6
O Ambiente de Design Integrado

Formalizar a representação de *design rationale* usando a linguagem definida por Kuaba, enriquecida com a semântica do modelo formal que descreve os artefatos, permite uma nova forma de reuso de design de software: o reuso através da integração de *design rationales* existentes para iniciar um novo design. Este tipo de reuso requer diferentes tipos de operações sobre o *design rationale* registrado, que envolvem combinar instâncias da ontologia Kuaba (representações de *design rationale*) para compor uma solução de design mais completa.

O ambiente de design proposto neste trabalho implementa as diversas operações necessárias à integração de representações de *design rationale*. Este ambiente usa o modelo formal do artefato para sugerir opções de design a cada passo do processo de design e registrar as escolhas feitas pelo projetista, usando a linguagem de representação definida pela ontologia Kuaba. Além disso, o ambiente apóia também a consulta de *rationale* de designs anteriores e o reuso das idéias de solução propostas nestes designs.

Neste capítulo, apresentamos inicialmente a arquitetura conceitual do ambiente de design integrado, proposto para apoiar o registro e o uso de *design rationale* durante o design de software (seção 6.1). Em seguida, descrevemos as operações necessárias à integração de *design rationales* e a implementação de uma destas operações (seção 6.1.1).

6.1. A Arquitetura Conceitual do Ambiente

A arquitetura conceitual proposta para o ambiente de design integrado inclui dois componentes principais: uma ferramenta de design de software estendida e o "processador" de *design rationale*, que é capaz de processar as representações de *design rationale* geradas com Kuaba. A Figura 42 ilustra os componentes da arquitetura proposta.

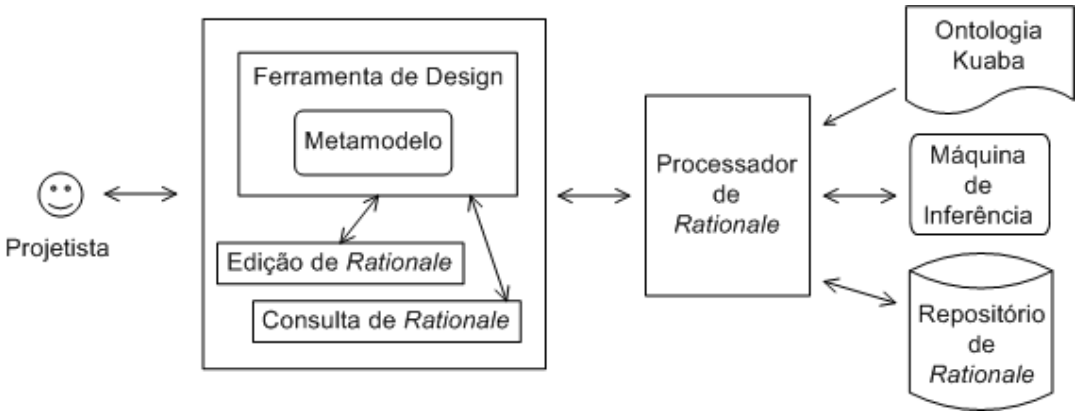

Figura 42 – Arquitetura conceitual de um ambiente apoiando *Design Rationale*

Considerando que a maioria das ferramentas de design de software já utiliza algum tipo de descrição formal dos artefatos sendo projetados, propomos uma extensão dessas ferramentas para permitir sua integração com o "processador" de *design rationale* desenvolvido neste trabalho. Como podemos observar na Figura 42, esta extensão enriquece as ferramentas de design adicionando duas camadas para apoiar a edição e a consulta de *design rationale*. Na camada de edição, o projetista informa os argumentos contra ou a favor das idéias de design consideradas e as justificativas para as decisões tomadas. Na camada de consulta, o projetista pode buscar designs existentes com seus *rationales*, formular questões sobre os designs encontrados e solicitar a integração de *rationales* para iniciar um novo design. Nesta camada, o projetista também pode visualizar graficamente o *rationale* do artefato que está sendo projetado, ou dos designs que estão sendo reusados em seu design.

