Filipe Del Nero Grillo
Profª. Drª. Renata Pontin de Mattos Fortes

AWMo: Accessible Web Modeler

Manual técnico e operacional

INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
UNIVERSIDADE DE SÃO PAULO
ICMC - USP

São Carlos - SP, Brasil

13 de Janeiro de 2014

Sumário

1 Introdução

1.1 Contexto e Motivação

Atualmente no Brasil, o número de pessoas que possuem algum tipo de deficiência é de 45,62 milhões. De acordo com o Censo 2010, este número representa cerca de 23,92% de toda a população do país [3].

Com o aumento de serviços de utilidade pública sendo oferecidos pelo governo por meio da Internet e outros meios eletrônicos, tendência chamada de *e-gov* (sigla para governo eletrônico), aumenta-se a preocupação com relação ao acesso desses cidadãos. Por esse motivo, existem algumas iniciativas que buscam melhorar a acessibilidade de tais serviços, como a portaria e-MAG (Modelo de Acessibilidade em Governo Eletrônico) [1] entre outras leis e decretos que abordam desde meios eletrônicos, como o e-MAG, quanto o acesso a ambientes físicos, com a obrigatoriedade da construção de rampas para pessoas que utilizem cadeiras de rodas.

Mais especificamente no âmbito da computação, os deficientes visuais têm historicamente uma participação mais ativa, pois os códigos de programas de computadores são essencialmente no formato texto e, portanto, mais facilmente acessíveis por meio de tecnologias assistivas, como leitores de tela e mostradores de braille, por exemplo.

No entanto, alguns tipos de dados ainda permanecem intrinsecamente dependentes de sentidos específicos, como a visão. Os modelos de *software* por exemplo, são geralmente representados na forma de diagramas visuais, usualmente compostos por formas geométricas e conectores ligando essas formas geométricas. Na maioria das vezes também existe conteúdo escrito nestes diagramas, mas o significado das notações visuais é tão grande que por meio da leitura do texto apenas, o entendimento do conteúdo fica bastante prejudicado. Além disso, com o crescimento da disciplina de engenharia de *software* e linguagens visuais como a UML (*Unified Modeling Language*) e consequente aumento de atividades de modelagem que utilizam essas linguagens visuais [4], o acesso por deficientes visuais é mais prejudicado, sendo necessário o auxílio de outra pessoas tanto para a leitura, quando para a edição desses modelos.

Neste contexto, foi desenvolvida a AWMo (*Accessible Web Modeler*), uma ferramenta *Web* desenhada para permitir que os usuários modelem diagramas de classe da UML em duas visões distintas: a *visão gráfica* onde desenvolvedores de *software* com visão podem desenhar os diagramas de maneira tradicional, arrastando os elementos e conectando-os visualmente. E a *visão textual*, onde desenvolvedores de *software*, com ou sem deficiência visual, podem acessar e desenhar os mesmos diagramas utilizando uma gramática textual desenvolvida especialmente para a AWMo.

1.2 Objetivos

O objetivo deste manual é auxiliar os usuários da AWMo nos processos de utilização da ferramenta e sua instalação em um servidor *Web*. Além disso, este manual traz detalhes técnicos sobre a AWMo e instruções necessárias para que outros desenvolvedores de *software* possam colaborar com seu desenvolvimento.

1.3 Organização deste manual

Neste capítulo foi apresentada a contextualização da AWMo, bem como os objetivos deste manual.

O restante deste trabalho é organizado da seguinte forma: o Capítulo 2 fornece instruções sobre como instalar a AWMo em um servidor *Web*. O Capítulo 3 contém um manual ensinando como utilizar a AWMo para criar diagramas de classe da UML, passando por todas as páginas e ações possíveis em cada uma delas. No Capítulo 4 encontra-se um manual detalhado sobre como utilizar a linguagem textual da AWMo e é por meio dessa linguagem que é possível criar diagramas de classe da UML na interface textual. Por fim, no Capítulo 5 são mostrados detalhes técnicos da arquitetura e organização da AWMo bem como instruções sobre como contribuir com o projeto.

2 *Manual de instalação*

2.1 Pré-requisitos do servidor

A AWMo pode ser executada tanto em máquinas Windows quanto Linux, no entanto este manual se concentra apenas em instruções para sua instalação em uma máquina Linux. Alguns detalhes como localização de diretórios podem ser diferentes de um sistema para o outro.

Para o funcionamento da AWMo, são necessários:

- Java 1.6
- Apache Tomcat 7.0.23
- MySQL 5 ou superior

2.2 Instalação

Para se instalar a AWMo, primeiro é necessário criar um banco de dados no MySQL que será utilizado para armazenar os modelos que serão criados pela AWMo. Supondo que o banco de dados se chame *awmo*:

Listagem 2.1: Comando para importar o schema inicial do banco de dados da AWMo.

```
mysql –Uusuario –p awmo < ./setup/schema.sql
```

Se quiser carregar um diagrama inicial para esta estrutura, para poder avaliar como ele é exibido na AWMo, pode-se importar o arquivo *seed.sql* do mesmo modo que o *schema.sql*.

Após realizar o *checkout* do código da AWMo, encontra-se uma estrutura de diretórios, conforme exibida na Figura 1.

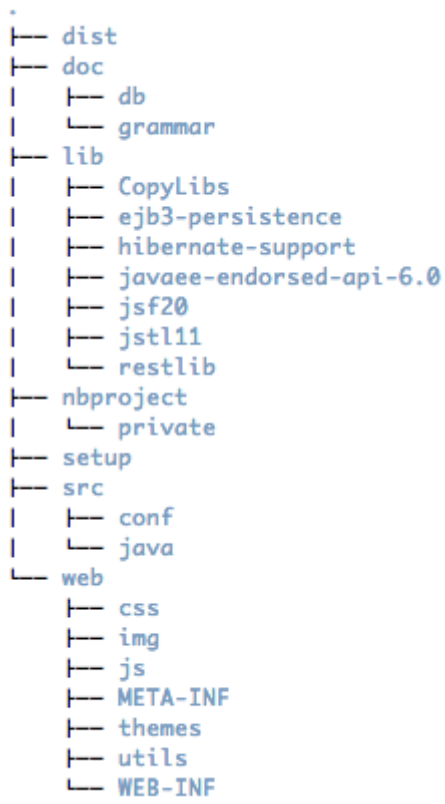

Figura 1: Estrutura de diretórios da base de código da AWMo.

O diretório *dist* possui uma versão de distribuição da AWMo em formato *war*. Para instalá-la no Apache Tomcat, basta copiar o arquivo *awmo.war* para dentro do diretório *webapps* da sua instalação do Tomcat.

Listagem 2.2: Comando para adicionar o arquivo .war da AWMo ao servidor Tomcat.

```
cp awmo.war /var/lib/tomcat7/webapps/
```

Após este passo, o Tomcat irá automaticamente extrair o conteúdo do arquivo e criará o diretório *awmo* dentro do diretório *webapps*. Dentro deste diretório encontram-se todos os arquivos necessários para a execução da AWMo e também alguns arquivos de configuração. Instruções de configuração são mostradas a seguir.

2.3 Configuração

Após realizados os passos de instalação mencionados na Seção anterior, a única configuração necessária para iniciar a operação da AWMo consiste na definição do banco de dados no arquivo de configuração do Hibernate.

Para isso basta alterar as seguintes diretivas no arquivo *hibernate.cfg.xml*:

Listagem 2.3: Configurações de acesso ao banco de dados para o Hibernate.

```
<property name="hibernate.connection.url">
    jdbc:mysql://localhost:3306/awmo?autoReconnect=true
</property>
<property name="hibernate.connection.username">awmo</property>
<property name="hibernate.connection.password">q1w2e3r4</property>
```

Na propriedade *hibernate.connection.url* deve-se substituir a url pela url de conexão do seu banco dados, caso ele não esteja instalado na mesma máquina ou na porta padrão.

Na propriedade *hibernate.connection.username* deve-se inserir o nome do usuário que possui acesso ao banco de dados criado para utilizar com a AWMo. Por questões de segurança não é recomendável utilizar o super usuário (*root)*.

Por fim, na propriedade *hibernate.connection.password*, deve-se inserir a senha do usuário definido anteriormente. Após realizadas essas configurações, já é possível acessar a AWMo para verificar se tudo foi configurado corretamente. Para isso, basta apontar seu navegador para a URL da AWMo no Tomcat, no caso de uma instalação na máquina local, este endereço é normalmente: *http://localhost:8080/awmo*, mas tanto a url quanto a porta podem variar de acordo com a sua instalação do Tomcat.

3 *Manual de uso*

Uma vez instalada e configurada, a AWMo pode ser acessada a partir de um navegador *Web*. A seguir, temos alguns dos requisitos necessários para sua utilização e instruções de como utilizá-la.

