor
               SE classe ENTÃO
                  criar código construtor
               criar código do arquivo
               criar arquivo
          Fim
          "página cliente"
               criar cabeçalho HTML
               SE existir atributos ENTÃO
                  criar código propriedades
               SE existir links ENTÃO
                  criar código link
               SE existir formulário ENTÃO
                  criar código do formulário
               criar código do arquivo
               criar arquivo
          Fim
     Fim CASO
FIM PARA
```

QUADRO 5 – Lógica de funcionamento do *add-in*

Como cada elemento num modelo do Rose é um RoseItem, é necessário identificar cada um deles para ser tratado. O Rose, através do método GetAllClasses( ) permite que seja retornado a coleção de classes do modelo. Caso esta classe seja uma *Server page*, seus atributos são consideradas variáveis de uma página PHP. As operações desta classe se tornam métodos (*functions*). Como a linguagem PHP permite a criação de classes, existe uma propriedade que foi criada na ferramenta WAE (*isClass*), com o intuito de indicar se a classe em questão é uma classe PHP ou uma página contendo um conjunto de funções. Caso o valor desta propriedade seja verdadeiro, o código gerado representa esta classe em PHP, contendo o método construtor, os demais atributos e funções.

```
      'Faz tratamento dos atributos da classe Formulário
      Function TreatFormAttributes (Attributes As AttributeCollection) As
String
            Dim theAttribute As Attribute