Na arquitetura proposta, a ferramenta de design transfere as opções de design e as informações de *rationale* fornecidas pelo projetista para o "processador" de *rationale*, responsável por criar as representações de *design rationale* (instâncias da ontologia Kuaba), realizar consultas no repositório de fatos, e processar a integração de *rationales*, quando solicitado pelo projetista. Em uma versão futura, a ferramenta de design também será capaz de gerar o artefato, baseada nas modificações e decisões tomadas pelo projetista sobre o *rationale* integrado.

6.1.1. O Processador de Rationale

Como mencionamos anteriormente, o uso das representações de *design rationale* requer diferentes tipos de operações sobre o conteúdo registrado. A

representação explícita e semântica de *design rationale* em uma linguagem formalmente definida e especificamente projetada para a descrição de ontologias permite que estas operações sejam computáveis por máquinas para apoiar o design de novos artefatos. O "processador" de *rationale* é um programa que implementa tais operações para criar e processar representações de *design rationale* usando a ontologia Kuaba.

As operações sobre as representações de *design rationale* podem ser agrupadas em:

- consultas;
- operações para criação e manipulação de uma representação (uma instância da ontologia);
- operações para integração de duas ou mais representações (duas ou mais instâncias da ontologia).

O primeiro grupo permite formular questões relevantes sobre o design realizado e sobre o artefato produzido. Por exemplo, podemos formular questões do tipo "*Quais foram as idéias de solução propostas durante o design do artefato CD?*", "*Por que a decisão de modelar gênero como um atributo foi tomada?*", "*Quem propôs a idéia de solução adotada para o artefato gênero?*", ou ainda, "*O artefato X está relacionado ao artefato Y?*". As consultas são realizadas de acordo com as relações semânticas, restrições e regras de inferência definidas para a ontologia Kuaba.

As operações para a criação e manipulação de representações de *design rationale* envolvem a criação e a destruição de instâncias das classes e propriedades definidas na ontologia Kuaba. Estas operações já estão implementadas na maioria dos editores de ontologia disponíveis, como por exemplo, Protégé (Noy et al., 2001). São elas:

- criar novos elementos de raciocínio (questões, idéias e argumentos);
- associar elementos de raciocínio;
- excluir elementos de raciocínio;
- criar novas decisões e justificativas
- alterar e/ou excluir decisões e justificativas existentes.

As operações do terceiro grupo, operações para integração de duas ou mais representações, envolvem tratamentos mais específicos. Estas operações

assemelham-se às operações necessárias para realizar o alinhamento de ontologias (Doan et al., 2002). No entanto, as operações para integração das representações de *design rationale* definidas neste trabalho diferem das operações de alinhamento usuais, uma vez que elas envolvem a combinação de instâncias de uma mesma ontologia (Kuaba) e não a combinação das taxonomias e instâncias definidas em ontologias distintas. Basicamente, as operações para a integração de *rationales* envolvem a busca, a cópia, a substituição e a união dos elementos de raciocínio presentes nas representações de *design rationale* sendo integradas.

As operações de busca permitem ao projetista selecionar quais elementos das representações consideradas na integração serão incluídos no novo design. Por exemplo, o projetista poderia fornecer uma questão e solicitar que a ferramenta de busca recupere apenas as idéias associadas a esta questão que possuem argumentos a seu favor.

As operações de substituição permitem ao projetista substituir um elemento em uma representação por um elemento correspondente em uma outra representação. Esta operação pode ser usada, por exemplo, quando o projetista deseja usar o *design rationale* de uma representação, mas deseja substituir um de seus elementos por um elemento especificado em outra representação.

As operações de cópia permitem ao projetista copiar elementos de uma representação para outra.

Finalmente, as operações de união permitem unir os elementos de raciocínio descritos nas representações de *design rationale* consideradas na integração para gerar um novo design. Estas operações podem ser implementadas de diversas formas, permitindo ao projetista determinar como a união dos elementos será realizada. Uma forma é permitir ao projetista especificar que partes das representações consideradas devem ser integradas. Por exemplo, ele pode definir a questão de design a partir da qual a união das representações será realizada. Ou ainda, ele pode restringir os elementos considerados durante a integração, por exemplo, especificando que a união deve considerar apenas as idéias que foram aceitas em suas respectivas representações. Geralmente a união de duas representações de *design rationale* envolve:

- unir as idéias que respondem a questões equivalentes;
- unir as questões sugeridas por idéias equivalentes;
- unir os argumentos associados a idéias equivalentes.