3.1 Pré-requisitos do cliente

Por se tratar de uma aplicação *Web*, o único requisito que se faz necessário para a utilização da AWMo é um navegador. Preferencialmente deve-se buscar utilizar versões recentes dos seu navegador de preferência. Alguns dos navegadores que foram testados:

- Mozilla Firefox[1]
- Google Chrome[2]
- Internet Explorer[3]

Outros navegadores modernos deverão funcionar, embora não tenham sido testados.

3.2 Navegação

A Navegação dentro da AWMo é realizada com o auxílio de dois componentes de interação principais: o menu à esquerda e a barra de contexto no topo da aplicação.

Quando não existe nenhum diagrama aberto para edição, a AWMo apresenta apenas o menu lateral esquerdo com as opções: Início e Ajuda. A primeira (Início) leva para a página principal da ferramenta, na qual podem ser encontrados os diagramas

[1] http://www.mozilla.org/firefox
[2] http://www.google.com/chrome
[3] http://windows.microsoft.com/en-us/internet-explorer/download-ie

existentes e novos diagramas podem ser criados. Essa página inicial é melhor detalhada na Seção 3.4. A segunda opção do menu leva para uma página de ajuda com conteúdo similar ao encontrado neste manual de uso.

Na Figura 2 podemos ver a aparência da AWMo quando não existe um diagrama aberto para edição. Nessa página podemos ver o menu lateral esquerdo em seu estado inicial.

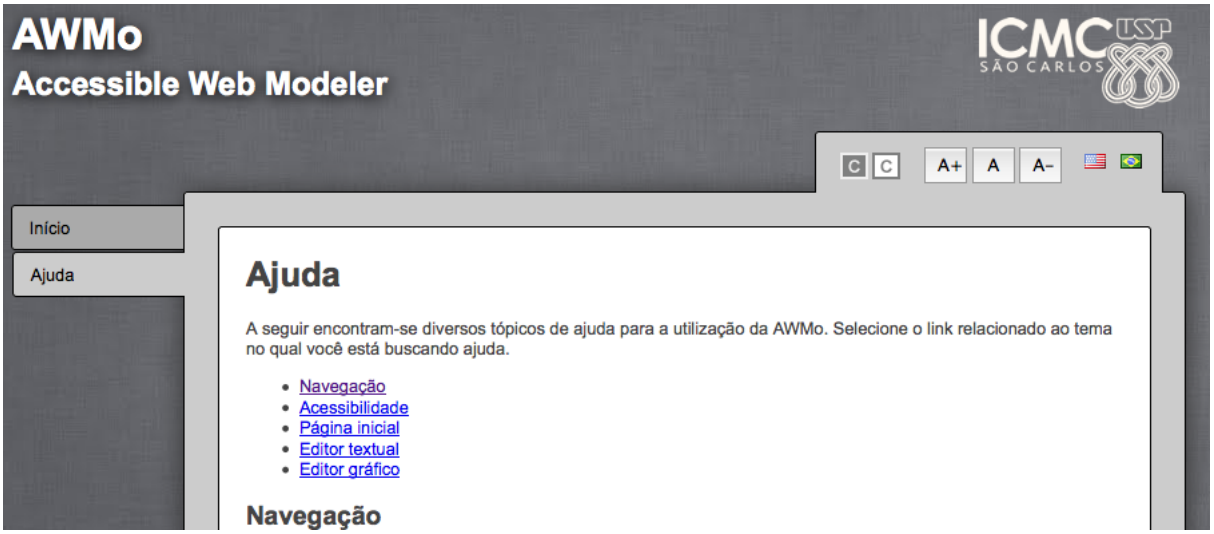

Figura 2: Captura de tela mostrando a aparência da AWMo quando não existe um diagrama sendo editado.

Quando um diagrama é aberto para edição, esta informação é mostrada em uma barra de contexto no topo da aplicação, ao lado da barra de ferramentas de acessibilidade. Na barra de contexto, é informado o nome do diagrama que se encontra aberto e existe um *link* que possibilita ao usuário fechar o diagrama atual.

Além disso, duas novas opções são exibidas no menu lateral esquerdo: "Visão Textual" que apresenta a interface do editor textual para o diagrama que se encontra atualmente aberto, e "Visão Gráfica" que apresenta a interface do editor gráfico para o diagrama que se encontra aberto. A Figura 3 mostra a aparência da AWMo com a barra de contexto no topo e novas opções no menu lateral esquerdo.

Nota: A visão gráfica da AWMo se encontra em uma versão preliminar e portanto ainda não pode ser utilizada completamente. Todo o conteúdo dos diagramas desenvolvidos com visão textual são exibidos na visão gráfica, porém a formatação e as ferramentas de edição e criação ainda precisam ser desenvolvidas.

Figura 3: Captura de tela mostrando a aparência da AWMo com um diagrama aberto para edição.

3.3 Acessibilidade

A AWMo possui algumas características de acessibilidade desenvolvidas internamente. A barra de ferramentas de acessibilidade mostrada na Figura 4 pode ser encontrada na parte superior direta da interface da AWMo.

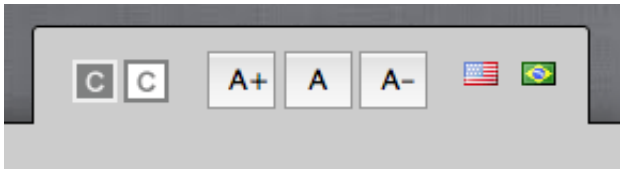

Figura 4: Captura de tela parcial mostrando a barra de ferramentas de acessibilidade da AWMo.

Da esquerda para a direta: Ativar alto contraste, Desativar alto contraste, Diminuir tamanho de fonte, Fonte em tamanho normal, Aumentar tamanho de fonte, Alterar idioma para Inglês e Alterar idioma para português.

As ferramentas descritas visam facilitar o uso da aplicação por pessoas com diferentes capacidades visuais e também usuários que sejam fluentes apenas em Inglês ou Português, atingindo assim uma gama maior de usuários.

3.4 Página inicial

A página inicial da AWMo tem duas funções: Possibilitar criação de novos diagramas na região com título: "Criar diagrama" e exibir a lista de todos os diagramas existentes na AWMo na região com título "Lista de diagramas".

Para criar um novo diagrama, deve-se preencher apenas o nome que se deseja para o diagrama a ser criado e pressionar o botão "Novo diagrama". Uma mensagem de sucesso será exibida no topo da página caso não ocorra nenhum erro durante a criação do diagrama e, após isso, o diagrama já poderá ser encontrado na lista de diagramas existentes na mesma página.

A lista de diagramas consiste em uma tabela que contém um diagrama por linha e para cada diagrama são exibidos: o nome, número de classes que o compõem e uma coluna com as ações possíveis de serem realizadas no diagrama da linha. As ações possíveis são:

- **Textual**: *Link* utilizado para abrir o diagrama para edição com o editor textual.
- **Gráfico**: *Link* utilizado para abrir o diagrama para edição com o editor gráfico.
- **Remover**: *Link* utilizado para remover um diagrama da AWMo. Antes de ser removido será solicitado que o usuário confirme a operação, pois esta é uma operação irreversível, ou seja, não é possível recuperar um diagrama que tenha sido removido.

É importante notar que mesmo existindo a opção de abrir os diagramas para edição com um determinado editor, após aberto pode-se alternar entre os editores textual e gráfico sem a necessidade de fechar o diagrama ou retornar à página inicial.

A Figura 5 ilustra o conteúdo da página inicial da AWMo e suas seções, onde podemos notar a existência de dois diagramas que contêm duas classes cada um deles.

3.5 Editor textual

O editor textual da AWMo é composto por uma área de texto que ocupa toda área visível e um único botão para salvar o estado atual.

A Figura 6 mostra um diagrama entitulado "Exemplo 2.8", aberto para edição no editor textual.

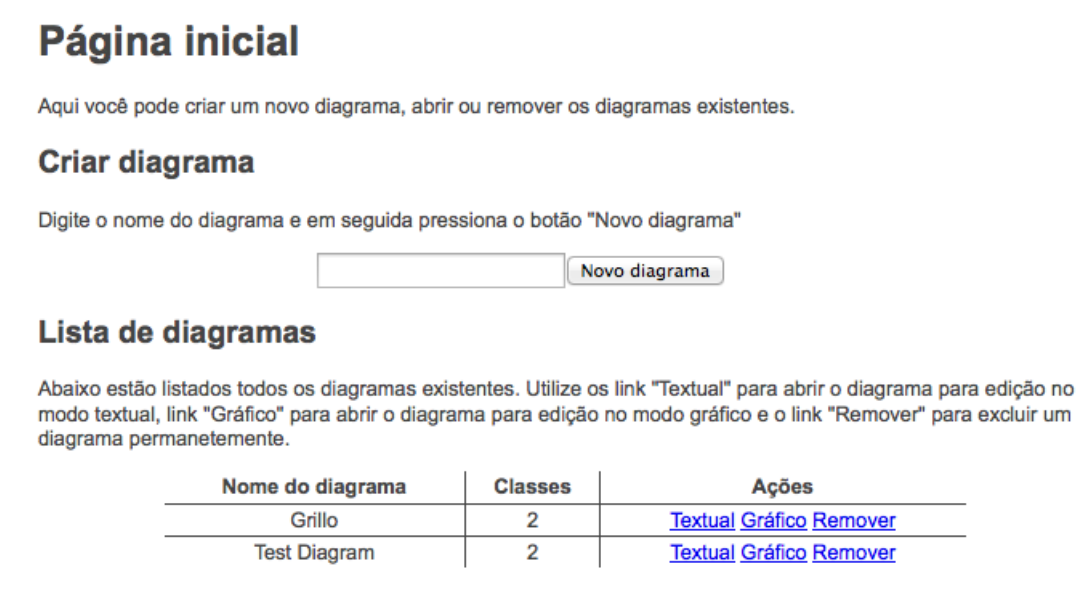

Figura 5: Captura de tela mostrando apenas o conteúdo da página inicial da AWMo.