            For AttID% = 1 To Attributes.Count
                  Set theAttribute = Attributes.GetAt(AttID%)
                  'Verifica o esteriótipo do atributo
                  'Input / Select / Textarea
                  'As propriedades Type, Name e InitValue são utilizadas
                  Select Case theAttribute.Stereotype
                        Case "aHTML Input"
                              AttCode$ = AttCode$ + "      <input type="""
+ theAttribute.Type + _
                              """ Name=""" + theAttribute.Name + _
                              """ Value=""" + theAttribute.InitValue + """>"
+ NewLine
                        Case "aHTML Select"
                              AttCode$ = AttCode$ + "      <select name="""
+ theAttribute.Name + _
                              """ >" + theAttribute.InitValue + _
                              "</select>" + NewLine
                        Case "aHTML Textarea"
                              AttCode$ = AttCode$ + "      <textarea
name=""" + theAttribute.Name + _
                              """ >" + theAttribute.InitValue + _
                              "</textarea>" + NewLine
                              FormType$ = "Text"
                  End Select
            Next AttID%
            TreatFormAttributes = AttCode$
      End Function
```

QUADRO 6 – Função que trata o formulário HTML

A medida que os valores das propriedade são capturados, estes são concatenados em uma *string* que representa o código do arquivo a ser criado. O Quadro 6 apresenta a função que cria o código HTML referente ao formulário.

Caso a classe do modelo contenha o estereótipo *Client page*, as propriedades da classe se tornam propriedades da página HTML, contidas na *tag* HTML *body*. É necessário também, verificar a presença de associações que partem desta classe, podendo elas ser *links*, ou indicando a existência de um formulário HTML. Quando existir um formulário HTML, o código deste é embutido no código da página HTML que o contém.

Quando uma classe apresenta o estereótipo *form* HTML, seus atributos representam os objetos de um formulário HTML. Estes podem ser do tipo *input*, *select* ou *textarea*. O formulário normalmente vem acompanhado da página que receberá sua informação, representado pela associação *submit*.

4.3 IMPLEMENTAÇÃO

Os *scripts* que compõem o protótipo foram implementados no ambiente de desenvolvimento da ferramenta Rational Rose apresentado na Figura 9.

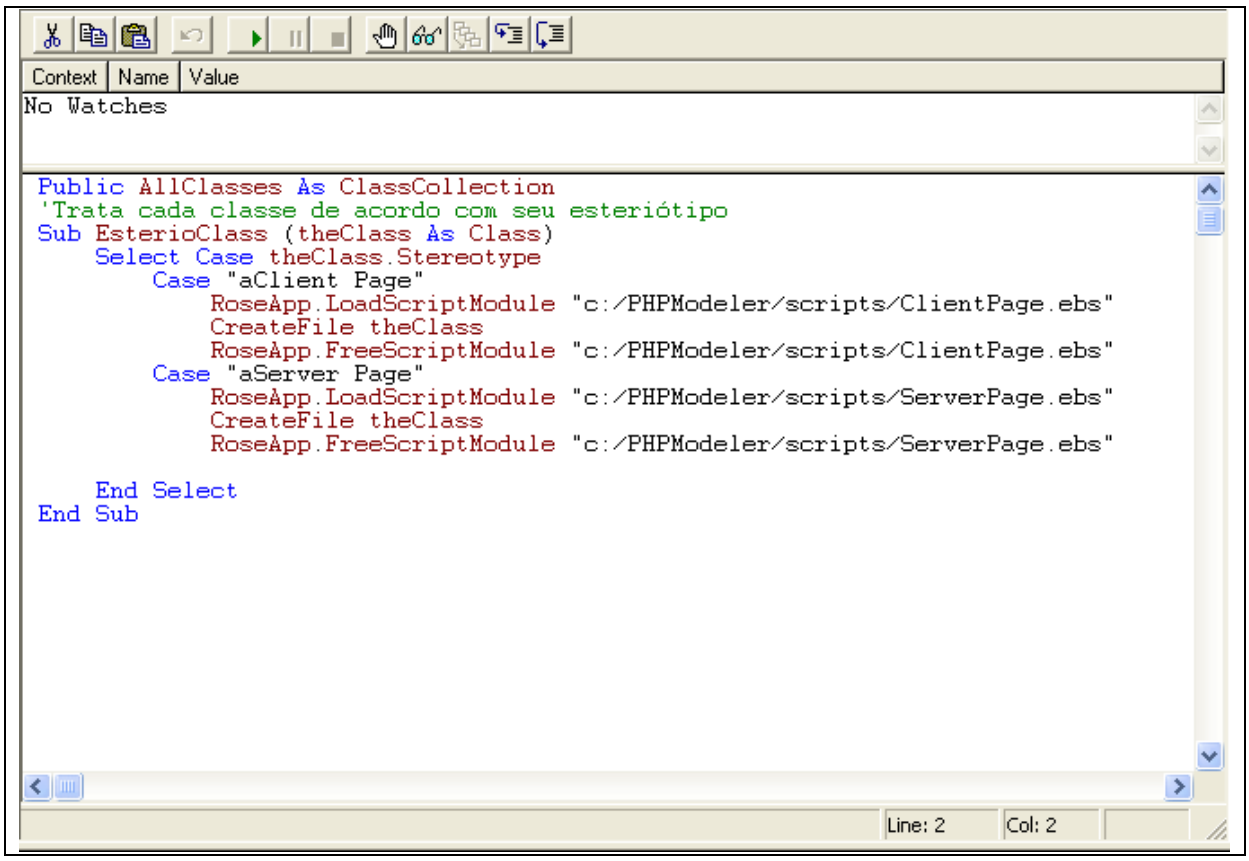


FIGURA 9 – Ambiente de desenvolvimento de *scripts* do Rational Rose

O diagrama de classes e a lógica representada através de pseudo-código foram sendo atualizadas a medida que o estudo sobre a ferramenta avançava. Tomando estas especificações como base, criava-se os *scripts* que fazem o tratamento do modelo para a geração de código.

Para o tratamento de alguns dos eventos que compõe a REI, houve a necessidade de se criar um servidor OLE. Um servidor OLE é implementado como uma dll. Para sua compilação, foi utilizada a ferramenta Visual Basic 6.0. Para que o servidor OLE funcione corretamente é necessário registrá-lo utilizando o programa Regsvr32.exe. O Anexo D apresenta o código fonte da dll que implementa o servidor OLE utilizado no protótipo. Uma das funções do servidor OLE implementado é alterar em tempo de execução os estereótipos de associação criados pelo usuário.

Para completar o *add-in*, houve a necessidade de criar os estereótipos que fazem parte da WAE. Os desenhos foram feitos utilizando a ferramenta gráfica Corel Draw e então, exportados no formato *Enhaced Windows Metafile* (*.emf*). Os outros arquivos de configuração, como as propriedades criadas (arquivo *.pty*), arquivo de configuração do *add-in* (*.ini*) e para adicionar os campos no registro do Windows (*.reg*) são arquivos de texto, criados utilizando o Notepad.

Conforme apresentado no capítulo 3, cada arquivo de configuração tem um formato específico e deve ser criado nestes padrões para que executem corretamente. Os ícones adicionados à barra de ferramentas do Rational Rose foram criados utilizando a ferramenta Adobe Photoshop.

Com a criação do arquivo de propriedades, foi adicionado à janela de propriedades de cada item, uma aba com o título WAE. Nesta aba, encontram-se quatro conjuntos de propriedades diferentes:

a) default:
   - *File Name*: nome do arquivo que será criado;

b) aClientPage:
   - contém um grupo de propriedades da *tag* HTML *body*, que quando possuem seu valor alterado, serão acrescentados na geração de código. Dentre esta propriedades estão *background*, *topmargin*, *leftmargin*, etc;

c) aServerPage:
   - *isClass*: indica que esta página servidor é uma classe PHP.

d) aForm HTML:
   - *Method*: método usado para enviar dados ao URL em atividade.

O Anexo E apresenta o código do arquivo de propriedades utilizado pelo protótipo.

4.4 LIMITAÇÕES DO PROTÓTIPO

O protótipo desenvolvido engloba somente os estereótipos básicos da WAE. Isto inclui:

a) classe Server Page;
b) classe Client Page;
c) classe Form HTML;

d) classes de associação: <<*link*>>, <<submit>>, <<build>>, <<include>>.

O desenvolvimento do protótipo ficou limitado somente à visão lógica da aplicação, não incorporando a visão de componentes. Há também limites quanto a usabilidade da ferramenta, visto que o código gerado depende da correta modelagem, seguindo a especificação da WAE, fazendo-se necessário que o usuário esteja familiarizado com a mesma.