6.1.1.1. A Implementação da Operação de União

Até o momento, o processador de *rationale* implementa a união completa de duas representações de *design rationale*. Esta operação consiste de um conjunto de regras implementadas na linguagem Flora-2[8], que traduz F-logic em código Prolog e o processa no sistema dedutivo XSB[9]. Estas regras são definidas com base nos elementos de raciocínio definidos no vocabulário da ontologia Kuaba e consistem, basicamente, no processamento recursivo das diversas sub-árvores de elementos que compõem as representações de *design rationale*.

Como mencionado anteriormente no capítulo 5, a união ou integração de duas representações de *design rationale* envolve: a definição da representação que será usada como base para o *design rationale* integrado; a especificação de igualdade para elementos equivalentes entre as representações; e a união dos elementos de raciocínio (equivalentes ou não) das duas representações.

A representação indicada como base pelo projetista representa parte do *design rationale* integrado que passa a ser modificado pelo processador de *rationale* com os resultados da integração. Já a especificação de igualdade entre os elementos das representações envolvidas na integração é usada pelo processador de *rationale* na identificação de questões e idéias equivalentes que devem ser tratadas durante a união. Esta igualmente é especificada formalmente na linguagem Flora-2 pelo predicado ":=:". Por exemplo, considerando a integração de designs para construir um novo diagrama de classes para o domínio de CDs, abordada no capítulo 5, a igualdade entre a idéia de solução "*Gênero*" proposta em um design, e a idéia "*Categoria*" usada em outro design, é especificada da seguinte forma:

```
categoria:=:gênero.
```

Baseado nessa especificação de igualdade, o processador de *rationale* primeiramente identifica as idéias equivalentes entre as duas representações de *design rationale*, aplicando as regras de equivalência ilustradas abaixo. De acordo com estas regras questões são consideradas equivalentes se elas possuem o mesmo texto (ou fazem parte da especificação de igualdade definida pelo projetista) e são sugeridas por idéias equivalentes. Idéias de domínio são consideradas equivalentes

[8] http://flora.sourceforge.net

[9] http://xsb.sourceforge.net

se elas possuem o mesmo texto e respondem a questões que possuem o mesmo texto ou fazem parte da especificação de igualdade definida pelo projetista. Finalmente, idéias de design são consideradas equivalentes se elas possuem o mesmo texto e respondem pelo menos uma mesma questão equivalente.

```
equivalent_question(Question1, Question2) :- Question1[hasText->_TQ1, isSuggestedBy->>_I1]@mod1,
                                             Question2[hasText->_TQ2, isSuggestedBy->>_I2]@mod2,
                                             (_TQ1=_TQ2;Question1:=:Question2),
                                             ( equivalent_domainIdea(_I1,_I2);
                                               equivalent_designIdea(_I1,_I2);
                                               _I1:=:_I2 ).

equivalent_domainIdea(Idea1, Idea2) :- Idea1[not (isDefinedBy->_X),hasText->_TI1,
                                            address->>_Q1[hasText->_TQ1]]@mod1,
                                       Idea2[not (isDefinedBy->_Y),hasText->_TI2,
                                            address->>_Q2[hasText->_TQ2]]@mod2,
                                       (_TI1=_TI2;Idea1:=:Idea2),
                                       (_TQ1=_TQ2;_Q1:=:_Q2), not(_Q1=_Q2).

equivalent_designIdea(Idea1, Idea2) :- Idea1[hasText->_TI1]@mod1,
                                       Idea2[hasText->_TI2]@mod2,
                                       _TI1=_TI2,
                                       LQ1=collectset{Q1|Q1[isAddressedBy->>Idea1]@mod1},
                                       LQ2=collectset{Q2|Q2[isAddressedBy->>Idea2]@mod2},
                                       get_equivalent_question(LQ1,LQ2,List),
                                       not(List = []).
```

Observe nas regras acima que, de forma diferente de F-logic, em Flora-2 o cabeçalho da regra é separado do corpo da regra pelo símbolo “`:-`” ao invés de “`<-`”, e “@mod1” e “@mod2” especificam *namespaces* diferentes para as representações usadas na integração. Um resumo da sintaxe usada pela linguagem Flora-2 e suas diferenças com relação à sintaxe da linguagem F-logic é apresentada no apêndice A.

