3
Um Modelo de Operações para a *web* semântica

3.1.
Modelo de Operações

As classes do Modelo de Operações representam a definição de como deve ser uma operação em uma aplicação, ou seja, quais os valores devem ser fornecidos para sua execução e quais valores devem ser retornados como resultado dessa execução. Estas classes expressam como deve ser a especificação de um serviço acionado por algum usuário ou sistema, com o objetivo de afetar um comportamento de um determinado objeto da aplicação.

No Modelo de Operações, temos primitivas adicionais de modelagem para representar operações. Neste novo modelo, o esquema conceitual é constituído sobre parâmetros de entrada, parâmetros de saída, pré-condições, pós-condições, propriedades de definição do tipo e linguagem de programação, além da definição do código da operação. Na figura 10 podemos ver a representação deste modelo:

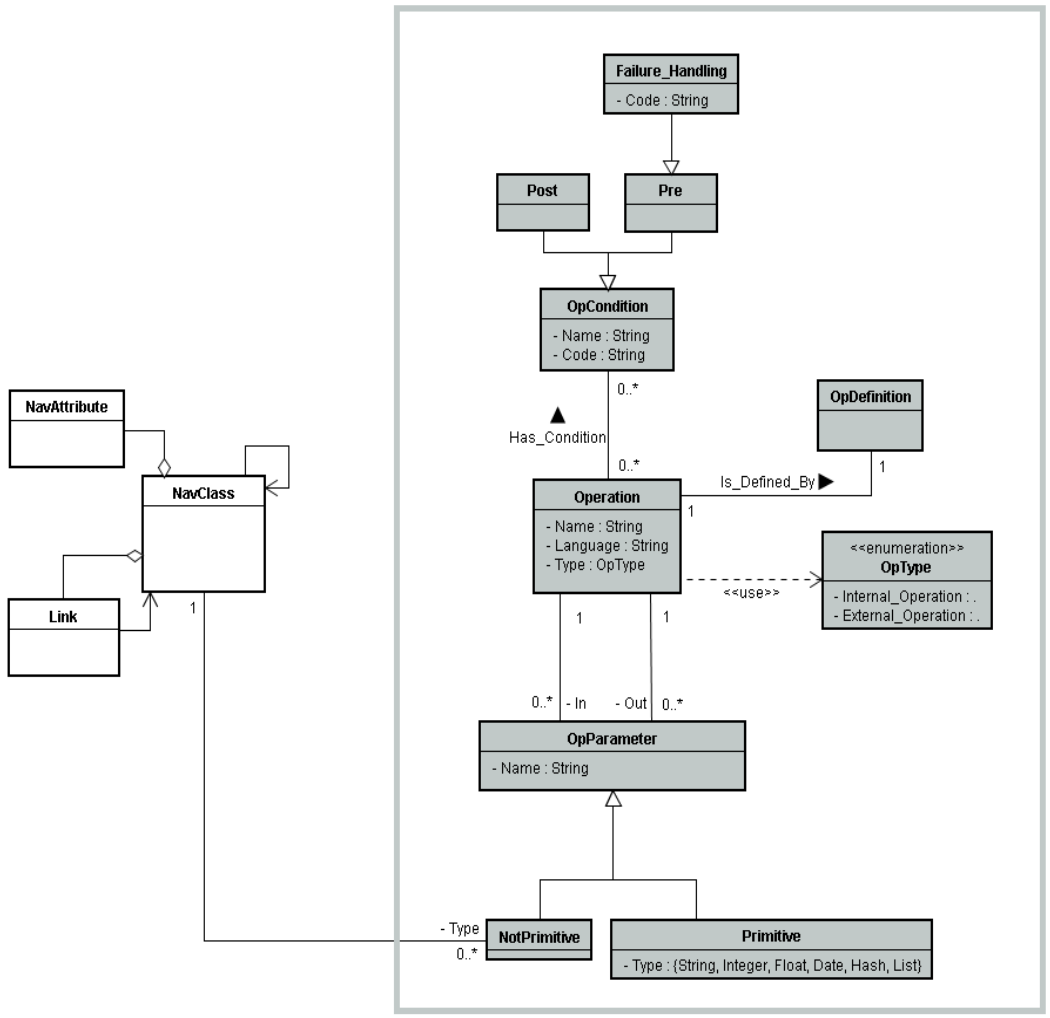

Figura 10 – Modelo de Operações

As classes que representam o Modelo de Operações estão descritas dentro do retângulo na figura. Já as classes fora do retângulo representam algumas classes do Modelo Navegacional do método SHDM: NavClass, para classes navegacionais, NavAttribute, para atributos navegacionais e Link para elos. As classes navegacionais se relacionam com o Modelo de Operações através da metaclasse NavClass.