4.5 OPERACIONALIDADE DA IMPLEMENTAÇÃO

A maior parte da interação com o *add-in* se dá através da janela de propriedades de cada item. Para acessar a janela de propriedades deve-se selecionar o item desejado e pressionar F4, ou através do duplo-clique no item.

Para selecionar a notação utilizada pelo protótipo criado, deve-se acessar as propriedades do modelo atual (pressionando F4). Na janela de propriedades, na aba *Notation*, na propriedade *Default Language*, deve-se selecionar a opção PHP Modeler.

A criação das classes e associações é realizada utilizando a barra de ferramentas do Rational Rose. A Figura 10 apresenta os ícones das ferramentas que são utilizadas pelo protótipo.

Para se alterar o estereótipo uma classe do modelo, deve-se acessar suas propriedades. Na aba *General*, está a propriedade *Stereotype*, através da qual pode se escolher:

a) aClient Page para página cliente;
b) aServer Page para página servidor;
c) aForm HTML para um formulário HTML.

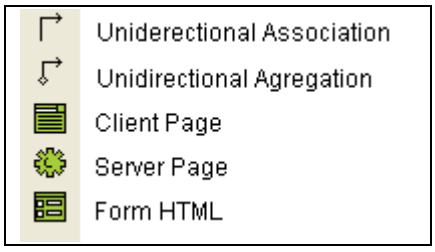


FIGURA 10 – Barra de ferramenta

Quando o item for uma associação, pode-se mudar seu estereótipo para:

a) aLink;
b) aSubmit;
c) aBuild;
d) aInclude.

No caso do estereótipo ser um aForm HTML, seus atributos são responsáveis por representar os objetos de um formulário conforme a Tabela 1.

Tabela 1 – Estereótipos dos atributos de um formulário

Estereótipo	**Type**	**Init Value**
aHTML Input	*text*, *radio*, *checkbox*, *button*, *file*, *image*	
aHTML Select		
aHTML Textarea		Texto inicial apresentado

O modelo pode conter classes e associações que não condizem com as especificações da WAE. Na geração de código, estes elementos serão ignorados.

4.5.1 Exemplo 1

Para apresentar a funcionalidade do protótipo, será modelado um fragmento de um *website* institucional de uma empresa. Serão criados as seguintes classes no modelo:

a) *Homepage*: *client page* que representa a página inicial do *web*site;
b) Contato: *client page* através da qual o usuário entraria em contato com a empresa;
c) FormContato: *form* HTML que contém os campos utilizados para o contato;
d) Enviar: *server page* que trata os campos da mensagem, podendo retornar uma mensagem de sucesso ou de erro quando ocorrer algum erro.

Para criar a classe *Homepage*, seleciona-se o ícone de *Client page* e clica-se na área do modelo. Na sua janela de propriedades deve-se alterar o nome do arquivo, encontrado no aba WAE, no *set default* (Figura 11).

A página Contato é criada seguindo a mesma metodologia e altera-se as seguintes propriedades:

a) *Name*: Contato;
b) *Stereotype*: aClient Page;
c) *Documentation*: Página de contato da empresa;

d) *File Name*: contato.htm.

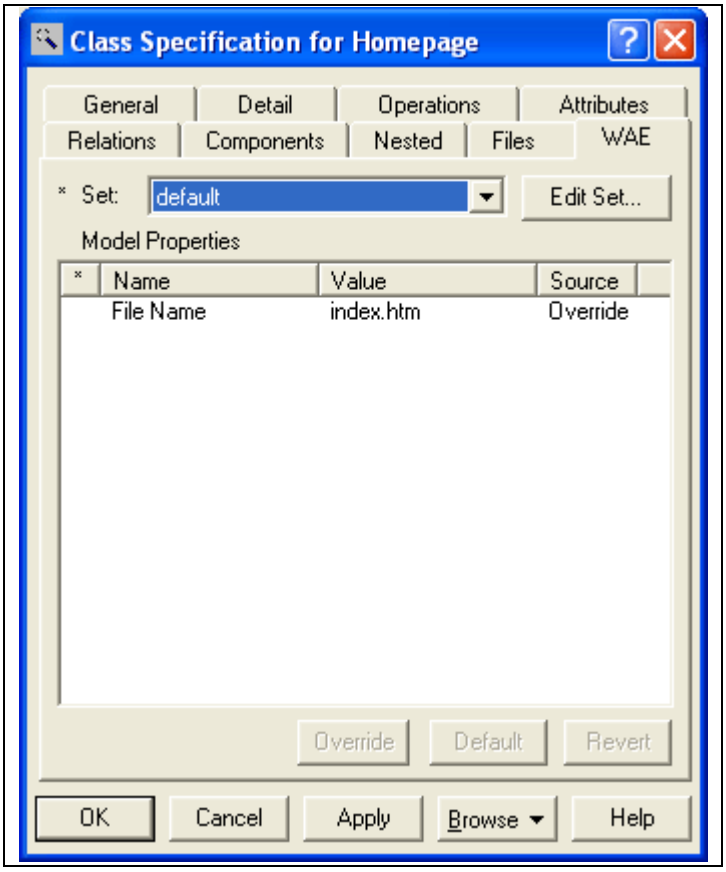


FIGURA 11 – Alterando o nome do arquivo

Entre estas duas páginas (*Homepage* e Contato) há uma associação *Link*. Para isso, seleciona-se a ferramenta *Unidirectional Association* na barra ferramentas e liga-se as duas classes. As propriedades alteradas nessa associação são:

a) *Name*: Contato;

b) *Stereotype*: aLink.

A página que contém o formulário é a Contato. Para se criar esta classe no modelo deve-se clicar com o botão direito sob a classe Contato, e selecionar a opção Criar Formulário (PHP Modeler > Criar Formulário). Será criado um formulário e a associação de agregação com a página HTML Contato. Para a classe formulário aparecer no modelo, deve-se acessar a janela de propriedades da página Contato, selecionar a aba *Nested*, e arrastar a classe *Form* para a área de trabalho do Rose.

Para criar os atributos que representam os campos do formulário HTML, deve-se clicar com o botão direito sob a classe formulário e selecionar o objeto de formulário que deseja-se criar, Input ou Textarea (PHP Modeler > Objetos do Formulário). Os campos criados são:

a) Nome (aHTML Input);
b) Email (aHTML Input);
c) Destino (aHTML Select);
d) Mensagem (aHTML Textarea).

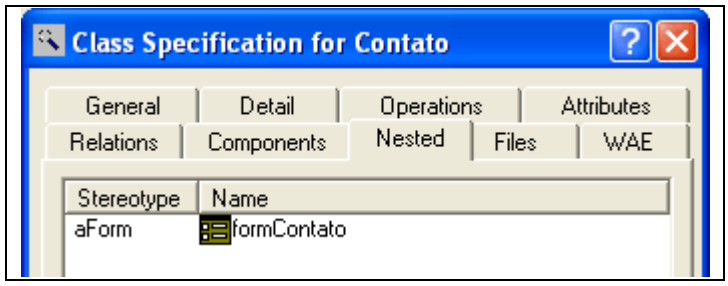


FIGURA 12 – Classe contato que contém o formulário formContato

Por último será criada a classe Enviar. Adiciona-se uma classe ao modelo e altera-se suas propriedades:

a) *Name*: Enviar;
b) *Stereotype*: aServer Page;
c) *File Name*: enviar.php;

O método Enviar é acrescentado a esta classe. Este método equivale a função Enviar que deve ser criada para tratar os dados do formulário. O modelo final é mostrado na Figura 13.

Após a modelagem completa, o usuário pode clicar em cima de qualquer um dos itens do modelo com o botão direito, selecionar entre as opções do menu: Gerar Código e Configurar Diretório. Configurar Diretório é utilizado para escolher o diretório onde serão gravados os arquivos que contém o código gerado. Este estudo de caso gera três arquivos, index.htm (Quadro 7), contato.htm (Quadro 8) e enviar.php (Quadro 9).

Os arquivos que contém o código HTML estão prontos para execução, mas seu código abstrai qualquer elemento gráfico, fazendo necessário que seu visual seja montado

implementado pelo desenvolvedor. A figura 14 exemplifica uma página montada a partir do código gerado. O Quadro 11 apresenta seu respectivo código HTML.

O arquivo PHP, para se tornar executável, necessita da manutenção do desenvolvedor. Neste exemplo, há a necessidade de implementar a função Enviar( ) e as operações para o envio do email. De forma simplificada, o código implementado é apresentado no Quadro 10.

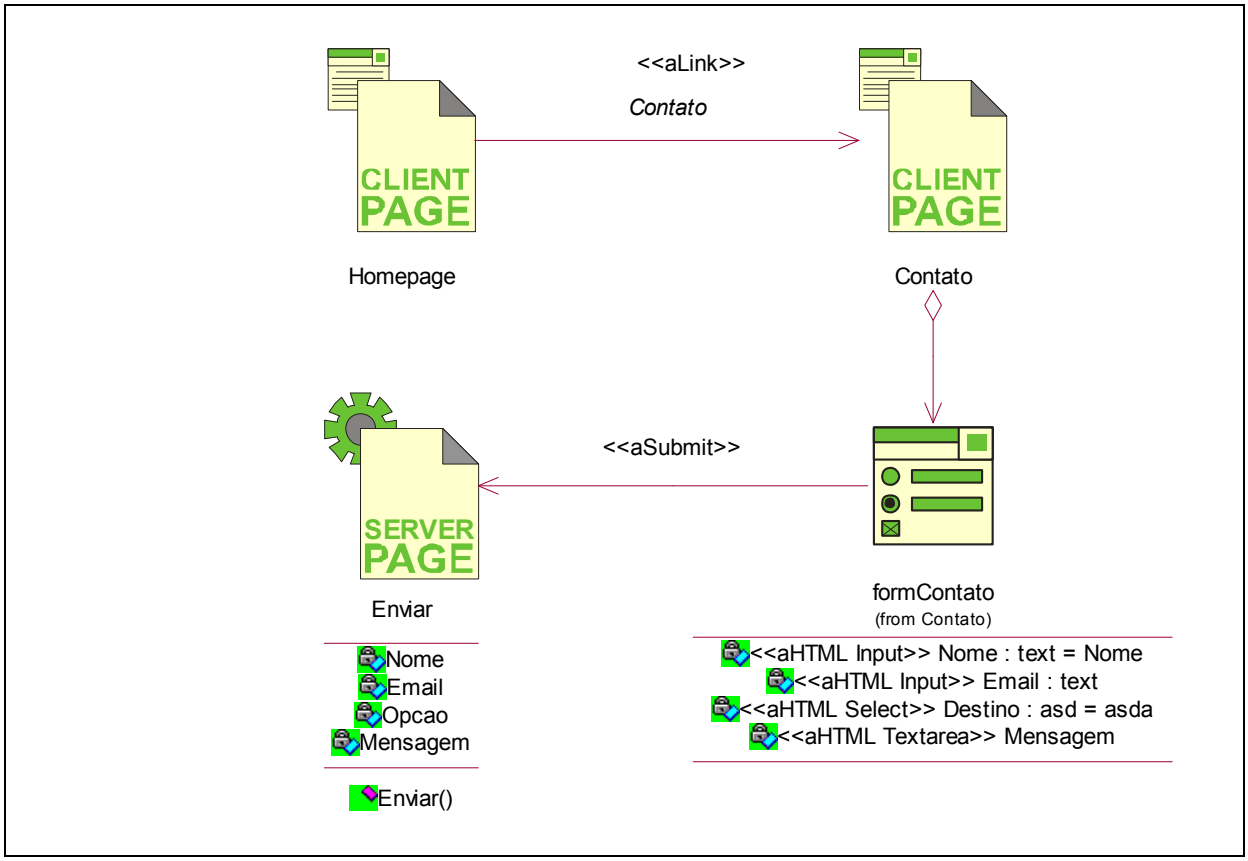


FIGURA 13 – Modelagem completa