Caso o processador de *rationale* encontre alguma idéia equivalente entre as representações sendo integradas, ele simplesmente copia os argumentos desta idéia da representação original para a representação base. Assim, no exemplo de integração de designs mencionado anteriormente, os argumentos apresentados pelo projetista para a idéia “*Classe*” na sub-árvore de “*Gênero*” são copiados para a idéia equivalente na sub-árvore de “*Categoria*” na representação base, como ilustra a figura abaixo.

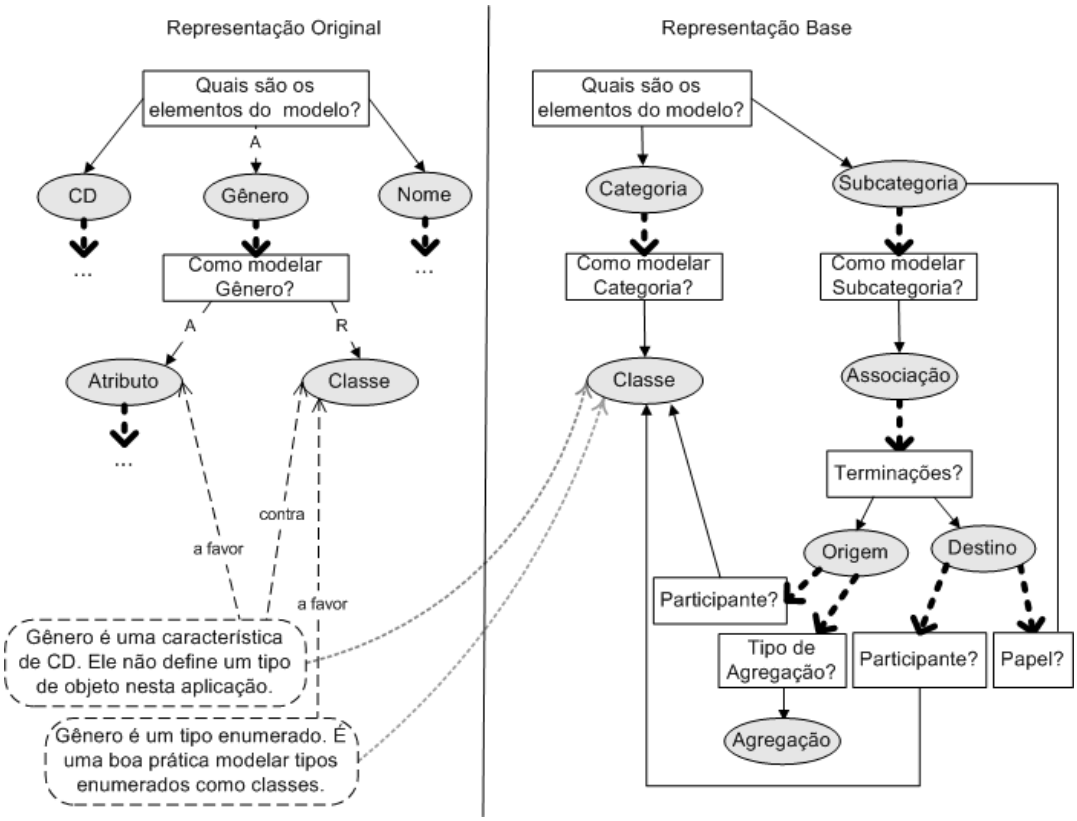

Figura 43 – Exemplo da cópia de argumentos para idéias equivalentes

O Quadro 7 mostra um trecho da representação base modificada após a cópia dos argumentos. Note que os identificadores dos dois argumentos apresentados para as idéias "*Atributo*" e "*Classe*" na sub-árvore do elemento "*Gênero*" foram incluídos como valores da relação "*tem argumento*" na descrição da idéia "*Classe*" para o elemento "*Categoria*" na representação base. As descrições dos argumentos copiados também foram alteradas para fazer referência à idéia "*Classe*" associada ao elemento "*Categoria*".

```
/* Facts */
// Reasoning Elements ----------
'categoryClass'          : 'idea'.
'categoryClass'          : 'symbol'.
'categoryClass'[
   hasText -> 'Class',   isIndicatedBy -> 'uml',
   isInvolved -> 'conceptualModel',  isPresentedBy -> 'daniel',
   hasCreationDate -> '2004-08-15T09:21:12',  suggests ->> {'whatElements'},
   address ->> {'hwModelCategory','subcategoryAssocDestParticipant',
                'subcategoryAssocOriginParticipant','whoseAttribCategoryName'},
   hasArgument ->> {'categoryClassArgument','categoryClassArgument2',
                    'genreAttribArgument','genreClassArgument1',
                    'genreNameAttribArgument2'}
].
'genreAttribArgument'   : 'argument'.
'genreAttribArgument'   : 'symbol'.
'genreAttribArgument'[
  hasText -> 'Genre is a feature of CD. It does not define an object type in this
              application',
   hasCreationDate -> '2004-08-25T09:21:56', isInvolved -> 'domainModelActivity',
   inFavorOf ->> {'genreAttribute'},   objectsTo ->> {'categoryClass'}
].
'genreClassArgument1':  'argument'.
'genreClassArgument1':  'symbol'.
'genreClassArgument1'[
   hasText -> 'Genre is an enumerated type. It is a good practice to model
               enumerated types as classes',
   hasCreationDate -> '2004-08-25T09:22:00', isInvolved -> 'domainModelActivity',
   inFavorOf ->> {'categoryClass'}
