**OverView da Abordagem**

O uso da abordagem pode seguir a seguinte ordem:

1. Especificação dos tipos de variáveis;

2. Especificação das classes (GPL);

2. Definição de funções pelo desenvolvedor (GPL);

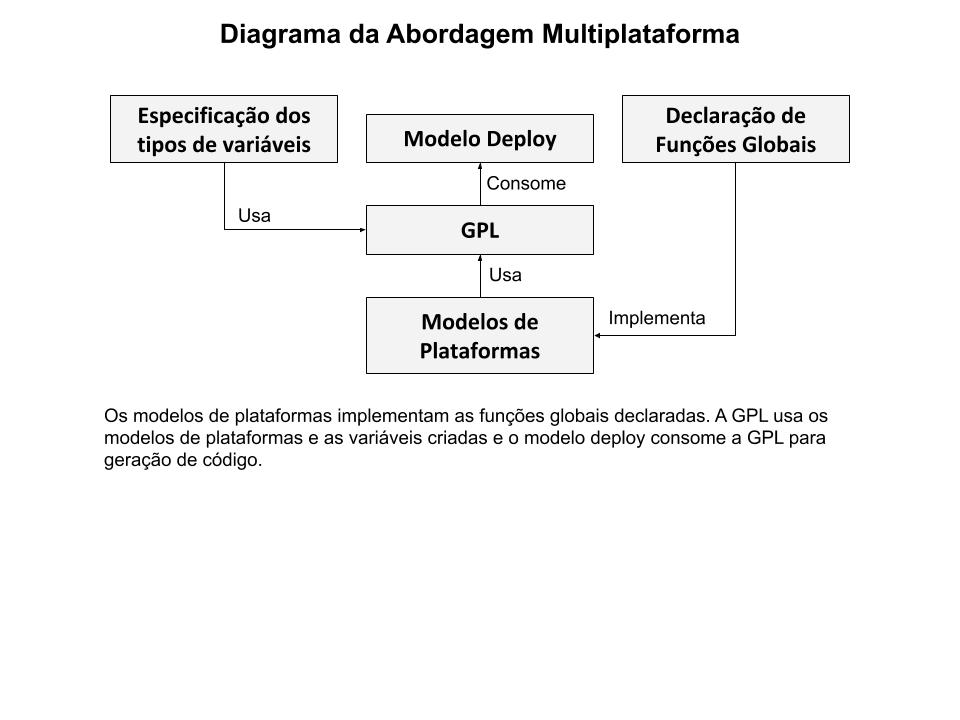
3. Declaração das funções globais;

4. Especificação dos modelos de plataforma;

5. Configuração da geração de código (Deploy);

6. Recursos extras; e

7. Gerador de código e Código gerado.



**1. Especificação dos tipos de variáveis.**  
Esta especificação pode ser feita em um arquivo dedicado e importado em todos os novos projetos. A Listagem a seguir apresenta a declaração de alguns tipos utilizando a GPL (General-Purpose Language) proposta.

|  |
| --- |
| datatype int datatype string datatype float datatype double datatype bool datatype dateTime datatype void datatype list<E> //E é um parâmetro genérico que permite a criação de qualquer tipo de lista na programação GPL |

**Listagem 1**

**2. Especificação das classes e definição de funções pelo desenvolvedor (GPL).**Cada classe tem atributos e operações, especificadas em uma linguagem própria, independente de plataforma. O trecho da GPL apresentado na Listagem a seguir apresenta a especificação da classe “ShoppingCartDAO”, onde é definida uma função “addProduct” (linhas 7-18) usada para inserir um produto em um carrinho de compras.

|  |
| --- |
| class ShoppingCartDAO {   usesGlobal SelectCustomer(id: string): Customer  userGlobal SelectProduct(id: string): Product  usesGlobal InsertProductIntoCart(customer: Customer, product: Product, quantity: int): string  operation addProduct(custId: string, prodId: string, quantity: int): list<string> {  objCust: Customer  objProd: Product  newProd: string  accessData: list<string>   objCust := global SelectCustomer(custId)  objProd := global SelectProduct(prodId)  newProd := global InsertProductIntoCart(objCust, objProd, quantity)  }  } |

**Listagem 2**

Como pode ser notado, essa listagem não apresenta detalhes dependentes de plataforma. No entanto, em determinados momentos (linhas 10, 11 e 12) é feita uma consulta em um banco de dados persistente para encontrar os clientes e produtos cadastrados, e adicionar o produto no carrinho. Na abordagem proposta, esse tipo de funcionalidade precisa ser implementada em uma plataforma específica. Para evitar que essa dependência de plataforma prejudique a reutilização das classes, a abordagem prevê declarações de funções globais.

