2
O Modelo: SetModel

2.1
Modelo de Informação

Modelo de informação é uma representação abstrata e formal de entidades incluindo suas propriedades, relações e operações que podem ser executadas sobre elas. As entidades são utilizadas para modelar domínios. O principal objetivo da definição de um modelo de informação é prover um formalismo para descrever um problema de um domínio sem restringir como essa descrição será mapeada para uma implementação em software. Podem existir diferentes mapeamentos para o modelo de informação, que são chamados de modelos de dados.

Um modelo de dados é uma representação concreta do modelo de informação. Ele representa entidades, propriedades, relações e operações definidas no modelo de informação de uma maneira que permite que instâncias reais destas entidades sejam manipuladas.

O modelo de informação sugerido nesta dissertação é o modelo de conjuntos, que é composto por elementos e conjuntos, onde cada elemento pode estar relacionado a um ou mais conjuntos.

Este modelo facilita a manipulação de coleções de nós similares que estão associados a um ou mais conjuntos. Os usuários transitam de um nó para outro dentro de um mesmo conjunto, e de um conjunto para outro pelos nós que fazem parte do conjunto interseção destes conjuntos. Eles não interpretam os nós como estando ligados diretamente uns aos outros, mas em termos dos conjuntos aos quais eles pertencem.

Utilizando o modelo de conjuntos será possível realizar um mapeamento direto das possíveis tarefas dos usuários para manipulações sobre os conjuntos que estarão identificando os dados de uma aplicação. O modelo de processamento da informação proposto relaciona o problema do usuário em um domínio de dados e consequentemente suas tarefas são interpretadas em cima do modelo de informação.

O modelo, neste caso, é definido sobre um domínio específico, mas pode ser genérico, e dessa forma os itens de informação são itens do domínio.

O modelo proposto é formado por M = <S, E, R>, conjuntos, elementos e relações. Através dos atributos e relações de um elemento, é possível gerar conjuntos de elementos. Por exemplo, um elemento do tipo *Livro* que possua uma relação *escrito por* com um elemento do tipo *Autor* pode gerar um conjunto de todos os livros escrito por este *Autor*, ou todos os *Autores* deste *Livro.* O mesmo ocorre com as queries arbitrárias que podem ser executadas sobre as propriedades dos elementos, também gerando novos conjuntos.

O usuário, tendo conhecimento do domínio, poderá gerar conjuntos a partir de propriedades induzidas de relações entre elementos.

Para resolver um problema precisamos manipular modelos. Essa manipulação engloba navegação. Ao obtermos objetos de um conjunto estamos realizando manipulações e ao mesmo tempo uma navegação, análogo ao que acontece ao hipertexto clássico. Neste caso, estar navegando significa estar manipulando conjuntos.

2.2 Vantagens

Este modelo, diferente dos utilizados atualmente, não está somente baseado no hipertexto clássico e conseqüentemente não está baseado em grafos. Para entender melhor o modelo proposto é necessário que o universo de informações seja analisado como uma composição de conjuntos.

O modelo oferece uma visão das informações em um nível mais alto do que vem sendo utilizado e mais próximo ao usuário. É um modelo capaz de expressar facilmente tarefas típicas de um usuário utilizando um sistema em diversos tipos de navegação. Ele unifica o modelo clássico com o modelo de conjuntos, que contém todas as operações possíveis de serem executadas sobre um conjunto.

Temos então a oportunidade de criar um modelo que unifique essas duas propriedades fundamentais que por sua vez nos permitirá mapear as tarefas do usuário claramente, podendo associar essas tarefas com operações em conjuntos, e criar um ambiente de implementação apropriado para a geração de sistemas que manipulem as informações como conjuntos e elementos.

Estas operações abrangem desde operações clássicas até operações que são induzidas pelo próprio modelo de domínio, ou seja, operações que fazem parte do contexto conhecido pelo usuário.

Uma DSL foi gerada para que o usuário programador possa interagir com o modelo criado. A DSL é responsável por executar as operações feitas sobre o modelo de informação e representar todas as possíveis manipulações que podem ser realizadas no domínio em questão (ARDF; NUNES, 2003 e ROR). Mais detalhes sobre a DSL proposta no próximo capítulo.

2.3 Novas operações

Como foi dito anteriormente, há algumas formas de navegação atualmente que utilizam uma operação básica de conjuntos, a interseção. As consultas realizadas em sites dessa categoria podem ser identificadas simplesmente como seguidas operações de interseção.

Nesse novo modelo, novas operações são propostas e implementadas para que se possa atender melhor às necessidades de diversos tipos de interfaces e soluções que podem ser transformadas e/ou representadas por conjuntos.

De acordo com os casos de uso descritos no capítulo 1, podemos enumerar as seguintes operações. Cada operação está relacionada a um caso de uso apenas para exemplificação, pois inúmeras outras situações podem utilizar estas operações. A solução para cada caso de uso também será expressa como um programa da DSL.