].
```

Quadro 7 - Exemplo das modificações realizadas na representação base após a cópia dos argumentos

Após identificar e tratar as idéias equivalentes, o processador de *rationale* identifica as questões que são diferentes (não-equivalentes) nas duas representações e as copia para a representação base. As operações usadas pelo processador para tratar questões não-equivalentes são ilustradas abaixo.

```
?- L=collectset{Q|Q[isSuggestedBy->>I]@mod1},
   L2=collectset{Q|Q[isSuggestedBy->>I]@mod2},
   get_non_equivalent_question(L,L2,List), copy_question(List).
```

No trecho de código acima, o processador de *rationale* cria duas listas (L, L2) com questões das duas representações de *design rationale* sendo integradas. Note que as questões recuperadas devem ter sido sugeridas por alguma idéia. Isto

permite distinguí-las das questões iniciais de cada representação, que são sempre equivalentes pelo fato de serem definidas pelo metamodelo do método de design utilizado. A operação "*get_non_equivalent_question*" processa estas listas, retornando uma nova lista com as questões diferentes. A operação "*copy_question*" mostrada abaixo copia as questões desta lista para a representação base. Caso a questão copiada seja respondida por uma idéia equivalente a uma idéia já existente na representação base, o processador altera a especificação dessa idéia existente para que ela passe a responder também a questão copiada. Este tratamento é realizado pela operação "*identify_equivalent_idea*", cujo código é apresentado no apêndice C.

```
copy_question([]).
copy_question([Question1|Tail]) :-
        insert{(Question1:question, Question1[hasText->X, hasCreationDate->Y,
                hasType->T, isInvolved->Z, isDefinedBy->W])@mod2 |
                Question1[hasText->X, hasCreationDate->Y, hasType->T,
                isInvolved->Z,isDefinedBy->W]@mod1},
        LI1=collectset{I|I[address->>Question1]@mod1},
        LI2=collectset{I|I:idea@mod2},
        identify_equivalent_idea(LI2,LI1,Question1),
        copy_question(Tail).
```

Após tratar as questões não-equivalentes, o processador de *rationale* identifica as idéias que são diferentes (não-equivalentes) nas duas representações e as copia para a representação base. As operações usadas pelo processador para tratar as idéias diferentes são ilustradas abaixo.

```
?- L=collectset{I|I:idea@mod1},
   L2=collectset{I|I:idea@mod2},
   get_non_equivalent_idea(L,L2,List), copy_idea(List).
```