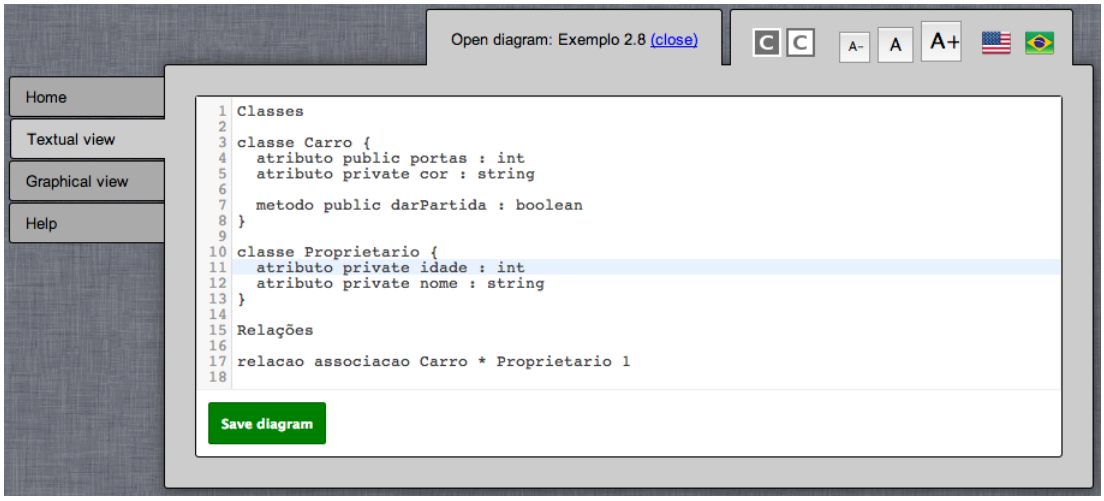

Figura 6: Captura de tela mostrando um diagrama aberto para edição na visão textual da AWMo.

Para se utilizar o editor textual é necessário conhecimento das estruturas da linguagem utilizada pela AWMo. A linguagem textual da AWMo é descrita em detalhes e com exemplos práticos na Seção 4.

3.6 Editor gráfico

O editor gráfico da AWMo encontra-se incompleto até a data em que este manual foi redigido. A Figura 7 mostra o mesmo diagrama "Exemplo 2.8" visto na Seção ante-

rior, porém aberto para edição na visão gráfica.

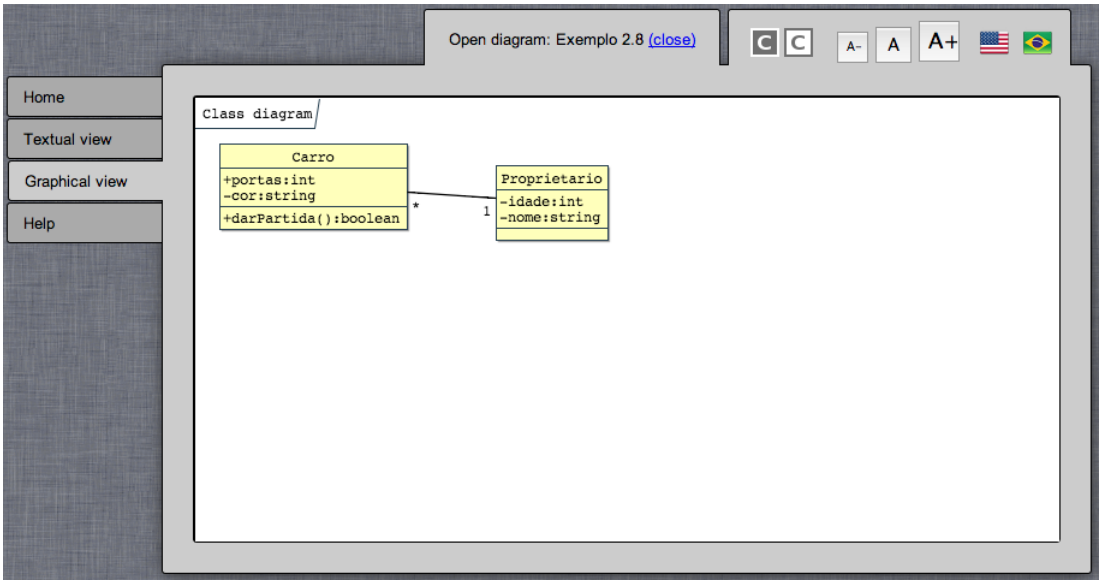

Figura 7: Captura de tela mostrando um diagrama aberto para edição na visão gráfica da AWMo.

A versão atual da AWMo é capaz de exibir todo o conteúdo criado por meio da interface textual e permite que o usuário manipule os elementos existentes com interações como: *drag & drop* para alterar a posição de uma classe no espaço do diagrama, adição e remoção de atributos e métodos. Porém, não é possível adicionar novas classes ou relacionamentos ao diagrama e, atualmente, não é possível salvar nenhuma das alterações realizadas no diagrama por meio da visão gráfica.

Esta restrição deve-se exclusivamente a uma restrição de tempo para a implementação do protótipo e necessidade de cronograma relacionados ao projeto de mestrado no qual a AWMo foi desenvolvida. Portanto, foi tomada a decisão de priorizar o desenvolvimento da interface textual para que fosse possível estudar o principal objetivo da ferramenta que é permitir o acesso aos deficientes visuais. No entanto, os estudos de viabilidade realizados mostraram que a implementação do retorno dos dados da interface gráfica para o banco de dados é possível e de fácil implementação.

A Seção 5 deste documento mostra mais detalhes técnicos de como a AWMo está estruturada e quais os passos necessários para que se possa colaborar com seu projeto, inclusive com a finalização da visão gráfica, por exemplo.

4 *Manual da linguagem textual*

A linguagem textual AWMo foi desenvolvida para permitir a modelagem de diagramas de classe da UML. Atualmente a linguagem se encontra em sua versão 0.4 e já é capaz de expressar as seguintes características da UML:

- Classes
- Atributos
- Métodos
- Herança entre classes
- Relação de associação
- Relação de agregação
- Relação de composição

Vale observar que para atributos e métodos, é possível definir sua visibilidade e tipos de dados, e para os métodos, é possível especificar uma lista de parâmetros e seus tipos de dados. Esta linguagem é utilizada na AWMo como uma forma de interação puramente textual, tanto para leitura, quanto para edição e criação de diagramas de classes. Seu principal objetivo é permitir que pessoas com dificuldade de visão sejam capazes de atuar e colaborar em projetos que envolvam modelagem de *software*.

Nas Seções 4.1 a 4.3 serão detalhadas as características da linguagem, mostrando como são definidas as classes, relações e todas as suas características. Todos os exemplos contidos neste manual são ilustrados com a visão gráfica do diagrama que é gerada pela AWMo a partir da versão textual.

4.1 Estrutura de um diagrama

Na AWMo, as informações contidas em um diagrama de classes são separadas em duas unidades: as Classes, principais elementos deste tipo de diagrama, e as Relações

entre estas classes, que juntos são capazes de modelar os conceitos de um sistema.

Na Listagem 4.1, podemos ver a palavra-chave **Classes** que é uma palavra obrigatória para delimitar o início das definições das classes do diagrama. Logo em seguida, são encontradas as classes em si, representadas nesta listagem apenas por reticências, pois serão vistas com detalhes na Seção 4.2.

Depois da definição de todas as classes do diagrama, encontra-se a palavra-chave **Relações** que é utilizada para indicar o início da declaração das relações entre as classes previamente declaradas. Na Listagem 4.1, as relações também são representadas por reticências e serão detalhadas na Seção 4.3.

Listagem 4.1: Exemplo da estrutura básica de um diagrama de classe na AWMo.

```
Classes
...

Relações
...
```

Na visão gráfica da AWMo, este exemplo mostra uma imagem vazia, por não possuir nenhuma classe.

4.2 Classes

Os principais elementos neste tipo de diagrama são as classes. Elas são utilizadas para modelar entidades de um determinado sistema e é possível mapear atributos e métodos (operações) que essas entidades possam possuir.