```
<!—Página de contato do website -->
<HTML>
      <BODY >
      <FORM Method="post" Action="enviar.php" >
            <input type="text" Name="Nome" Value="Nome">
            <input type="text" Name="Email" Value="Email">
            <select name="Destino" > </select>
            <textarea name="Mensagem" >Mensagem</textarea>
            <input type="submit">
      </FORM>
      </BODY>
</HTML>
```

QUADRO 7 – Código da Homepage

```
<!-- Página inicial do website -->
<HTML>
      <BODY >
            <!-- Link para a classe: Contato -->
            <a href="contato.htm">Contato</a>
      </BODY>
</HTML>
```

QUADRO 8 – Código da página Contato

```
<?
$Nome;
$Email;
$Opcao;
$Mensagem;

function Enviar() {

}

?>
```

QUADRO 9 – Código da página Enviar

```
<?
$Nome;
$Email;
$Opcao;
$Mensagem;

function Enviar($m) {
      return(mail("destino@netuno.com.br","Contato",$m));
}

if (Enviar($Mensagem))
   {
   echo"Email enviado";
   }
else
   {
   echo"Ocorreu um erro";
   }
?>
```

QUADRO 10 – Código alterado da página Enviar

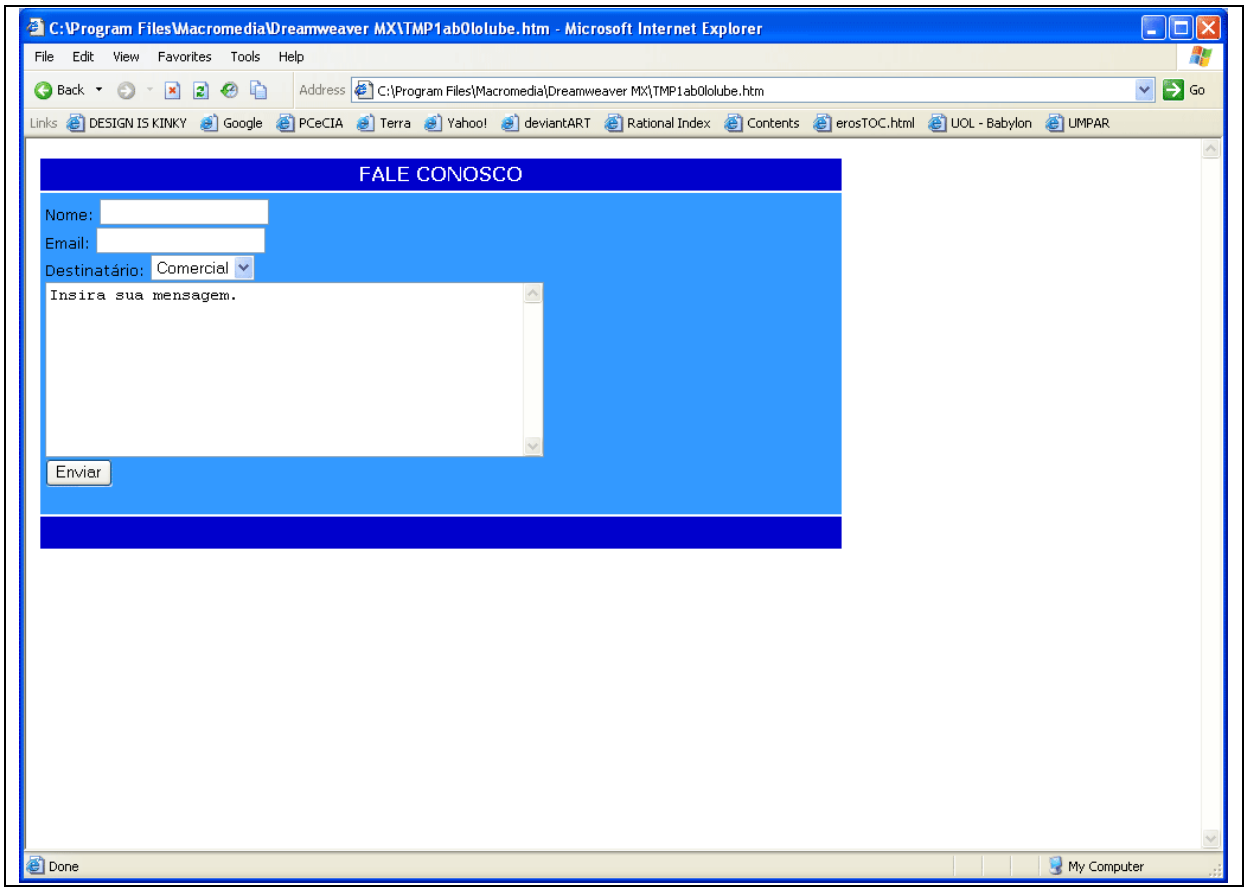


FIGURA 14 – Página de contato alterada

```
<!-Página de contato do website -->
<HTML>
      <BODY >
<table width="70%" border="0" cellpadding="4" cellspacing="2"
bordercolor="#CCCCCC" bgcolor="#FFFFFF">
  <tr>
    <td bgcolor="#0000CC"><div align="center"><strong><font color="#FFFFFF"
size="3" face="Verdana, Arial, Helvetica, sans-serif">FALE
        CONOSCO</font></strong> </div></td>
  </tr>
  <tr>
    <td bgcolor="#3399FF">
      <FORM Method="post" Action="enviar.php" >
        <p> <font size="2" face="Verdana, Arial, Helvetica, sans-
serif">Nome:</font>
          <input type="text" Name="Nome">
          <br>
          <font size="2" face="Verdana, Arial, Helvetica, sans-
serif">Email:</font>
          <input type="text" Name="Email">
          <br>
          <font size="2" face="Verdana, Arial, Helvetica, sans-
serif">Destinat&aacute;rio:</font>
          <select name="Destino" >
```

```
            <option selected>Comercial</option>
            <option>Cria&ccedil;&atilde;o</option>
          </select>
          <br>
          <textarea name="Mensagem" cols="50" rows="9" >Insira sua
mensagem.</textarea>
          <br>
          <input name="submit" type="submit" value="Enviar">
        </p>
      </FORM></td>
  </tr>
  <tr>
    <td bgcolor="#0000CC"> </td>
  </tr>
</table>
</BODY>
</HTML>
```

QUADRO 11 – Código HTML da página Contato

4.5.2 Exemplo 2

Este exemplo é apresentado de forma a apresentar a capacidade do gerador em gerar as classes na linguagem PHP, incluindo seus construtores. Sua função é ilustrativa. Para tanto, será criada a classe Carro, que contém dois atributos:

a) Modelo, com valor padrão Gol;

b) Cor, com valor padrão Vermelho.

A classe também apresenta o método Acelerar( ). A modelagem criada no Rational Rose é apresentada na Figura 15. Como neste exemplo deseja-se representar uma classe em PHP, deve-se alterar a propriedade *isClass* para *true*. Fazendo isso, o gerador automaticamente criará o código para a função construtora. O código gerado para esta classe pode ser visto no Quadro 12.

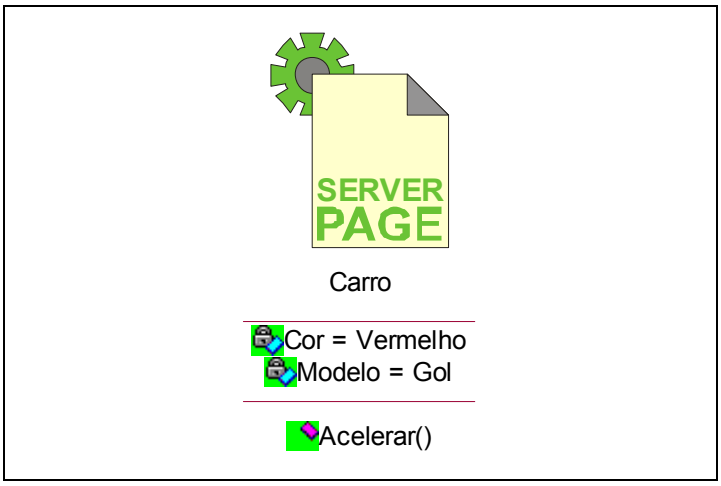


FIGURA 15 – Classe Carro