**3. Declaração das funções globais.**

Funções globais (palavra-chave global) podem ser definidas como algoritmos que contém códigos específicos de plataformas e podem ser adaptadas em tempo de geração de código para qualquer classe do sistema. A Listagem a seguir apresenta a declaração de três funções globais, utilizadas na Listagem 2: SelectCustomer, SelectProduct e InsertProductIntoCart. A declaração consiste em um protótipo de função, que contém todos os parâmetros e tipo de retorno necessários para utilização.

|  |
| --- |
| global SelectCustomer(id: string): Customer global SelectProduct(id: string): Product global InsertProductIntoCart(customer: Customer, product: Product, quantity: int): string |

**Listagem 3**

As funções globais podem ser utilizadas em qualquer classe sem que esta necessite conhecer sua implementação, como por exemplo acontece na Listagem 2. Para isso, basta que seja definido que aquela classe utiliza uma função global, por meio da palavra-chave "usesGlobal" (linhas 2-4 da Listagem 2).

**4. Especificação dos modelos de plataforma.**

Cada função global terá uma implementação diferente em cada plataforma. Cada plataforma é declarada, por meio de um nome e uma linguagem. A Listagem a seguir mostra quatro plataformas: Web / C\#, Android / Java, iOS / Swift e Augmented Reality, uma plataforma inteiramente nova desenvolvida para este projeto, em C\#. Essas plataformas contém três implementações diferentes para a função InsertProductIntoCart: baseada em SQL, para a plataforma Web/C\# (linhas 2-5), baseada em Hashmap, para a plataforma Android/Java (linhas 8-11) e baseada em SQLite, para a plataforma iOS/Swift (linhas 14-17). A plataforma AugmentedReality não possui suporte para essa função, portanto não apresenta sua implementação, e sim de outra função global. Nota-se que essas implementações podem ser feitas diretamente na linguagem da plataforma, possibilitando assim controle total sobre a implementação.

|  |
| --- |
| platform Web: CSharp {   implementsGlobal InsertProductIntoCart(customer: Customer, product: Product, quantity: int): string {  //C# code that implements the function on the web platform,  //using SQL for persistence  } }  platform Android: Java {  implementsGlobal InsertProductIntoCart(customer: Customer, product: Product, quantity: int): string {  // Java code that implements the function on the Android platform,  // using HashMap for memory persistence  } }  platform iOS: Swift{   implementsGlobal InsertProductIntoCart(customer: Customer, product: Product, quantity: int): string {  // Swift code that implements the function on the iOS platform,  // using SQLite for device persistence  } }  platform AugmentedReality: CSharp {  implementsGlobal GetDiskImagesAsync(picturesFolder: StorageFolder): list<img> {  //C# code that implements the function on   //the platform for Augmented Reality  } } |

**Listagem 4**

**5. Configuração da geração de código (Deploy).**

Uma vez definidos os tipos de dados, classes e plataformas, o próximo passo consiste em definir um modelo de implantação (deploy), onde cada classe pode ser implantada em uma plataforma específica. Essa definição servirá de guia para a geração de código. O trecho da GPL mostrado na Listagem 5 apresenta um exemplo de implantação de diferentes classes em diferentes plataformas. A única restrição é que as plataformas escolhidas para implantação possuam uma implementação para todas as funções globais utilizadas pelas classes nela implantadas. Neste exemplo, a classe ShoppingCartDAO está sendo implantada nas plataformas Android e iOS, mas não na plataforma Web, o que significa que será gerado código apenas para essas plataformas.

|  |
| --- |
| deploy Ecommerce{  User: Android  User: iOS  User: Web   UserDAO: Android  UserDAO: iOS  UserDAO: Web  Connection: Android  Connection: iOS  Connection: Web  ShoppingCartDAO: AndroidDataBase  ShoppingCartDAO: AndroidHashMap  } |

**Listagem 5**

A estrutura da abordagem permite a modelagem dos conceitos de um sistema de forma genérica e facilita a troca da configuração de um sistema multiplataforma. A troca de configuração pode ser desde a escolha das plataformas que irão rodar certas classes até a troca da forma como determinadas classes operam.

**6. Recursos extras da abordagem.**

O modelo-base descrito até aqui funciona para muitas situações diferentes. No entanto, existem casos onde a declaração de funções globais e sua implementação por meio de código fixo não é suficiente. Para isso, foram incorporadas alguns recursos adicionais: o uso de tipos genéricos nas funções globais, a possibilidade de definição de classes globais, para aumentar a capacidade expressiva da linguagem nesse nível, e o uso do Velocity para possibilitar que as implementações dos corpos de funções possam utilizar recursos de geração de código também, flexibilizando assim seu uso e aumentando sua capacidade de reutilização.