De forma semelhante ao tratamento dado às questões, o processador de *rationale* também cria duas listas (L, L2) com as idéias das duas representações de *design rationale* sendo integradas. A operação "*get_non_equivalent_idea*" processa estas listas, retornando uma nova lista com as idéias diferentes. A operação "*copy_idea*" mostrada abaixo copia as idéias desta lista para a representação base. Para cada idéia copiada, note que o processador copia também seus argumentos (operação "*copy_argument*") e trata questões equivalentes às questões respondidas pelas idéias copiadas (operação "*identify_equivalent_question*"). Caso já exista na representação base uma questão

equivalente à questão respondida pela idéia copiada da representação original, o processador altera a especificação dessa idéia para que ela passe a responder a questão existente na representação base, mantendo a consistência das relações entre questões e idéias na representação integrada. O código referente a este tratamento encontra-se detalhado no apêndice C.

```
copy_idea([]).
copy_idea([Idea1|Tail]) :-
        if not(Idea1:idea@mod2)
        then
          (insert{(Idea1:idea,Idea1[hasText->X, hasCreationDate->Y, isInvolved->Z])@mod2 |
                  Idea1[hasText->X, hasCreationDate->Y, isInvolved->Z]@mod1},
           insert{Idea1[address->>{Q}]@mod2 | Idea1[address->>Q]@mod1},
           insert{Idea1[hasArgument->>{A}]@mod2 | Idea1[hasArgument->>A]@mod1},
           LA=collectset{A1|A1[inFavorOf->>Idea1]@mod1;A1[objectsTo->>Idea1]@mod1},
           copy_argument(LA,Idea1),
           LQ1=collectset{Q1|Q1[isAddressedBy->>Idea1]@mod1},
           LQ2=collectset{Q2|Q2:question@mod2},
           identify_equivalent_question(LQ2, LQ1, Idea1),
           if Idea1[suggests->>_X]@mod1
           then
             (insert{Idea1[suggests->>{Q1}]@mod2 | Idea1[suggests->>Q1]@mod1}),
              if Idea1[isDefinedBy->M]@mod1
              then
                 (insert{Idea1[isDefinedBy->M]@mod2 | Idea1[isDefinedBy->M]@mod1}))
              else
                 (insert{Idea1[address->>{Q}]@mod2 | Idea1[address->>Q]@mod1}),
          copy_idea(Tail).
```

No exemplo de integração de *design rationales* para o design do elemento "*Gênero*", o processador de *rationale* copia para a representação base as idéias de domínio "*CD*" e "*Nome*", a idéia de design "*Classe*" da sub-árvore de CD, as idéias "*Atributo*", "*String*", "*1*" e "*n*" da sub-árvore de Gênero e as idéias de design "*Atributo*" e "*String*" da sub-árvore de Nome, como ilustra a Figura 44. Para simplificar a ilustração não representamos a cópia da idéia de domínio "*Nome*" e das idéias "*Atributo*" e "*String*" de sua sub-árvore.

Neste exemplo podemos observar que as idéias copiadas são associadas às questões não-equivalentes copiadas anteriormente para a representação base, como por exemplo, as questões "*Tipo?*" e "*Multiplicidade Mínima?*". Note que a relação entre a questão "*De quem?*" sugerida pela idéia de design "*Atributo*" e a idéia "*Classe*" na sub-árvore do elemento "*CD*" foi mantida na representação integrada, mantendo a consistência com o modelo formal da UML para diagrama de classes.

De acordo com este modelo formal, um atributo sempre deve estar associado a uma classe.