```
<?
class Carro {

	function Carro ($Cor="Vermelho",$Modelo="Gol") {
		this->Cor = $Cor;
		this->Modelo = $Modelo;
	}

	function Acelerar( ) {

	}
?>
```

QUADRO 12 – Código gerado para a classe Carro

4.6 RESULTADOS E DISCUSSÃO

O *add-in* criado permite a geração do código PHP, e HTML. A utilização dos arquivos gerados pelo *add-in* facilitam a codificação, principalmente por estarem totalmente de acordo com a especificação.

As classes incorporadas no *add-in* permitem a modelagem de aplicações básicas, de pequena complexidade. Para torná-lo comercialmente viável, há a necessidade de adicionar outros estereótipos que compõe a WAE, além da geração de código baseado na visão de componentes. O código gerado necessita de excessivas alterações, mas cumpre com o objetivo de análise.

Os resultados desejados com este trabalho foram alcançados. Comparando o funcionamento do protótipo com o *add-in* nativo do Rational Rose, que gera código para ASP

e JSP, é possível observar que ambos apresentam resultados semelhantes em nível de código gerado. O Anexo 3 apresenta o mesmo estudo de caso e seu respectivo código gerado (em ASP) através do Rational Rose.

O código gerado a partir do protótipo apresenta características semelhantes com o gerado pelos *add-ins* que acompanham a ferramenta Rational Rose. Em sua maioria, eles geram apenas a estrutura da aplicação, fazendo-se necessário codificações adicionais para tornar o código executável.

5 CONCLUSÕES

Através deste trabalho pode-se aprimorar os conhecimentos sobre a ferramenta CASE Rational Rose e suas capacidades de extensão. Este estudo permitiu a percepção de que, muitas vezes, aproveitar as ferramentas existentes adicionando-se novas funcionalidades pode ser mais adequado do que o desenvolvimento de um novo aplicativo.

Verificou-se que a utilização da UML para a especificação de aplicações *web* facilita o entendimento das mesmas, garante uma qualidade maior do aplicativo e ajuda a evitar erros durante a implementação. Outra vantagem é que, munidos da especificação da aplicação, a manutenção se torna uma tarefa mais fácil entre uma equipe de desenvolvedores.

Pode-se perceber também através das pesquisas realizadas, que esta área está em ascensão visto a grande quantidade de ferramentas de modelagem que começam a incorporar a WAE em seu conjunto de estereótipos. Infelizmente, ainda não é comum sua utilização, provado através da escassez de literatura e exemplos.

Sobre a geração de código conclui-se que pode ser muito útil se bem utilizada. Focando sua utilização em aplicações *web*, assim como a UML, evita que erros de implementação sejam cometidos e agiliza o processo de codificação. Mesmo havendo a necessidade de codificação manual, a conciliação entre especificação e codificação, facilita muito o entendimento das aplicações.

Por último, foram atingidos os objetivos com o desenvolvimento do protótipo. Foi desenvolvido um *add-in*, que gera código PHP a partir da ferramenta CASE Rational Rose. Para alcançar o objetivo, foram criados os arquivos de configuração do *add-in* e também um conjunto de *scripts*. Além disso, pode-se estudar e compreender melhor a metodologia de desenvolvimento de um *add-in*, área que apresenta uma carência muito grande de material e exemplos para ajudar e incentivar esse tipo de desenvolvimento.

5.1 EXTENSÕES

O protótipo desenvolvido neste trabalho utiliza somente a visão lógica do modelo Rational Rose, constituindo-se em um *add-in* básico. Como sugestão de extensão, pode-se desenvolver um *add-in* de linguagem, tirando-se proveito da visão de componentes. Além

disto, pode-se ampliar a quantidade de estereótipos suportados, abrangendo cada vez mais elementos da WAE.

Outra sugestão é o desenvolvimento de um gerador de código utilizando a capacidade do Rose de trabalhar com Frameworks.

REFERÊNCIAS BIBLIOGRÁFICAS

ANSELMO, Fernando. **PHP e MySQL para Windows**. Florianópolis: Visual Books, 2000.

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML**: guia do usuário. Rio de Janeiro: Campus, 2000.

CASTAGNETTO, Jesus et al. **Professional PHP programando**. São Paulo: Makron books, 2001.

CONALLEN, Jim. **Building *web* application with UML**. Harlow: Addison Wesley Longman, 1999.

CONALLEN, Jim. **Desenvolvimento de aplicações *Web* com UML**. Rio de Janeiro: Campus, 2003.

CONALLEN, Jim. **Modeling *web* applications with UML**. [S.l.], 1999. Disponível em: <www.conallen.com/whitepapers/*web*apps/Modeling*Web*Applications.htm>. Acesso em: maio 2002.

HERRINGTON, Jack. **Are You Missing Out on Code Generation?** Disponível em: <http://www.devx.com/java/editorial/15511> [S.l.], 2003. Acesso em: maio 2003.

MATOS, Alexandre Veloso de. **UML**: prático e descomplicado. São Paulo: Érica, 2002.

OFFUT, Jeff. Quality attributes of *web* software applications. **IEEE Software**, Califórnia, v. 19, n. 2, p. 25-32, mar 2002.

PAGE-JONES, Meilir. **Fundamentos do desenho orientado a objeto com UML**. São Paulo: Makron Books, 2001.

RATIONAL, **Rational Rose 2000e, Rose Extensibility Reference**. Disponível em: < http://www.cs.hmc.edu/tech_docs/qref/rational/DevelopmentStudioUNIX.2000.02.10/docs/html/rose_REI_ref/Preface.html#982088> [S.l], 2000. Acesso em: jan 2003.

RATIONAL, **Rational Rose 2000e, Rose Extensibility User's Guide**. Disponível em: < http://www.cs.hmc.edu/tech_docs/qref/rational/DevelopmentStudioUNIX.2000.02.10/docs/html/rose_REI_guide/REIGdePreface.html#474898> [S.l], 2000. Acesso em: jan 2003.

SOARES, Walace. **Programando em PHP**: conceitos e aplicações. 2. ed. São Paulo: Érica, 2000.

Anexo A – Especificação da extensão WAE

Este apêndice contém a especificação dos elementos mais comuns que compõe a WAE. Os estereótipos de associação são apresentados na Tabela 2.

Tabela 2 – Estereótipos de associação

Estereótipo	Descrição
<<*link*>>	Um relacionamento entre uma página do cliente e um recurso do lado do servidor ou página *web*. O objetivo pode ser uma classe *client page* ou uma classe *server page*. Uma associação <<*link*>> é uma abstração do elemento âncora HTML, quando o atributo href está definido nele. Um valor *tag* importante é definido para essa associação estereotipada: *parameters*. Ese valor contém parâmetros passados juntamente com a solicitação HTTP. Esse valor de *tag* é formatado como uma seqüência de caracteres e pode conter codificações de implementação específica.
<<*build*>>	Um relacionamento direcional entre uma página do servidor e uma página do cliente. Esse relacionamento identifica a saída HTML de uma execução da página do servidor.
<<*submit*>>	Um relacionamento direcional entre um <<HTML *form*>> e uma página do servidor. Semelhante a um relacionamento <<*link*>>, se refere ao recurso do lado do servidor. Porém, quando o recurso é solicitado a partir do servidor, todos os atributos de campo do formulário são enviados, juntamente com a solicitação na qual são processados.
<<*redirect*>>	Um relacionamento direcional entre uma página do cliente ou página do servidor e outra página. Essa associação indica um comando ao cliente para solicitar outro recurso.
<<*forward*>>	Um relacionamento direcional entre uma página do servidor e outra página do servidor ou página do cliente. Essa associação representa a delegação de processamento de uma solicitação do cliente de um recurso para outra página do lado do servidor.
<<*object*>>	Um relacionamento de confinamento desenhado a partir de uma página cliente para outra classe lógica, normalmente uma que represente um *applet*, controle ActiveX ou outro componente que possa ser embutido. Essa associação abstrai os elementos HTML <*object*> e <*applet*>.
<<*include*>>	Uma associação direcional de uma classe <<*server page*>> para outra classe <<*server page*>> ou <<*client page*>>. Durante a montagem da página em tempo de execução, essa associação indica que a página incluída é processada, se for dinâmica, e que seus conteúdos ou subprodutos são usados pelo pai.

Fonte: Conallen, 2003.

A seguir segue uma descrição dos elementos que caracterizam a visão lógica do modelo, formado principalmente por classes estereotipadas. A WAE define três classes principais apresentadas na Tabela 3.

Tabela 3 – Estereótipos de classe da WAE

Página do Servidor	
Descrição	Um servidor representa uma página*w eb* dinâmica que contém o conteúdo agrupado no servidor sempre que é solicitado. Normalmente, uma página do servidor contém *scripts* que são executados pelo servidor que interagem com os recursos do lado do servidor: banco de dados, componentes de lógica do negócio, sistemas externos e assim por diante. As operações do objeto representam as funções no *script* e seus atributos representam variáveis que são visíveis no escopo da página, acessível por todas as funções na página.
Restrições	As páginas de servidor podem ter apenas relacionamentos normais com objetos no servidor.
Ícone	
Página do cliente	
Descrição	Uma instância da página do cliente é uma página *Web* formatada em HTML com uma mistura de dados, apresentação e ainda lógica. As páginas de cliente são apresentadas pelos navegadores de clientes e podem conter *scripts* que sejam interpretados pelo navegador. As funções da página do cliente mapeiam para funções em *tags* na página. Os atributos da página do cliente mapeiam para variáveis declaradas nas *tags* de *scripts* da página que são acessíveis por qualquer função na página ou limite de escopo da página. As páginas do cliente podem ter associações com outras páginas do cliente e do servidor
Ícone	
Formulário HTML	
Descrição	Uma classe estereotipada como uma <<*form*>> é uma coleção de campos de entrada que são parte de uma página do cliente. Essa classe mapeia diretamente para a *tag* <*form*> HTML. Seus atributos representam os campos de entrada do formulário HTML: caixas de seleção e campos ocultos. Uma classe <<*form*>> não possui operações, visto que não pode ser encapsulada em um formulário. Qualquer operação que interaja com o formulário seria a propriedade da página que contém o formulário.
Valores com *tags*	GET ou POST: o método usado para enviar dados ao URL em atividade.
Ícone	

Fonte: Adaptado de Conallen, 2003.

Anexo B – Sintaxe de arquivos .pty