Caso de uso 1: como o usuário deseja ver todas as facetas em que um item selecionado faz parte, seria necessária uma operação que, dado um item, fosse possível encontrar todas as facetas às quais este está associado.

Solução: permitir ao usuário que veja de alguma forma todas as facetas que estão relacionadas ao objeto em questão e que possa navegar por essas outras facetas. Neste caso será necessária uma funcionalidade que a partir de um elemento possa ser possível definir todos os relacionamentos do mesmo e a partir de um relacionamento identificado definir quais elementos são pertinentes.

Programa:

ArrayList relations = element.ReturnRelations();

O usuário seleciona qual relação deseja e então esta relação é aplicada sobre o elemento, retornando assim um conjunto:

Set newSet = element.Allfacetas;

Caso de uso 2: como o usuário quer ver os itens que não pertencem a uma faceta, é necessária a operação clássica de *diferença* da teoria de conjuntos. Todos os elementos que estão no conjunto A e não estão no conjunto B, por exemplo. Neste caso A seria formado por todos os vinhos e B pelos vinhos que pertencem aos Estados Unidos.

Solução: permitir que o usuário faça uma operação de diferença entre os conjuntos de faceta, ou seja, ver objetos que pertencem a alguns conjuntos e não pertence a outros. Atualmente só é possível ver a lista de vinhos que pertencem aos Estados Unidos, mas não ver todos os vinhos existentes, exceto os que pertencem aos Estados Unidos.

Expressão em conjuntos:

se (usuário seleciona {vinhos dos Estados Unidos}) então

Universo = Universo – {vinhos dos Estados Unidos}

Programa:

Set universo = new Set(); // conjunto universo

Set EUA = Class.findSetBy("region", "EUA"); // conjunto de vinhos dos EUA

universo = universo.Difference(EUA);

Caso de uso 3: duas soluções distintas poderiam ser adotadas. Para a primeira solução seria necessária a operação de *união*, onde o usuário poderia escolher *n* facetas, e então seus itens seriam mostrados em uma mesma lista. Para a segunda solução, seria necessária novamente a *união* para o usuário poder selecionar *n* facetas e então a *interseção* para poder incluir uma restrição.

Solução 1: permitir que o usuário selecione mais de 1 categoria, independente de hierarquia, e faça um pacote com categorias de sua preferência ordenado pelo meta-dado desejado.

Expressão em conjuntos:

Pacote = { }

se (usuário seleciona {X}) então

  Pacote = Pacote U {X}

Programa:

Set pacote = new Set(); // conjunto que representa o pacote a ser montado

Set X = new Set(); // conjunto que representa a categoria selecionada

pacote = pacote.Union(X);

Solução 2: permitir ao usuário que selecione 2 tipos de produtos, por exemplo, câmera digital e PDA e inclua uma restrição, por exemplo, que o formato não seja "memory stick". Isto seria uma manipulação de conjuntos, complemento de itens que não tenham memory stick e então uma interseção com o conjunto de elementos gerado pela união dos que são máquina digital e os que são PDA.

Neste caso o usuário estaria selecionando 2 grupos distintos e colocando uma restrição para ser aplicada nesse novo grupo formado. Também pode ser interpretada como uma operação de união entre esses 2 grupos e então uma operação de subtração para retirar os elementos que utilizam como memória o padrão da Sony "memory stick".

Expressão em conjuntos:

se (usuário seleciona *camera digital*) então

  Pacote = Pacote U camera digital

se (usuário seleciona *PDA*) então

  Pacote = Pacote U PDA

se (usuário deseleciona *itens sem memory stick*) então

  Pacote = Pacote ∩ Complemento(itens sem memory stick)

Programa:

Set pacote = new Set(); // conjunto que representa o pacote a ser montado

Set câmera = new Set(); // conjunto que representa as câmeras digitais

Set PDA = new Set(); // conjunto que representa os PDAs

Set MS = universo.Difference(itensSemMS) ; // conjunto que representa itens com MS

pacote = pacote.Union(câmera);

pacote = pacote.Union(PDA);

pacote = pacote.Intersection(MS);

Caso de uso 4: neste caso seria utilizada uma funcionalidade que, a partir de uma função definida pelo usuário e aplicada a um dado conjunto, um novo conjunto resultante seria formado.

Solução: criar uma função que só considere os elementos de um conjunto de acordo com uma data e então aplicá-la sobre o conjunto de todas as fotos existentes. Essa função estaria funcionando como um filtro por data, retornando as fotos com a mesma data de uma foto escolhida.

Expressão em conjuntos:

Conjuntonovo = função(Conjunto)

Programa:

Set newSet = originalSet.MapSet(new Set.FunctionMapSet(DateFunction));

Caso de uso 5: neste caso seria utilizada uma funcionalidade que, a partir de uma função definida pelo usuário e aplicada a um dado conjunto, seja retornado um valor escalar.