O Modelo de Operações é um modelo geral para especificação da lógica de negócio de aplicações. Foi descrito em termos do SHDM, pois as classes navegacionais do metamodelo deste método foram utilizadas como entrada para os parâmetros. Para usar este modelo em outros métodos de *design* de aplicações hipermídia é necessário substituir a metaclasse NavClass (classe navegacional) por um conceito equivalente no método selecionado. No Capítulo 5 será detalhado como o Modelo de Operações interage com os outros modelos de um método de desenvolvimento de aplicações hipermídia.

Nas subseções seguintes serão apresentados os componentes do Modelo de Operações.

3.1.1. Operation

Uma operação é especificada como uma instância de Operation. Três atributos são definidos no momento da criação de uma nova operação:

- Name: define o nome da operação e deve ser iniciado por letra minúscula;
- Language: define a linguagem de programação que será utilizada pelo usuário para descrever o código da operação;
- Type: define o tipo da operação, podendo ser:
    - Operação interna (*Internal_Operation*): não é visível às entidades externas à aplicação, portanto, nunca pode ser invocada por estas. Pode ser invocada por outras operações ou pelo Modelo de Interface Concreta;
    - Operação externa (*External_Operation*): funciona como um canal de comunicação entre as entidades externas e a aplicação. Pode ser invocada por meio de requisições http (*Hypertext Transfer Protocol*) a uma URL (*Uniform Resource Locator*) padronizada e produz como resultado a renderização de uma interface visual ou o redirecionamento à outra URL.

As operações são representadas no Modelo de Operações como pequenas caixas com estilo visual semelhante ao Cartão de Contexto do esquema navegacional do método SHDM, como mostra a figura 11. A especificação completa da operação é ilustrada na figura 12, por meio do *template* do Cartão de Operação.

Contexto:	
Parâmetros:	
Elementos:	
Classes em Contexto:	
Ordenação:	
Navegação Interna:	
Operações	
Usuário:	**Permissão:**
Comentários:	

Figura 11 – Cartão de Contexto

Operation:	
Language:	**Type:**
Precondition:	**Failure_Handling:**
Parameter In:	
Parameter Out:	
Postcondition:	

Figura 12 – Cartão de Operação

Os elementos de uma Operation são representados da seguinte forma:

Notação:

- Name: *new_author*
- Language: *Ruby*
- Visibility: *internal_operation*

Exemplo:

Operation: new_author	
Language: Ruby	**Type:** internal_operation
Precondition:	**Failure_Handling:**
Parameter In:	
Parameter Out:	
Postcondition:	

Figura 13 – Exemplo de Operation

Outros elementos compõem uma operação e também devem ser especificados. Estes elementos são representados por suas classes e relações com Operation.

3.1.2. OpCondition

OpCondition define as condições que devem ser válidas antes e após a execução da operação. Possui dois atributos:

- Name: define o nome da condição;
- Code: trecho de código em alguma linguagem de programação definida anteriormente no atributo “Language” da classe Operation.

Esta diretiva pode ser de dois tipos:

- Pre: representa a pré-condição, ou seja, condição que deve ser verificada antes da execução da operação. A operação só será executada se essa condição for verdadeira;
- Post: representa a pós-condição, ou seja, condição que deve ser válida após a execução da operação. A pós-condição estabelece o que muda na estrutura da informação manipulada pela operação.

A classe “Pre”, que representa a especificação da pré-condição, possui uma especialização, Failure_Handling (tratamento de falha). O tratamento de falha lida com os possíveis acontecimentos fora do comportamento normal ou desejado para uma determinada pré-condição. Cada pré-condição pode possuir ou não um tratamento de falha. Failure_Handling possui um atributo:

- Code: trecho de código que contém o tratamento da ocorrência de condições fora do fluxo normal de execução da pré-condição.