A Listagem 4.2 mostra a definição de uma classe simples na linguagem AWMo, sem atributos ou métodos:

Listagem 4.2: Exemplo do código de uma classe sem atributos ou métodos na AWMo.

```
Classes

classe Carro {
}

Relações
```

A Figura 8 ilustra a visão gráfica do mesmo diagrama do exemplo anterior.

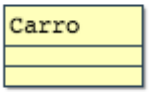

Figura 8: Exemplo de diagrama de classe gerado pela AWMo a partir do conteúdo da Listagem 4.2. Contendo apenas a classe Carro, sem métodos ou atributos

4.2.1 Atributos

Os atributos de classes da UML geralmente são utilizados para representar propriedades da entidade que está sendo representada pela classe. Por exemplo, todos os carros possuem uma cor e um número de portas, então a classe Carro pode ser representada contendo esses dois atributos.

Os atributos possuem duas propriedades: visibilidade e tipo. A visibilidade indica qual o nível de acesso que é permitido a esse atributo. A Tabela 1 mostra os tipos de visibilidade existentes na AWMo:

Tabela 1: Tipos de visibilidade existentes na AWMo

Visibilidade	**Descrição**
public	atributos e métodos com visibilidade pública, ou seja, podem ser acessados por todas as classes.
private	atributos e métodos com visibilidade privada apenas podem ser acessados pela classe que os possui. Nenhum acesso externo é permitido.
protected	com visibilidade protegida, os atributos e métodos podem ser apenas pela própria classe e por subclasses da mesma.

O tipo de um atributo indica qual o tipo de dado que este atributo irá armazenar. Se é uma *string* ou um valor inteiro, por exemplo. Os tipos podem ser tipos básicos, como os mencionados, ou tipos complexos. Os tipos complexos são outras classes existentes no próprio diagrama. A Tabela 2 mostra os tipos básicos de dados existentes na AWMo.

A Listagem 4.3 contém um exemplo da Classe carro citada com seus atributos e

Tabela 2: Tipos básicos de dados suportados pela AWMo

Visibilidade	**Descrição**
string	Uma *string* representa uma cadeia de caracteres de qualquer tamanho.
int	Utilizado quando o atributo ou parâmetro irá receber um valor numérico inteiro.
float	Outro tipo numérico, porém utilizado quando o elemento for receber valores reais.
boolean	Este tipo representa um valor booleano, ou seja, que pode assumir os valores 'verdadeiro' ou 'falso'.

respectivos tipo e visibilidade.

Listagem 4.3: Exemplo do código de uma classe com dois atributos de tipos simples.

```
Classes

classe Carro {
    atributo private Cor : string
    atributo public Portas : int
}

Relações
```

A Figura 9 ilustra a visão gráfica do mesmo diagrama do exemplo anterior.

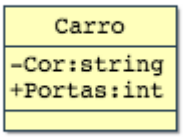

Figura 9: Exemplo de diagrama de classe gerado pela AWMo a partir do conteúdo da Listagem 4.3. Contendo a classe Carro e seus dois atributos

Além dos tipos simples, atributos também podem possuir tipos complexos de dados, ou seja, outras classes definidas no próprio diagrama. Na Listagem 4.4, o diagrama de classe contém uma classe Carro cujo tipo de dado do atributo motor é a própria classe Motor, também definida no mesmo diagrama.

Listagem 4.4: Exemplo de uso de atributos com tipos complexos.

```
Classes

classe Motor {
    atributo private fabricante : string
    atributo private potencia : int
}

classe Carro {
    atributo private cor : string
    atributo public portas : int
    atributo private motor : Motor
}

Relações
```

A Figura 10 ilustra a visão gráfica do mesmo diagrama do exemplo anterior.

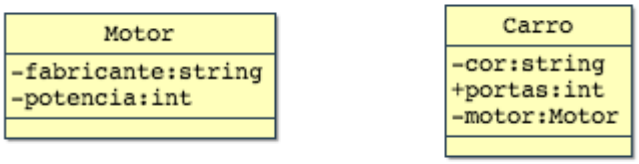

Figura 10: Exemplo de diagrama de classe gerado pela AWMo a partir do conteúdo da Listagem 4.4. Contendo as classes Motor e Carro, sendo que a classe Carro possui um atributo cujo tipo de dado é a classe Motor

4.2.2 Métodos

Os métodos ou operações representam ações que uma classe pode realizar. No exemplo do carro, dar a partida poderia representar um método que colocaria o motor do veículo em funcionamento. Os métodos também possuem as propriedades de visibilidade e tipo da mesma forma que os atributos, no entanto, o tipo de um método representa qual o tipo de dado do valor que será retornado quando o método for invocado. A Listagem 4.5 mostra um exemplo de modelo em que a classe Carro possui um método sem parâmetros.

Listagem 4.5: Exemplo de um método sem parâmetros.

```
Classes

classe Carro {
    atributo private cor : string
    atributo public portas : int

    metodo public darPartida() : boolean
}

Relações
```

A Figura 11 ilustra a visão gráfica do mesmo diagrama do exemplo anterior.

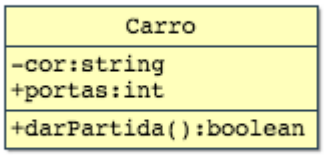

Figura 11: Exemplo de diagrama de classe gerado pela AWMo a partir do conteúdo da Listagem 4.5. Contendo a classe Carro, seus atributos e o método 'darPartida' que não possui nenhum parâmetro.

Além de visibilidade e tipo, os métodos podem ou não possuir uma lista de parâmetros. A Listagem 4.6 mostra a classe Carro com o método AbrirVidro que possui um parâmetro para indicar de qual porta se deseja abrir o vidro. Note que após o nome do parâmetro deve-se especificar seu tipo de dado, da mesma forma que para atributos. Para uso de mais parâmetros, basta separá-los por vírgula.

Listagem 4.6: Exemplo de um método com parâmetros.

```
Classes

classe Carro {
    atributo private cor : string
    atributo public portas : int

    metodo public abrirVidro(porta : int) : boolean
}

Relações
```

A Figura 12 ilustra a visão gráfica do mesmo diagrama do exemplo anterior.

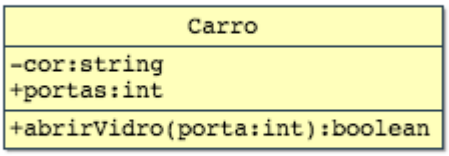

Figura 12: Exemplo de diagrama de classe gerado pela AWMo a partir do conteúdo da Listagem 4.6. Contendo a classe Carro e um método 'abrirVidro' que recebe como parâmetro um número inteiro que identifica qual porta deverá ter o vidro aberto.

4.2.3 Herança

A herança é um tipo especial de relacionamento entre classes. Este relacionamento implica que as classes filhas (que herdam a classe pai) possuem todas as propriedades da classe pai, mas podem possuir algum comportamento específico que não se aplique à classe pai. Assim, a classe que herda possui automaticamente todos os atributos e métodos da classe que é herdada.

Na Listagem 4.7, temos o exemplo de uma classe CarroCorrida que herda a classe Carro, pois carros de corrida possuem todas as características de um carro comum, porém ainda possuem algumas outras especificidades. Note que muito embora isto não esteja escrito explicitamente no conteúdo da classe, a classe CarroCorrida também possui todos os atributos cor e portas e o método darPartida() existentes na classe Carro da qual ela herda suas propriedades.

Listagem 4.7: Exemplo de uso de herança entre duas classes.

```
Classes

classe Carro {
    atributo private cor : string
    atributo public portas : int

    metodo public darPartida() : boolean
}

classe CarroCorrida herda Carro {
    atributo public numero : int
    atributo public patrocinador : string
}

Relações
```

A Figura 13 ilustra a visão gráfica do mesmo diagrama do exemplo anterior.

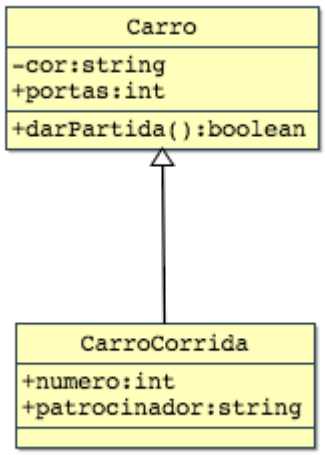

Figura 13: Exemplo de diagrama de classe gerado pela AWMo a partir do conteúdo da Listagem 4.7. Contendo as classes Carro e CarroCorrida sendo que CarroCorrida herda a classe Carro e possui alguns atributos específicos apenas para um carro de corrida.

4.3 Relacionamentos

Além da declaração de classes, seus atributos e métodos, durante a modelagem de sistemas de *software*, geralmente também é preciso mostrar como essas classes se

relacionam entre si. Para isso, a AWMo suporta, além da herança vista na Seção 4.2.3, as relações de associação, agregação e composição, detalhadas nas seções a seguir.

4.3.1 Associação

A associação é o tipo mais simples de relacionamento (relação) entre classes, ela representa literalmente que um conjunto de objetos pode estar relacionado a um objeto de uma outra classe, por exemplo, na Listagem 4.8 a relação de associação indica que carros têm um proprietário e proprietários possuem carros.