```
# Inicia informação sobre versão
(object Petal
            version    number
            _written   "add-in name"
            charSet    0)

# inicia a definição da ferramenta (Tool)
(list Attribute_Set
            # Tool setup
            (object Attribute
                  tool    "tool"
                  name    "propertyID"
                  value   "809135966")

            # Inicia definição dos elementos
            (object Attribute
                  tool    "tool"
                  name    "set__model element"
                  value   (list Attribute_Set
                              # Define first property
                              (object Attribute
                                    tool    "tool"
                                    name    "property"
                                    value   datatype)

                              # Define second property
                              (object Attribute
                                    tool    "tool"
                                    name    "property"
                                    value   datatype)
                              ...

                              # Define nth property
                              (object Attribute
                                    tool    "tool"
                                    name    "property"
                                    value    datatype)
                  )
            )

            # Inicia definição de novo elemento
            (
                  ...
            )
)
# Fim da definição da ferramenta
```

QUADRO 13 – sintaxe de arquivos .pty

O Quadro 13 apresenta a sintaxe para a criação de arquivos de propriedades de *add-in*. Dentro do mesmo arquivo podem ser criados vários conjuntos de propriedades, desde que se obedeça ao mesmo formato.

Anexo C – Estudo de caso

A Figura 16 apresenta a modelagem feita utilizando a notação WAE nativa do Rational Rose. O Quadro 14 apresenta o código gerado em ASP dos arquivos.

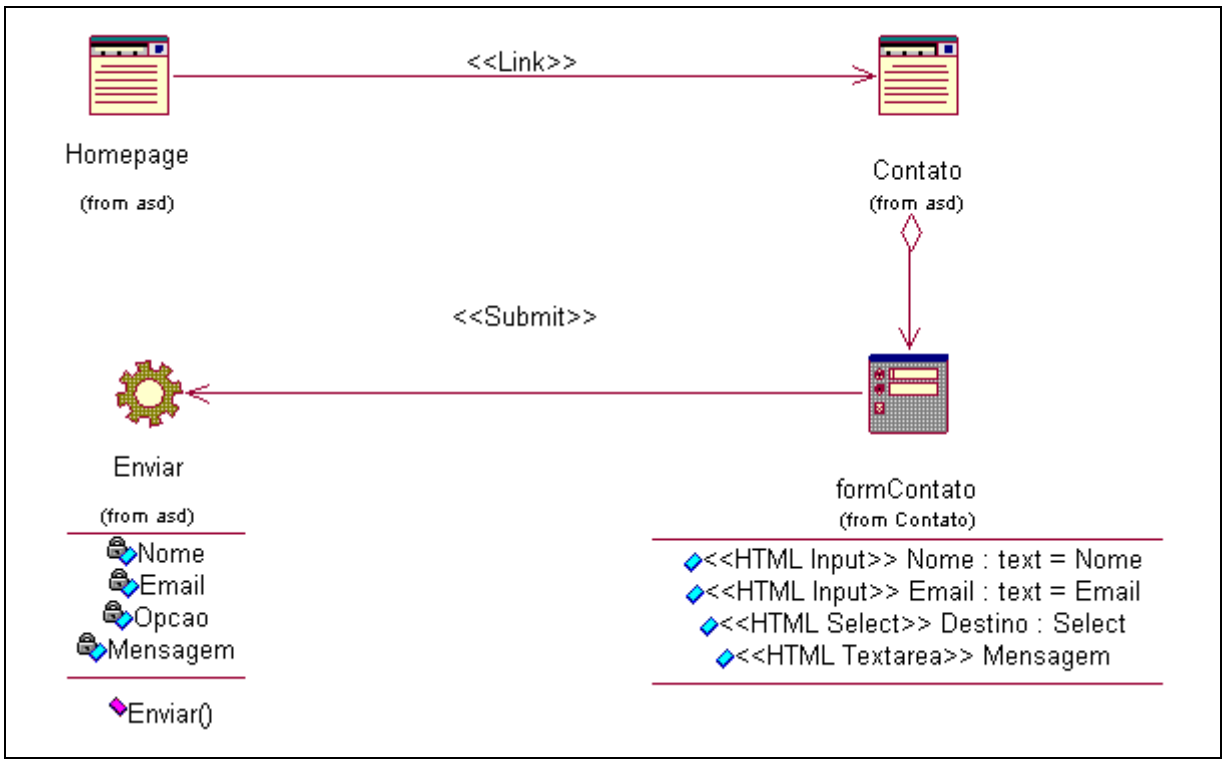


FIGURA 16 – Modelagem utilizando notação nativa do Rational Rose