Neste modelo não foi definido exceção compensatória ou tratamento de falha para lidar com os fluxos anormais oriundos da pós-condição. Isto fará parte de outro trabalho que descreverá transações em aplicações *web*.

Operation possui um relacionamento com OpCodition, denominado Has_Condition, no qual zero ou muitas operações podem possuir zero ou muitas condições e, vice-versa.

Para descrever pré e pós-condição é utilizada a linguagem de primeira ordem [Barwise e Etchemendy, 1999], na qual as fórmulas são definidas a partir de predicados. Um predicado é definido a partir de um símbolo predicativo, que descreve relações e/ou qualidades dos indivíduos do discurso [Favero, 2006]. Além de poder ser definido para indivíduos arbitrários com o uso de variáveis.

Temos como exemplo “ama(joao, maria)”, que pode ser lido como “João ama Maria”.

A motivação para o uso de lógica de primeira é o fato de possuir um formalismo bem estudado e possibilitar que qualquer condição sobre o estado de um objeto possa ser descrito por uma fórmula lógica de primeira ordem.

A lógica de primeira ordem foi utilizada para efeitos de documentação. Caso se deseje verificar a validade das expressões de pré e pós-condições, deve-se substituir os predicados descritos em linguagem de primeira ordem por trecho de código em alguma linguagem de programação.

Notação:

- Pré-condição: ¬ ∃ author(Author(author)), ou seja, quando o cadastro está sendo realizado, a aplicação verifica por meio da pré-condição se o autor que está sendo cadastrado não existe, caso não exista, ele é aceito, evitando assim autores duplicados
- Tratamento de falha da pré-condição: ∃ author(Author(author)) execute Print_Inclusion_Error(), ou seja, caso o autor que está sendo cadastrado já exista, é executado o método Print_Inclusion_Error()
- Pós-condições: new(author, Author) e send(email, Chair), ou seja, um autor for criado e um e-mail foi enviado ao Chair para notificá-lo da criação do autor

Exemplo:

Operation: new_author	
Language: Ruby	**Type:** internal_operation
Precondition: ¬ ∃ author(Author(author))	**Failure_Handling:** ∃ author(Author(author)) execute Print_Inclusion_Error()
Parameter In:	
Parameter Out:	
Postcondition: new(author, Author) / send(email, Chair)	

Figura 14 – Exemplo de Operation com Pré e Pós-Condições

3.1.3. OpParameter

OpParameter define quais são as informações que integram a operação. Estas informações podem ser de dois tipos de acordo com a relação que OpParameter tiver com Operation:

- In: expressa os parâmetros de entrada da operação;
- Out: expressa os parâmetros de saída da operação.

OpParameter possui um atributo denominado “name”, que identifica o nome do parâmetro.

Os parâmetros, tanto de entrada quanto de saída, podem ter duas especializações diferentes. Essas especializações representam os tipos de dados para o parâmetro, suportados pelo método:

- Primitive: são os tipos de dados primitivos de linguagens de programação. Os tipos escolhidos para serem suportados na operação são:
  - String: textos;
  - Integer: números inteiros;
  - Float: números de ponto flutuante;
  - Date: data (dia, mês, ano);
  - Hash: lista composta de associações chave-valor;
  - List: conjunto de dados de forma a preservar a relação de ordem linear entre eles.
- NotPrimitive: tipos de dados que advêm da relação entre OpParameter e NavClass. Os parâmetros de entrada ou saída são definidos através da ocorrência de instâncias de NavClass. Um parâmetro só pode ser de uma instância de NavClass, no entanto, uma NavClass pode ocorrer em vários parâmetros.

Uma Operation pode possuir zero ou muitos parâmetros de entrada e saída. Porém, os parâmetros só podem pertencer a uma Operation específica.