Além disso, existe a representação da cardinalidade dessa relação. A cardinalidade na AWMo está localizada logo após o nome de classe e pode assumir valores inteiros ou * (asterisco) que representa zero ou mais.

Novamente no caso da listagem 4.8, as cardinalidades indicam que proprietários podem possuir qualquer número de carros. No entanto, cada carro pode possuir apenas um proprietário.

Na notação visual da UML a associação é representada como uma linha conectando as duas classes.

Listagem 4.8: Exemplo de uso relação de associação entre duas classes.

```
Classes

classe Carro {
  atributo public portas : int
  atributo private cor : string

  metodo public darPartida : boolean
}

classe Proprietario {
  atributo private idade : int
  atributo private nome : string
}

Relações

relacao associacao Carro * Proprietario 1
```

A Figura 14 ilustra a visão gráfica do mesmo diagrama do exemplo anterior.

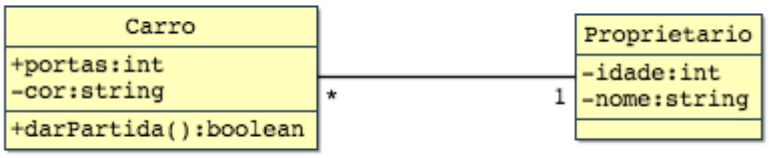

Figura 14: Exemplo de diagrama de classe gerado pela AWMo a partir do conteúdo da Listagem 4.8. Contendo as classes Carro e Proprietário com relacionamento de associação entre eles sendo que um proprietário pode ter diversos carros, mas um carro pode ter apenas um proprietário.

4.3.2 Agregação

A agregação é um tipo de relacionamento utilizado para indicar que uma das classes representa um conjunto de outra. No entanto, na agregação as classes ainda podem existir mesmo sem o relacionamento. No exemplo da Listagem 4.9, temos as classes Carro e Detran, o banco do Detran é composto por diversos veículos, mas o Detran existe independente da existência de veículos. O mesmo é válido para carros, eles são registrados no Detran, porém podem existir independentemente de estarem cadastrados.

Uma classe é um agrupamento de outras, porém essas classes não dependem dessa relação para existirem, diferente da Composição que será detalhada na Seção 4.3.3.

Na notação visual da UML, a agregação é representada como um losango vazio do lado da classe que "manda" no relacionamento.

A cardinalidade também é utilizada na agregação, da mesma forma que na associação.

Listagem 4.9: Exemplo de uso relação de agregação entre duas classes.

```
Classes

classe Carro {
  atributo public portas : int
  atributo private cor : string

  metodo public darPartida : boolean
}

classe Detran {
  atributo private telefone : string
  atributo private endereco : string
}

Relações

relacao agregacao Carro * Detran 1
```

A Figura 15 ilustra a visão gráfica do mesmo diagrama do exemplo anterior.

Figura 15: Exemplo de diagrama de classe gerado pela AWMo a partir do conteúdo da Listagem 4.9. Contendo as classes Carro e Detran que possuem um relacionamento de agregação entre elas.

4.3.3 Composição

A composição é similar à agregação, porém, ela representa a contenção. Dessa forma, os objetos que estão contidos dentro do outro objeto dependem dele para existir e, portanto, são instanciados e destruídos de acordo com o objeto que os contém. Na Listagem 4.10, temos as classes Pedido e ItemPedido, em um modelo de pedidos de compra, o item do pedido não tem razão para existir se não for dentro de um pedido. No entanto, ele é um dos principais componentes de um pedido.

Na notação visual da UML, a composição é representada como um losango preenchido do lado da classe que "manda" no relacionamento.

A cardinalidade também é utilizada na composição, da mesma forma que na associação.

Listagem 4.10: Exemplo de uso relação de composição entre duas classes.

```
Classes

classe Pedido {
  atributo public data : string
  atributo private cliente : int

  metodo public adicionarItem(codigo : int, nome : string) : boolean
}

classe ItemPedido {
    atributo private codigo : int
    atributo private nome : string
}

Relações

relacao composicao Pedido 1 ItemPedido *
```

A Figura 16 ilustra a visão gráfica do mesmo diagrama do exemplo anterior.

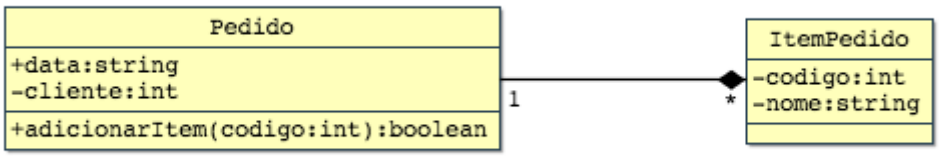

Figura 16: Exemplo de diagrama de classe gerado pela AWMo a partir do conteúdo da Listagem 4.10. Contendo as classes Pedido e ItemPedido que possuem um relacionamento de composição onde um pedido é composto por itens de pedido.

5 *Manual técnico*

As seções a seguir mostram detalhes técnicos da AWMo com o objetivo de facilitar que futuros desenvolvedores possam colaborar com seu desenvolvimento.

5.1 Tecnologias empregadas

A AWMo é composta por duas partes principais:

- Aplicação *Web*
- Linguagem textual

Em termos práticos, toda a infraestrutura da linguagem textual da AWMo está contida dentro de uma biblioteca chamada textualDiagram-0.4.jar. Mais detalhes sobre a biblioteca e seu desenvolvimento serão descritos na Seção 5.3.

A Figura 17 mostra a estrutura do projeto exibida pelo NetBeans[1]. Dentro do NetBeans, os arquivos do projeto ficam organizados em cinco categorias:

***Web* Pages:** ficam aqui os arquivos referentes as páginas *web* da aplicação, desde os arquivos xhtml contendo código html e *tags* do JSF até arquivos de folhas de estilo (css), JavaScript e as imagens que são exibidas nas páginas.

Source Packages: contém todos os arquivos de código fonte java da aplicação, arquivos de mapeamento e de configuração do hibernate.

Libraries: contém todas as bibliotecas (.jar) utilizadas no projeto: TextualDiagram-0.4.jar que é a biblioteca da linguagem textual desenvolvida especialmente para a AWMo que será detalhada na Seção 5.3 e suas dependências (EMF e Xtext); bibliotecas do Java Server Faces e Hibernate, blibioteca para conexão com banco de dados MySQL e uma biblioteca para fazer *parsing* de JSON chamada Gson.

[1] https://netbeans.org/

Configuration Files: é um atalho criado pelo NetBeans para mostrar os arquivos de configuração separados, facilitando assim que sejam encontrados. Aqui são encontrados os arquivos *MANIFEST.MF*, *context.xml*, *faces-config.xml* e *web.xml*.

Server Resources: encontram-se aqui os arquivos com instruções SQL para criar a estrutura inicial do banco de dados da AWMo e populá-la com dados iniciais: *schema.sql* e *seed.sql*, respectivamente.

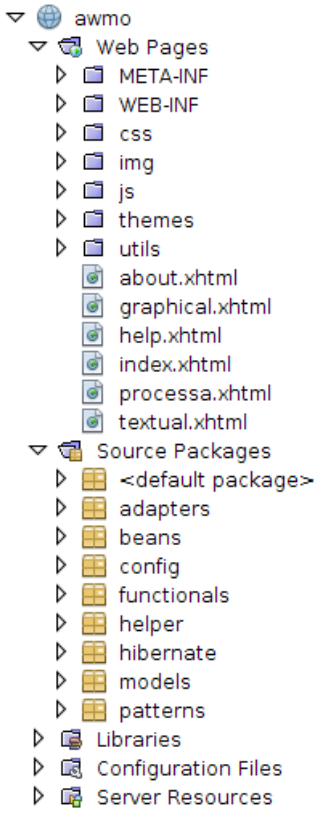

Figura 17: Estrutura do projeto como vista na IDE NetBeans.

5.2 Aplicação *Web*

A AWMo utiliza o Java Server Faces (JSF) 2.1 e para seu desenvolvimento, utilizou uma arquitetura de referência que garantiu que o projeto fosse iniciado com uma base sólida e seguindo critérios de acessibilidade [2].

Dentre as páginas *Web* que podem ser vistas no diretório *Web* Pages da Figura 17 temos:

about.xhtml: possui um texto citando os autores e colaboradores da AWMo e também o reconhecimento pelo apoio financeiro recebido pelo CNPq.

graphical.xhtml: esta página é onde o editor gráfico é exibido para o usuário da AWMo quando o mesmo abre um diagrama para edição em modo gráfico. Este editor encontra-se atualmente incompleto.

help.xhtml: página de ajuda com instruções para auxiliar os usuários a opera a AWMo.

index.xhtml: Está é a página inicial da AWMo, ela contém um formulário que permite ao usuário criar novos diagramas e abaixo exibe a lista de todos os diagramas existentes na ferramenta. Para cada um dos diagramas da lista existem ações para abri-los com a visão textual ou gráfica e ainda uma ação que permite excluí-los.

processa.xhtml: esta página não possui nenhum conteúdo exibido pelo usuário, ela é utilizada apenas como parte do processo de salvar os diagramas textuais e depois redireciona o usuário novamente para o o editor textual (textual.xhtml) com o resultado do processamento.

textual.xhtml: esta página contém o editor textual da AWMo. É exibida uma grande área de texto e um botão onde o usuário pode salvar o estado atual do diagrama. Além disso, quando esta página é mostrada após a operação de salvar um diagrama são listadas mensagens ao usuário na parte superior da página informando se a operação foi realizada com sucesso ou se houve algum erro.