Como exemplo desses recursos adicionais, apresenta-se na Listagem 6 a função global “InsertObject”. Esta função tem o objetivo de gravar em um banco de dados qualquer objeto. Na sua declaração, aparece o tipo do parâmetro E, como um tipo genérico (Linha 2). Na implementação, o tipo genérico E é consultado, em tempo de geração de código, utilizando diretrizes do Velocity para realizar a substituição (Linhas 4, 6 e 11). No uso, basta que seja declarado o tipo real a ser utilizado no código gerado, neste caso, foi definida a classe "Order" (Linha 22). A abordagem se encarrega de realizar o vínculo entre essas definições e gerar código correto.

|  |
| --- |
| // declaracao global <E> InsertObject(e: <E>): string // implementacao implementsGlobal <E> InsertObject(e: <E>): string { try{  #foreach($f in $E.attributes)  #if($f.name != 'id')  cmd.Parameters.AddWithValue("@${f.name}", ${f.name});  #end   #end  SQL = "insert into $E.name (#foreach($f in $E.attributes)#if($f.name != 'id')${f.name}#if($foreach.hasNext) , #end#end#end) values (#foreach($f in $E.attributes)#if($f.name != 'id')@${f.name}#if($foreach.hasNext) , #end#end#end)";  cmd.Connection = Conn;  cmd.CommandText = SQL;  cmd.ExecuteNonQuery();  result="Data inserted into database successfully!";  objCon.CloseConnection();   }catch (MySql.Data.MySqlClient.MySqlException ex){  result = "Error " + ex.Number + " has occurred: " + ex.Message;  } } // uso usesGlobal <Order> InsertObject(e: Order): string |

**Listagem 6**

Outro recurso usado para a geração de código multiplataforma são as funções “globalDetails”. Estas são semelhantes às funções globais. As funções “globalDetails” também são operações globais especificadas nos modelos de plataformas e são responsáveis em adicionar detalhes específicos de plataformas nas classes onde elas forem usadas. Essas funções são recomendadas quando uma determinada operação requer detalhes específicos de plataformas e são complexas e/ou possuem muitas diferenças entre as plataformas para se programar de forma unificada. Por exemplo, as funções “globalDetails” podem ser usadas em operações de conexão, operações para uso de algum recurso específico dos dispositivos (ex: GPS e acelerômetro), operações de controle de fluxo de dados (camada Controller da arquitetura MVC), entre outras. Assim como a função global, a “globalDetails” também precisa da declaração, implementação e uso para sua utilização.

A Listagem 7 apresenta um exemplo de “globalDetails” para a função DBConnection. Na parte da implementação (linhas 4-18), nota-se que dentro da função existem outros métodos (linhas 7-13) com detalhes específicos de conexão. Esses detalhes e métodos se diferem muito entre as plataformas, tornando o “globalDetails” ideal para esses casos. Na linha 21 da Listagem \ref{lst:globalDetails} é apresentado a chamada da função dentro da classe “Connection”, nesta chamada é possível ver a classe “DBConf” sendo passada como parâmetro genérico. Essa classe (linhas 23-30) contém dados sobre a conexão de forma genérica e no exemplo da Listagem é utilizado para as plataformas web, Android e iOS.

|  |
| --- |
| // Declaracao:  globalDetails <E> DBConnection   // Implementacao:  implementsGlobalDetails <E> DBConnection{  '''  private MySqlConnection Conn;  public MySqlConnection OpenConnection(){  #set($server=$E.attributes.get(3).name + "." + $E.attributes.get(4).name + "." + $E.attributes.get(5).name)  Conn = new MySqlConnection("server=$server;database=$E.attributes.get(0).name;"  + "uid=$E.attributes.get(1).name;pwd=$E.attributes.get(2).name");  Conn.Open();  return Conn;   }  public void CloseConnection(){  Conn.Close();  }   '''   }  // Uso:  class Connection{  usesGlobalDetails <DBConf> DBConnection  }  class DBConf {  ecommerce: string //name of database  db\_rest\_user: string //user  restCommerce2019: string //password  MYSQL5011: string //Server  Smarterasp: string //Server  net: string //Server  } |

**Listagem 7**

**7. Gerador de código e Código gerado.**

Uma vez modelado o sistema com a abordagem basta salvar o documento e os códigos já serão gerados na pasta src-gen com o nome das respectivas classes.