Notação:

- Name*: name* – Tipo Primitive: *string*
- Name: *institution* – Tipo NotPrimitive: *Institution*
- Name: *author* – Tipo NotPrimitive: *Author*

Exemplo:

Operation: new_author	
Language: Ruby	**Type:** internal_operation
Precondition:	**Failure_Handling:**
¬ ∃ author(Author(author))	∃ author(Author(author)) execute Print_Inclusion_Error()
Parameter In: name:String institution:Institution	
Parameter Out: author:Author	
Postcondition: new(author, Author) / send(email, Chair)	

Figura 15 – Exemplo de Operation com Parâmetros de Entrada e Saída

3.1.4. OpDefinition

Em OpDefinition é definido o código da operação. Deve-se utilizar uma descrição que permita especificar os comportamentos que criam, usam ou modificam um objeto. Pode-se fazer uso de qualquer notação que possibilite a modelagem das transformações ocorridas em decorrência da ação de uma operação. No caso mais geral, usa-se um trecho de código em uma determinada linguagem de programação, que até o presente momento, é a única forma de descrição de comportamento que se encontra implementada. Modelos especializados também são permitidos na modelagem conceitual, tais como máquina de estados e *workflow*, com o intuito de obter um melhor entendimento da lógica de negócio da aplicação.

OpDefinition possui um relacionamento com Operation denominado Is_Defined_By, expressando que uma operação só pode possuir uma definição e vice-versa.

Exemplo:

```
user = Author_.new
user.name = params_user[“name”]
user.birthday = params_user[“birthday”]
user.based_near = params_user[“based_near”]
user.phone = params_user[“phone”]
user.title = params_user[“title”]
user.position = params_user[“position”]
user.academic_degree = params_user[“academic_degree”]
user.email = params_user[“email”]
user.login = params_user[“login”]
user.set_password(@context, nil, params_user[“password”])

if user.save
  return true
else
  return {:error => false, :msg => “nao foi possivel salvar na
base de dados”}
end
```

Figura 16 – Exemplo de OpDefinition em um trecho de código Ruby

O exemplo acima mostra um trecho de código utilizando a linguagem de programação Ruby para o cadastro de um novo autor. Uma nova instância da classe Author_ é criada e os parâmetros recebidos são vinculados como atributos desta instância. Se o novo autor for salvo com sucesso, a operação retorna *true*, senão, retorna um *hash* com o valor *false* e uma mensagem de erro.

3.2. Operações de navegação como instâncias do Modelo de Operações

As aplicações hipermídia possuem como uma de suas principais características, a noção de navegação. A navegação permite que os usuários percorram os itens navegacionais com o objetivo de atender as tarefas descritas na fase de Levantamento de Requisitos.

No SHDM está prevista a etapa de Modelagem Navegacional, responsável por criar o Modelo Navegacional de uma aplicação. O Modelo Navegacional representa uma visão navegacional sobre o esquema conceitual da fase de Modelagem Conceitual.

A partir da inclusão do Modelo de Operações no método SHDM, podemos perceber que as operações de navegação nada mais são que instâncias das operações definidas no Modelo de Operações. As operações do Modelo Navegacional passaram, então, a serem construídas utilizando o vocabulário de definição de operações do Modelo de Operações.

O Modelo Navegacional, por já possuir uma semântica definida, contém um conjunto de operações pré-determinadas com o objetivo de realizar a navegação sobre os elementos da aplicação. Neste modelo temos a descrição das primitivas de navegação, referentes às classes e instâncias do metamodelo que especificam a navegação, juntamente com um conjunto de operações que interpretam essa descrição. As operações de navegação, instâncias das operações do Modelo de Operações, operam sobre as primitivas descritas pelo Modelo Navegacional.

As operações criadas com suporte do Modelo de Operações podem operar sobre qualquer tipo de primitiva, não só as de navegação. Dessa forma, as operações de navegação e qualquer outro tipo de operação se baseiam no Modelo de Operações apenas para aproveitar seus metadados e vocabulário.

Podemos concluir que o Modelo de Operações é tão expressivo quando a própria linguagem Ruby. Isto se deve ao fato de que qualquer tipo de operação pode se basear no modelo e operar sobre qualquer primitiva, excluindo a necessidade de classes especializadas. O Modelo de Operações possibilita criar todo artefato de *software* necessário para interpretar os outros modelos do HyperDE.