Além disso, no diretório *utils*, existem outros arquivos de páginas que contêm porções que se repetem em todas ou muitas páginas, por exemplo, podemos encontrar o menu lateral, a barra de funções no topo, o cabeçalho, rodapé e o *template* que os agregam.

5.3 Linguagem textual

A linguagem textual representa uma peça muito importante na abordagem proposta pela AWMo, pois é por meio dessa linguagem que os engenheiros de *software* com deficiência visual são capazes de interagir com os modelos de *software*.

Em sua versão atual, a AWMo possibilita apenas a edição de diagramas de classe da UML e, portanto, a única linguagem textual desenvolvida até o momento serve especificamente para esse fim.

Na Listagem 5.1 temos a definição completa da gramática que define a linguagem da AWMo. Esta gramática livre de contexto está representada em notação EBNF (*Extended Backus-Naur Form*).

Nas linhas 1 e 3 temos instruções para o Xtext, sendo que a definição da gramática em si é iniciada na linha 5. A partir desta definição o Xtext é capaz de gerar toda a infraestrutura da linguagem textual, as quais precisam ser extraídas em uma biblioteca isolada para que possa ser incluída na ferramenta *web* da AWMo. A seguir, encontram-se instruções de como gerar a infraestrutura da linguagem com o Xtext e como extraí-la do projeto do Eclipse.

Listagem 5.1: Gramática em notação EBNF que define a linguagem da AWMo.

```
1  grammar br.org.awmo.textual.TextDiagram with org.eclipse.xtext.common.↩
       Terminals
2
3  generate textDiagram "http://www.org.br/awmo/textual/TextDiagram"
4
5  ClassDiagram:
6   {ClassDiagram}
7
8      'Classes'
9   classes += ClassElement*
10
11  'Relações'
12  relations += Relation*
13 ;
14
15 ClassElement:
16  'classe' name = ID ('herda' parent = [ClassElement])? '{'
17
18   attributes += Attribute*
19
20   methods += Method*
21
22  '}'
23 ;
24
25 Attribute:
26  'atributo' visibility=Visibility name = ID ':' type=ComplexType
27 ;
28
29 Method:
30  'metodo' visibility=Visibility name = ID ('(' (parameters+=Parameter*)? ↩
       ')')? ':' type=ComplexType
31 ;
32
33 Parameter:
34  name = ID ':' type=ComplexType (',' otherParamenter=Parameter)?
35 ;
```

```
36
37 Relation:
38  'relacao' relationType=RelationType source=[ClassElement] ↩
       sourceCardinality=CARDINALITY target=[ClassElement] targetCardinality↩
       =CARDINALITY
39 ;
40
41 ComplexType:
42  type=Type | aggregation=[ClassElement]
43 ;
44
45 terminal CARDINALITY: '*' | 'n' | '1';
46
47 enum RelationType:
48  ASSOCIATION = 'associacao' |
49  AGGREGATION = 'agregacao' |
50  COMPOSITION = 'composicao'
51 ;
52
53 enum Visibility:
54  PUBLIC = 'public' |
55  PRIVATE = 'private' |
56  PROTECTED = 'protected'
57 ;
58
59 enum Type:
60  STRING = 'string' |
61  INT = 'int' |
62  FLOAT = 'float' |
63  BOOLEAN = 'boolean'
64 ;
```

Para o desenvolvimento da linguagem, foi utilizado o Eclipse Modeling Tools, uma versão da IDE Eclipse que já é distribuída com os pacotes e ferramentas de modelagem da *Eclipse Foundation*. A Figura 18 mostra uma captura de tela com os detalhes da versão utilizada neste projeto. Recomenda-se fortemente que se utilize as versões mostradas nesta figura para se editar a gramática mostrada na Listagem 5.1.

Para editar a gramática, deve-se criar um novo projeto do Xtext no Eclipse, como mostra a Figura 19.

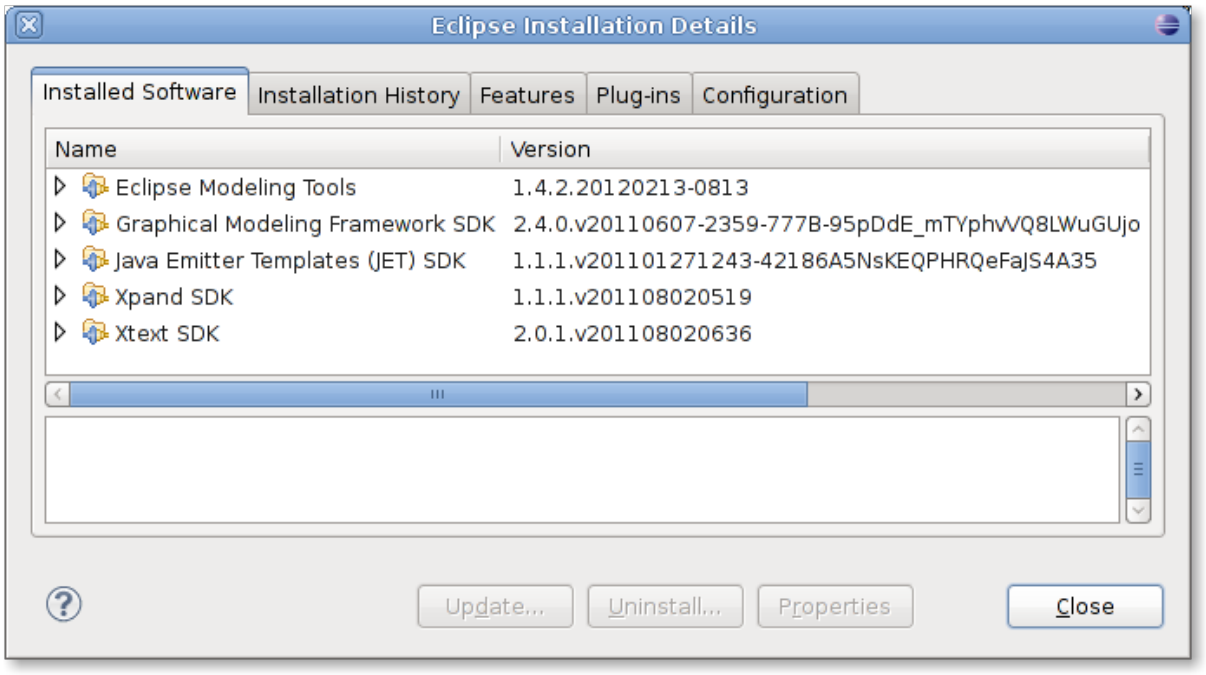

Figura 18: Captura de tela dos detalhes da instalação da IDE Eclipse utilizara para o desenvolvimento da linguagem textual da AWMo.

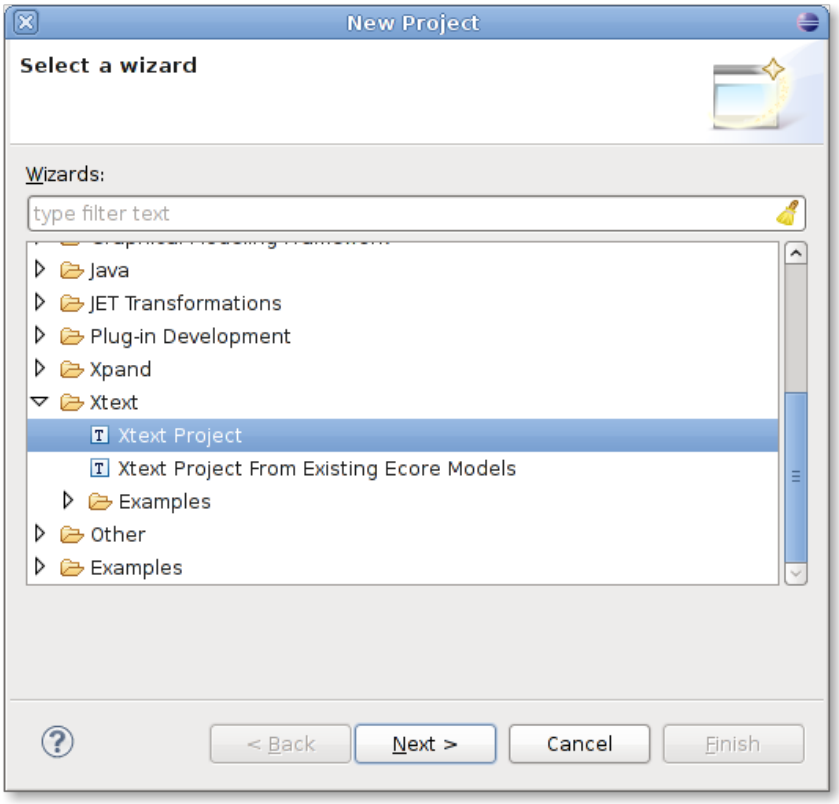

Figura 19: Captura de tela mostrando o tipo de projeto Xtext que deve ser utilizado para se trabalhar com a linguagem textual.