```
Arquivo ENVIAR.ASP
<%
Function Enviar()
End Function

%>

ARQUIVO Homepage.html
<html>
      <body>
            <a HREF="Contato.html"></a>
      </body>
</html>

ARQUIVO Contato.html
<html>
      <body>
            <form Action="Enviar.asp" Name="formContato">
                  <textarea Name="Mensagem">
                  </textarea>
                  <select Name="Destino">
                  </select>
                  <input Name="Email" Type="text" Value="Email">
                  <input Name="Nome" Type="text" Value="Nome">
            </form>

      </body>
</html>
```

QUADRO 14 – Código ASP gerado pelo Rational Rose

Anexo D – Código fonte do servidor OLE

O Quadro 15 apresenta o código fonte da dll que implementa o servidor OLE do protótipo.

```
Public AllClasses As RoseClassCollection
Dim GenerateCode As RoseContextMenuItem
Dim CreateForm As RoseContextMenuItem
Dim CreateFormObj1 As RoseContextMenuItem
Dim CreateFormObj2 As RoseContextMenuItem

Private Const BIF_RETURNONLYFSDIRS = 1
Private Const BIF_DONTGOBELOWDOMAIN = 2
Private Const MAX_PATH = 260

Private Declare Function SHBrowseForFolder Lib "shell32" (lpbi As
BrowseInfo) As Long

Private Declare Function SHGetPathFromIDList Lib "shell32" (ByVal pidList
As Long, ByVal lpBuffer As String) As Long

Private Declare Function lstrcat Lib "kernel32" Alias "lstrcatA" (ByVal
lpString1 As String, ByVal lpString2 As String) As Long

Private Type BrowseInfo
   hWndOwner As Long
   pIDLRoot As Long
   pszDisplayName As Long
   lpszTitle As Long
   ulFlags As Long
   lpfnCallback As Long
   lParam As Long
   iImage As Long
End Type
Private Function BrowseFolder() As String
'Opens a Browse Folders Dialog Box that displays the
'directories in your computer
Dim lpIDList As Long ' Declare Varibles
Dim sBuffer As String
Dim szTitle As String
Dim tBrowseInfo As BrowseInfo

szTitle = "Escolha o diretório onde serão gravados os arquivos"
' Text to appear in the the gray area under the title bar
' telling you what to do

With tBrowseInfo
   '.hWndOwner = Me.hWnd ' Owner Form
   .lpszTitle = lstrcat(szTitle, "")
   .ulFlags = BIF_RETURNONLYFSDIRS + BIF_DONTGOBELOWDOMAIN
End With

lpIDList = SHBrowseForFolder(tBrowseInfo)

If (lpIDList) Then
   sBuffer = Space(MAX_PATH)
   SHGetPathFromIDList lpIDList, sBuffer
   sBuffer = Left(sBuffer, InStr(sBuffer, vbNullChar) - 1)
   BrowseFolder = sBuffer
```

```
End If
End Function

Public Sub OnActivate(RoseApp As RoseApplication)
    Dim thisAddIn As RoseAddIn
    Set thisAddIn = RoseApp.AddInManager.AddIns.GetFirst("PHP Modeler")
    If Not thisAddIn Is Nothing Then
        Dim tmpMenuItem As RoseContextMenuItem
        Set tmpMenuItem = thisAddIn.AddContextMenuItem(rsClass,
"Separator", "")
        Set GenerateCode = thisAddIn.AddContextMenuItem(rsClass, "Submenu
&Php Modeler", "")
        Set GenerateCode = thisAddIn.AddContextMenuItem(rsClass,
"&Configurar Diretório", "ConfigDir")
        Set GenerateCode = thisAddIn.AddContextMenuItem(rsClass, "&Gerar
Código", "GenerateCode")
        Set GenerateCode = thisAddIn.AddContextMenuItem(rsClass,
"Separator", "")
        Set CreateForm = thisAddIn.AddContextMenuItem(rsClass, "&Criar
Formulário", "CreateForm")
        Set CreateFormObj = thisAddIn.AddContextMenuItem(rsClass, "Submenu
&Objetos do Form", "")
        Set CreateFormObj1 = thisAddIn.AddContextMenuItem(rsClass,
"&Input", "Input")
        Set CreateFormObj2 = thisAddIn.AddContextMenuItem(rsClass, "&Text
Area", "Textarea")
        Set CreateFormObj = thisAddIn.AddContextMenuItem(rsClass,
"endsubmenu", "")
        Set GenerateCode = thisAddIn.AddContextMenuItem(rsClass,
"endsubmenu", "")
        Set tmpMenuItem = thisAddIn.AddContextMenuItem(rsClass,
"Separator", "")
        CreateFormObj1.MenuState = 0
        CreateFormObj2.MenuState = 0
        CreateForm.MenuState = 0
    End If
End Sub

Public Sub OnDeactivate(RoseApp As RoseApplication)
End Sub

Public Function OnSelectedContextMenuItem(RoseApp As RoseApplication,
internalName As String) As Boolean
  Select Case internalName
    Case "GenerateCode"
        RoseApp.ExecuteScript "c:/PHPModeler/scripts/Main.ebs"
    Case "CreateForm"
        Dim aClass As RoseClass
        Dim theNested As RoseClass
        Dim theAssociation As RoseAssociation

        Set aClass = RoseApp.CurrentModel.GetSelectedClasses.GetAt(1)
        Set theNested = aClass.AddNestedClass("Form")
        theNested.Stereotype = "aForm"
        Set theAssociation = theNested.AddAssociation("", aClass.Name)
        Set theRole1 = theAssociation.GetCorrespondingRole(aClass)
        Set theRole2 = theAssociation.GetOtherRole(aClass)
        theRole1.Aggregate = True
        theRole1.Navigable = False
        theRole2.Navigable = True
```