Solução: criar uma função que some todos os preços de cada elemento de um conjunto. Essa função seria aplicada ao conjunto que o usuário encontrou, somando todos os preços e retornando um valor escalar com o somatório.

Expressão em conjuntos:

valor = função(Conjunto)

Programa:

int soma = (int)originalSet.Map(new Set.FunctionMap(Sum));

Caso de uso 6: neste caso seria utilizada a função de relação entre itens de um conjunto, as operações de interseção, união e mapeamento de funções e mapeamento de conjuntos.

Os hotéis teriam um atributo do tipo “localizado em”, que apontaria para uma cidade. Então, seria feita primeiramente a interseção entre as cidades e as cidades dos hotéis e teríamos como resultado uma lista de hotéis que estão localizados nas três cidades. Para isto, utilizamos o mapeamento de um conjunto em outro. Com posse destes dados, poderíamos utilizar a operação de união para incluir os hotéis selecionados pelo usuário deste resultado no pacote.

Posteriormente, faríamos uma interseção das linhas de trem com as cidades. Cada linha teria um atributo do tipo “passa por”, que apontaria para uma lista de cidades por onde o trem passa. Assim, faríamos a inteserção entre as cidades escolhidas e as cidades pelas quais os trens passam. Da mesma forma, utilizaríamos o mapeamento de conjuntos. Como resultado, teríamos somente as linhas que passam pelas três cidades escolhidas pelo usuário. Novamente, seria feita uma operação de união para incluir as linhas de trem selecionadas pelo usuário deste resultado no pacote.

Finalmente, utilizaríamos o mapeamento de função que calcularia a soma de todos os itens deste pacote.

Expressão em conjuntos:

Pacote = { }

Resultado = { }

Resultado = mapeamento(Cidades Escolhidas) ∩ Hotéis Selecionados

Resultado = função(Resultado)

Pacote = Pacote U Resultado

Resultado = mapeamento(Cidades Escolhidas) ∩ Linhas de Trens

Resultado = função(Resultado)

Pacote = Pacote U Resultado

Soma = função(Pacote)

Programa:

Set pacote = new Set();

Set resultado = new Set();

resultado = sets.MapIn(CidadesEscolhidas, “possui”, Hoteis, Union, Intersection);

resultado = resultado.MapSet(new Set.FunctionMapSet(HotelComCafeContinental));

pacote = pacote.Union(resultado);

resultado = sets.MapIn(CidadesEscolhidas, "possui", LinhasTrem, Union, Intersection);

resultado = resultado.MapSet(new Set.FunctionMapSet(PassagemEntre50e70));

pacote = pacote.Union(resultado);

int soma = (int)pacote.Map(new Set.FunctionMap(Sum));

Caso de uso 7: neste caso seria utilizada a operação de interseção entre os produtos que possuem as características desejadas.

Seria realizada uma operação de interseção entre o conjunto de produtos que possuem a tecnologia wireless e o conjunto de produtos que são modem ADSL. Depois, uma interseção do conjunto resposta com o conjunto de produtos que são roteadores. E então, interseção com o conjunto de produtos que são switch. Por fim, uma interseção com o conjunto de produtos que possuem a tecnologia VoIP.

Expressão em conjuntos:

Produtos = { }

Produtos = tecnologia wireless ∩ modem ADSL

Produtos = Produtos ∩ roteadores

Produtos = Produtos ∩ switch

Produtos = Produtos ∩ VoIP

Programa:

Set produtos = new Set();

produtos = TecnologiaWireless.Intersection(ModemADSL);

produtos = produtos.Intersection(Roteadores);

produtos = produtos.Intersection(Switch);

produtos = produtos.Intersection(VoIP);

Caso de uso 8: neste caso seria utilizada a operação de diferença e o mapeamento entre conjuntos.

Primeiramente, tendo os dois professores que não devem fazer parte da busca, realiza-se uma operação de diferença com o universo de professores. E então, realizamos uma operação de interseção entre os professores restantes e as dissertações através do mapeamento de conjuntos. Cada orientador possui, por exemplo, um atributo do tipo "orientou", que aponta para uma dissertação. Portanto, seria feita a união entre todas as dissertações orientadas pelos professores selecionados e uma interseção com todas as dissertações encontradas no repositório. Teríamos como resultado as dissertações que tiveram como orientador os professores selecionados. Por fim, necessitaríamos realizar o mesmo processo, porém agora com a área que cada dissertação possui como atributo.

Expressão em conjunto:

Professores Busca = { }

Resultado = { }

Professores Busca = Universo – Professores Escolhidos

Resultado = mapeamento(Professores Busca) ∩ Dissertações

Resultado = Resultado ∩ mapeamento(Áreas)

Programa:

Set professoresBusca = new Set();

Set resultado = new Set();

professoresBusca = universo.Difference(professoresEscolhidos);

resultado = sets.MapIn(professoresBusca, "orienta", dissertações, Union, Intersection);

resultado = sets.MapIn(resultado, "possui", Áreas, Union, Intersection);