A partir dessas definições, serão criados três projetos no *workspace* do Eclipse, como mostra a Figura 20. O projeto terminado com o nome na linguagem (neste caso TextualDiagram) é onde a gramática será definida e sua infraestrutura será gerada pelo Xtext. Para os propósitos da AWMo os outros dois projetos, terminados em ".tests" e ".ui", não são utilizados.

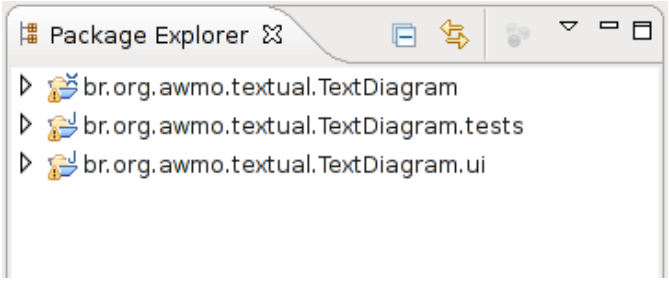

Figura 20: Exemplo dos três projetos criados quando se cria um novo projeto de tipo Xtext.

Uma vez que a gramática foi adicionada ao arquivo com extensão ".xtext" do projeto, pode-se executar o Xtext para dar início a geração da linguagem textual em si. Para tanto, deve-se executar o *Workflow* MWE2, mostrado em destaque na Figura 21. Para executá-lo, basta clicar com o botão direito, escolher a opção "Executar como", e em seguida selecionar "MWE2 Workflow".

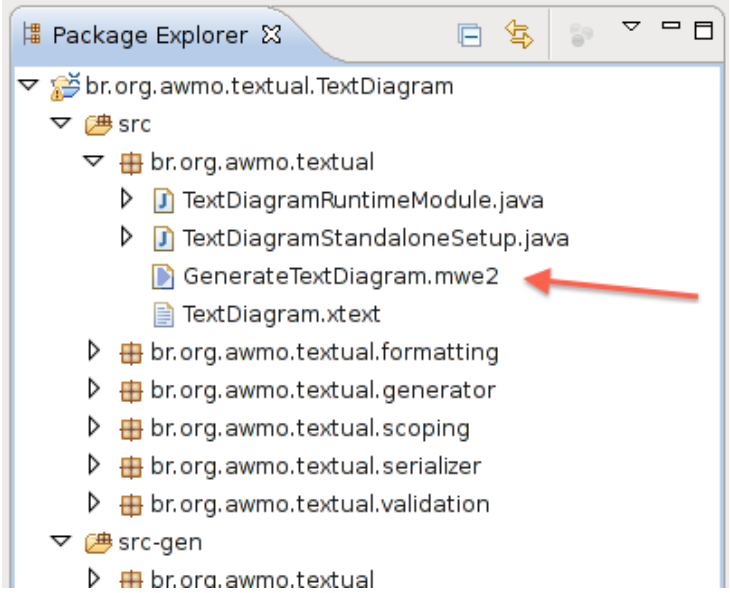

Figura 21: Arquivos do projeto da linguagem textual, mostrando em destaque o script de *workflow* responsável pela geração da linguagem textual a partir da gramática definida no arquivo TextualDiagram.xtext.

Após a execução, toda a infraestrutura da linguagem textual da AWMo terá sido gerada. Os pacotes gerados podem ser vistos na Figura 22.

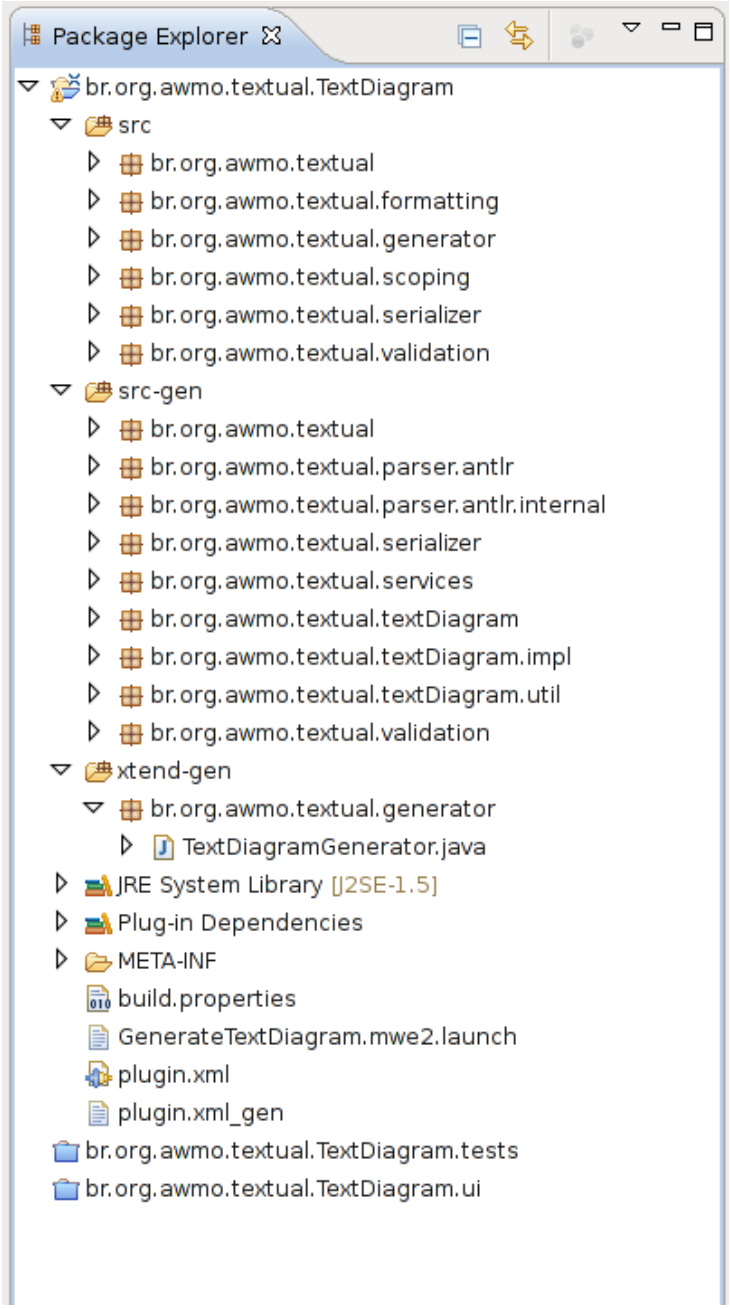

Figura 22: Pacotes existentes no projeto Xtext após a execução do *workflow* mwe2.

Uma vez que se tem a infraestrutura gerada, deve-se exportar os pacotes como uma biblioteca JAR. Para isso é preciso criar um novo projeto no Eclipse do tipo "Java Project". Em seguida, copiar todos os pacotes do projeto do Xtext para o novo projeto criado. Isso inclui todos os pacotes dos diretórios "src", "src-gen" e "xtend-gen" para o diretório "src" do novo projeto Java, como mostra a Figura 23.

Além dos pacotes do projeto Xtext, é necessário adicionar ao novo projeto algumas das bibliotecas utilizadas pelo projeto Xtext e que são dependências dos códigos gerados. A Figura 23 mostra a lista de todas as bibliotecas que são necessárias para o projeto que será exportado como uma biblioteca JAR da linguagem textual.