```
    Case "ConfigDir"
        ok = RoseApp.CurrentModel.CreateProperty("WAE", "Directory",
BrowseFolder, "String")
    Case "Input"
        RoseApp.ExecuteScript "c:/PHPModeler/scripts/Input.ebs"
    Case "Textarea"
        RoseApp.ExecuteScript "c:/PHPModeler/scripts/Textarea.ebs"
    Case Else
     MsgBox "Ocorreu um erro!"
  End Select
End Function
Public Function OnEnableContextMenuItems(RoseApp As RoseApplication,
MenuItem As RoseContextMenuItemType) As Boolean
    CreateForm.MenuState = 0
    CreateFormObj1.MenuState = 0
    CreateFormObj2.MenuState = 0
    If RoseApp.CurrentModel.GetSelectedClasses.Count = 1 Then
        Stereotype$ =
RoseApp.CurrentModel.GetSelectedClasses.GetAt(1).Stereotype
        If Stereotype = "aClient Page" Then
            CreateForm.MenuState = 1
        End If
        If Stereotype = "aForm" Then
            CreateFormObj1.MenuState = 1
            CreateFormObj2.MenuState = 1
        End If
    End If
End Function
Public Function OnNewModelElement(RoseApp As RoseApplication, objItem As
RoseItem)
    objType = objItem.IdentifyClass
    Dim Role1 As RoseRole
    Dim Role2 As RoseRole
    Dim theAssociation As RoseAssociation

    If objType = "Association" Then
        Set theAssociation = objItem
        Set Role1 = theAssociation.Role1
        Set Role2 = theAssociation.Role2
        s1 = Role1.Class.Stereotype 'destino
        s2 = Role2.Class.Stereotype 'origem
        If (s2 = "aClient Page") And ((s1 = "aClient Page") Or (s1 =
"aServer Page")) Then
            objItem.Stereotype = "aLink"
        End If
        If (s2 = "aServer Page") And (s1 = "aClient Page") Then
            objItem.Stereotype = "aBuild"
        End If
        If (s2 = "aForm") And (s1 = "aServer Page") Then
            objItem.Stereotype = "aSubmit"
        End If
    End If
End Function
```

QUADRO 15 – Código do servidor OLE

Anexo E – Arquivo de propriedades

```
# WAE Web Aplications Extension
# Begin version information
(object Petal
    version       45
    _written      "PHP Modeler"
    charSet       0)
# End version information


(list Attribute_Set


      # Begin set and model element definition
      (object Attribute
            tool "WAE"
            name "default__Class"

            #Defining Properties
            value       (list Attribute_Set
                        (object Attribute
                              tool  "WAE"
                              name  "File Name"
                              value       "")
                  ) #End properties
      ) # End set and model element list


      # Begin set and model element definition
      (object Attribute
            tool "WAE"
            name "aClientPage__Class"

            #Defining Properties
            value       (list Attribute_Set
                        (object Attribute
                              tool  "WAE"
                              name  "aLink"
                              value       "")
                        (object Attribute
                              tool  "WAE"
                              name  "Background"
                              value       "")
                        (object Attribute
                              tool  "WAE"
                              name  "BgColor"
                              value       "")
                        (object Attribute
                              tool  "WAE"
                              name  "BottomMargin"
                              value       "")
                        (object Attribute
                              tool  "WAE"
                              name  "Class"
                              value       "")
                        (object Attribute
                              tool  "WAE"
                              name  "ID"
                              value       "")
```

```
                    (object Attribute
                          tool  "WAE"
                          name  "LeftMargin"
                          value       "")
                    (object Attribute
                          tool  "WAE"
                          name  "MarginHeight"
                          value       "")
                    (object Attribute
                          tool  "WAE"
                          name  "MarginWidth"
                          value       "")
                    (object Attribute
                          tool  "WAE"
                          name  "RightMargin"
                          value       "")
                    (object Attribute
                          tool  "WAE"
                          name  "Tittle"
                          value       "")
                    (object Attribute
                          tool  "WAE"
                          name  "TopMargin"
                          value       "")
                    (object Attribute
                          tool  "WAE"
                          name  "Vlink"
                          value       "")
                    (object Attribute
                          tool  "WAE"
                          name  "OnLoad"
                          value       "")
                    (object Attribute
                          tool  "WAE"
                          name  "OnUnload"
                          value       "")

            ) #End properties
    ) # End set and model element list

    # Begin set and model element definition
    (object Attribute
          tool "WAE"
          name "aServer__Class"

          #Defining Properties
          value       (list Attribute_Set

                    (object Attribute
                          tool  "WAE"
                          name  "isClass"
                          value       TRUE)
            ) #End properties
    ) # End set and model element list

    # Begin set and model element definition
    (object Attribute
          tool "WAE"
          name "aForm__Class"
```

```
            #Defining Properties
            value       (list Attribute_Set

                        (object Attribute
                              tool "WAE"
                              name  "MethodType"
                              value  (list Attribute_Set
                                    (object Attribute
                                          tool "WAE"
                                          name  "Get"
                                          value  1)
                                          (object Attribute
                                          tool "WAE"
                                          name  "Post"
                                          value 2)
                                    )
                        )

                        (object Attribute
                              tool  "WAE"
                              name  "Method"
                              value       ("MethodType" 1)) # 1 é Get e 2 é
Post
                  ) #End properties
      ) # End set and model element list


      # Begin set and model element definition
      (object Attribute
            tool "WAE"
            name "aLink__Association"

            #Defining Properties
            value       (list Attribute_Set
                        (object Attribute
                              tool  "WAE"
                              name  "Target"
                              value       "")
                        (object Attribute
                              tool  "WAE"
                              name  "Href"
                              value       "")
                        (object Attribute
                              tool  "WAE"
                              name  "Style"
                              value       "")
                        (object Attribute
                              tool  "WAE"
                              name  "TabIndex"
                              value       "")
                        (object Attribute
                              tool  "WAE"
                              name  "OnBlur"
                              value       "")
                        (object Attribute
                              tool  "WAE"
                              name  "OnClick"
                              value       "")
                        (object Attribute
                              tool  "WAE"
```

```
                              name  "OnDbClick"
                              value       "")
                        (object Attribute
                              tool  "WAE"
                              name  "OnFocus"
                              value       "")
                        (object Attribute
                              tool  "WAE"
                              name  "OnKeydown"
                              value       "")
                        (object Attribute
                              tool  "WAE"
                              name  "OnKeypress"
                              value       "")
                        (object Attribute
                              tool  "WAE"
                              name  "OnKeyup"
                              value       "")
                        (object Attribute
                              tool  "WAE"
                              name  "OnMouseDown"
                              value       "")
                        (object Attribute
                              tool  "WAE"
                              name  "OnKeypress"
                              value       "")
                        (object Attribute
                              tool  "WAE"
                              name  "OnMouseMove"
                              value       "")
                        (object Attribute
                              tool  "WAE"
                              name  "OnMouseOut"
                              value       "")
                        (object Attribute
                              tool  "WAE"
                              name  "OnMouseOver"
                              value       "")
                        (object Attribute
                              tool  "WAE"
                              name  "OnMouseUp"
                              value       "")
                  ) #End properties
      ) # End set and model element list

)
```

QUADRO 16 – Código do arquivo de propriedades utilizado no protótipo