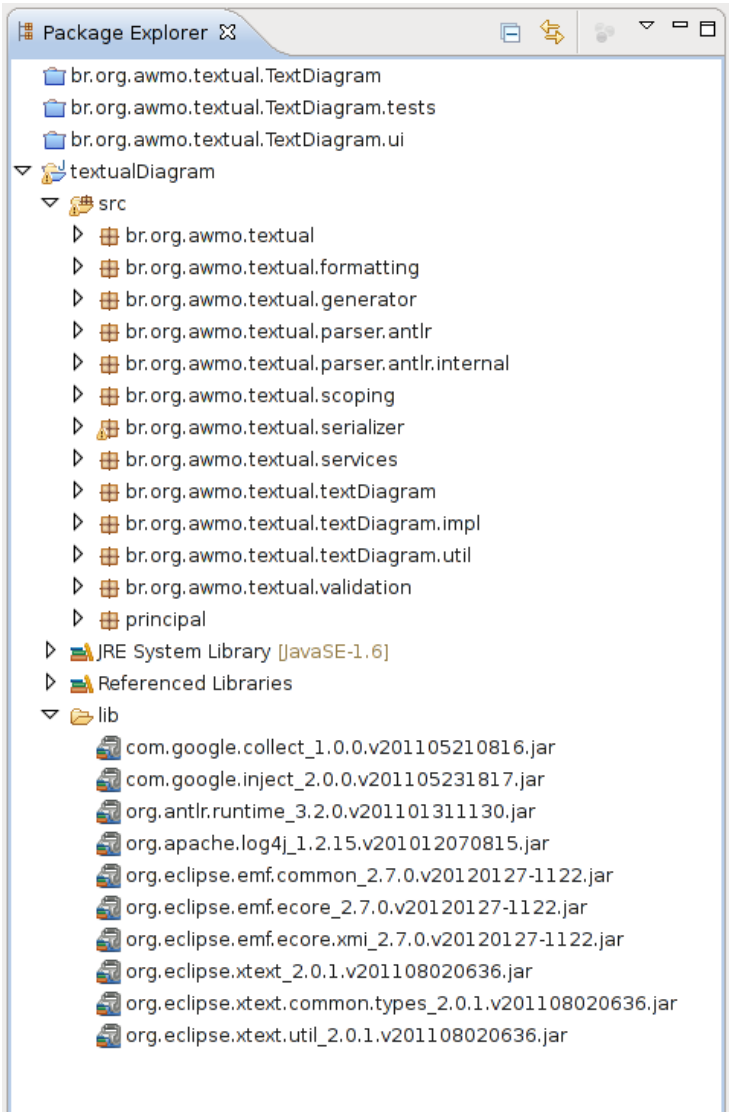

Figura 23: Exemplo do projeto Java criado para exportar a infraestrutura da linguagem textual como um arquivo JAR.

Neste ponto, o Eclipse não deve estar acusando nenhum erro de dependência e a biblioteca da linguagem textual pode ser exportada. Para isso, deve-se clicar com o botão direito do *mouse* no projeto, neste caso chamado "textualDiagram", e selecionar a opção "Exportar". Uma nova janela, como a mostrada na Figura 24, irá surgir para se escolher qual o tipo de exportação desejada. Deve-se selecionar a opção "Arquivo JAR". Com isso uma nova versão da linguagem textual foi gerada e o arquivo JAR está pronto para ser inserido no projeto *Web* da AWMo, citado nas seções anteriores.

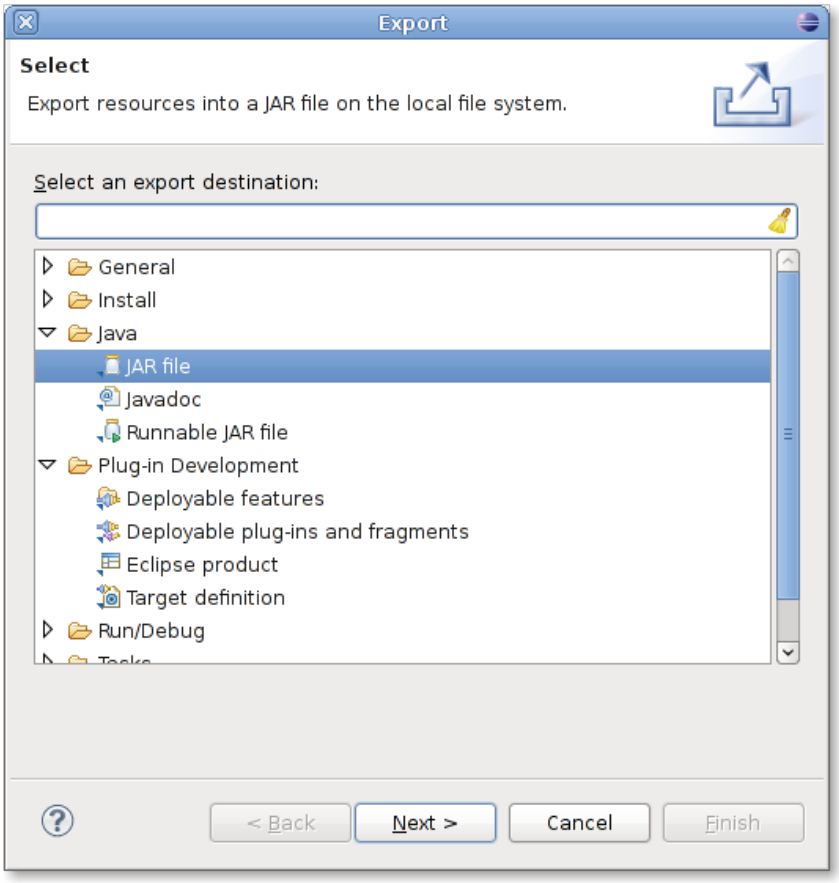

Figura 24: Caixa de seleção do tipo de exportação a se utilizar para criar a biblioteca JAR da linguagem textual da AWMo.

É importante lembrar que sempre que forem realizadas alterações na gramática da linguagem textual, estas alterações também deve ser refletidas no código da aplicação *Web*, em especial os tradutores encontrados no pacote 'functionals', os serializadores encontrados no pacote 'adapters', e o mapeamento do hibernaste para o banco de dados, se necessário. Dependendo das mudanças realizadas na gramática textual, pode haver grande impacto em todo código da aplicação *Web*, que atualmente foi construída a partir da versão 0.4 da linguagem textual, que é distribuída junto com o projeto e tem sua gramática mostrada na Listagem 5.1.

5.4 Licença

A AWMo utiliza a versão simplificada da licença BSD para permitir que seja distribuído e modificado como um *software* livre sob as condições impostas pela licença.

A seguir, temos uma cópia da licença que também pode ser encontrada na raiz do projeto da AWMo e que deve ser mantida em caso de alterações em seu código e re-distribuições.

Listagem 5.2: Licença BSD simplificada que é utilizada pela AWMo.

```
1  Copyright (c) 2013, Filipe Del Nero Grillo
2  All rights reserved.
3
4  Redistribution and use in source and binary forms, with or without
5  modification, are permitted provided that the following conditions
6  are met:
7      * Redistributions of source code must retain the above
8        copyright notice, this list of conditions and the following
9        disclaimer.
10     * Redistributions in binary form must reproduce the above
11       copyright notice, this list of conditions and the following
12       disclaimer in the documentation and/or other materials
13       provided with the distribution.
14     * Neither the name of the University of São Paulo nor the
15       names of its contributors may be used to endorse or promote
16       products derived from this software without specific prior
17       written permission.
18
19 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND ↩
       CONTRIBUTORS
20 "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
21 LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
22 FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL Filipe
23 Del Nero Grillo BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
24 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
25 LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
26 USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED ↩
       AND
27 ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
28 OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
29 OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
30 SUCH DAMAGE.
```

5.5 Como contribuir

O projeto da AWMo está hospedado no GitHub em:

Listagem 5.3: URL do repositório da AWMo no GitHub.

```
1 https://github.com/awmo/awmo
```

Para contribuir com o projeto da AWMo, basta fazer um *fork* ou *download* do código do projeto. Alguns dos pontos que pode-se colaborar com a AWMo, no momento da escrita deste manual, são:

- Adicionar uma nova visão com uma representação em HTML para os modelos, utilizando *links* para melhorar a navegação do usuário no conteúdo do modelo durante sua leitura e permitindo que a leitura possa ser realizada de maneira não linear.
- Adicionar outras linguagens (outros tipos de modelos e diagramas, UML ou não)
- Traduzir e melhorar as mensagens de erros geradas pelo Xtext e adicionar a coluna onde o erro foi encontrado.

Além disso, caso haja qualquer dúvida sobre como contribuir com a AWMo, entre entre em contato seus autores:

Filipe Del Nero Grillo filipe.grillo at gmail.com

Renata Pontin de Mattos Fortes renata at icmc.usp.br

Referências

[1] Portaria Nº 3, de 7 de maio de 2007. Institucionaliza o Modelo de Acessibilidade em Governo Eletrônico - e-MAG no âmbito do Sistema de Administração dos Recursos de Informação e Informática - SISP. *Diário Oficial [da] República Federativa do Brasil*, page 103, 5 2007. n. 87, Seção 1, ISSN 1677-7042.

[2] Renata P. M. Fortes, Ana L. Dias, Filipe D. N. Grillo, and Paulo C. Masiero. Creating a project towards universal access: is it possible? In *Proceedings of INTERACT 2013 Workshop on Rethinking Universal Accessibility: A broader approach considering the digital gap*, Cape Town (South Africa), 2013. INTERACT 2013 Workshop on Rethinking Universal Accessibility: A broader approach considering the digital gap.

[3] IBGE. *Censo Demográfico 2010 - Resultados Preliminares da Amostra*. Instituto Brasileiro de Geografia e Estatística, Rio de Janeiro, 2010.

[4] Luiz A. Ricci and Daniel Schwabe. An authoring environment for model-driven web applications. In *Proceedings of the 12th Brazilian Symposium on Multimedia and the web*, WebMedia '06, pages 11–19, New York, NY, USA, 2006. ACM.