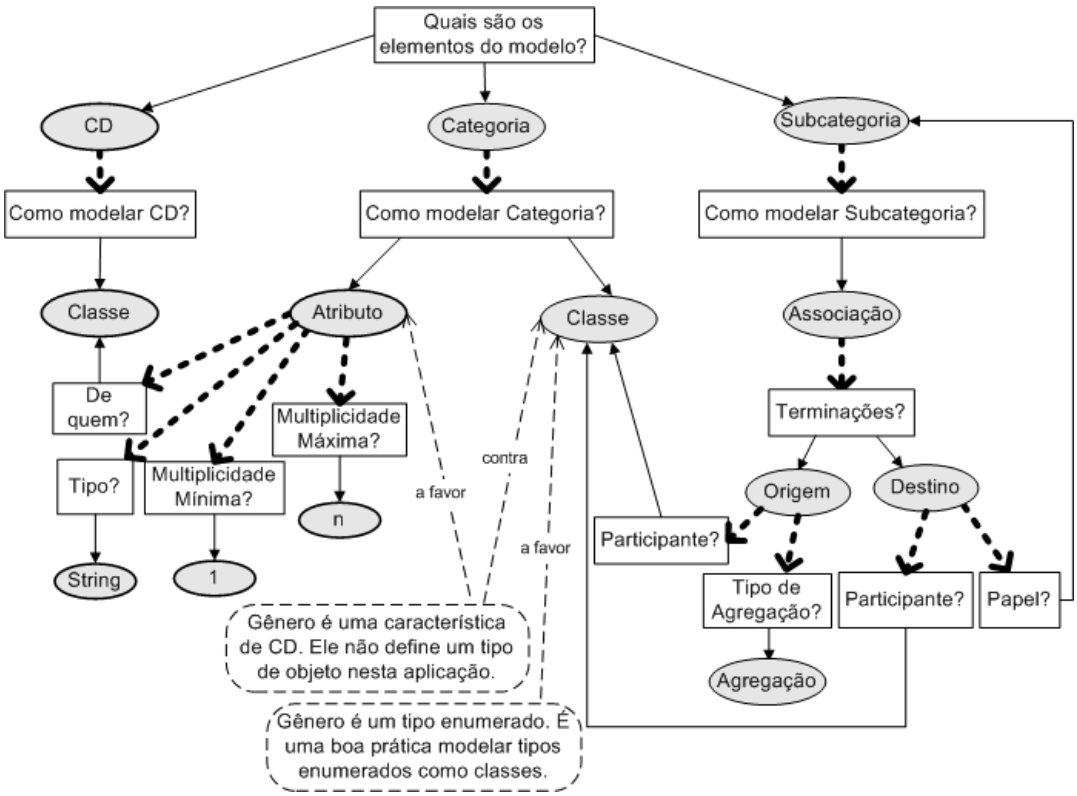

Figura 44 – Representação de *design rationale* obtida após a operação de união

O quadro abaixo mostra um trecho da representação base modificada após a cópia das idéias não equivalentes. Neste exemplo, podemos observar que a idéia de design "*Atributo*" é incluída na representação base já com o valor da relação "*responde*" modificado para a questão "*Como modelar Categoria?*". Para manter a consistência do *design rationale*, esta idéia também é incluída como um dos valores da relação "*é respondida por*" na especificação da questão sobre como modelar categoria.

```
/* Facts */
// Reasoning Elements ----------
'hwModelCategory'      : 'question'.
'hwModelCategory'      : 'symbol'.
'hwModelCategory'[
   hasText -> 'How to model Category?',
   hasType -> 'XOR',
   isInvolved -> 'conceptualModel',
   isPresentedBy -> 'daniel',
   isAddressedBy ->> {'categoryClass', 'genreAttribute'},
   isSuggestedBy ->> {'category'},
   hasCreationDate -> '2004-08-15T09:15:02',
   isIndicatedBy -> uml
].
'genreAttribute'           : 'idea'.
'genreAttribute'           : 'symbol'.
'genreAttribute'[
   hasCreationDate -> '2004-08-25T09:21:02',
   hasText -> 'Attribute',
   isInvolved -> 'domainModelActivity',
   address ->> {'hwModelCategory'},
   hasArgument ->> {'genreAttribArgument'},
   suggests ->> {'maxMultAttribGenre','minMultAttribGenre','whoseAttrGenre'},
   isIndicatedBy -> uml
].
```

Quadro 8 – Exemplo das modificações realizadas na representação base, após a cópia das idéias não-equivalentes

Finalizando a integração, o processador de *rationale* trata as questões equivalentes entre as duas representações sendo integradas para garantir a consistência nas relações entre as questões e idéias copiadas para a representação base. Este tratamento é realizado pela operação "*verify_equivalence_question*" ilustrada abaixo. O código desta operação encontra-se detalhado no apêndice C.

```
?- L=collectset{Q|Q:question@mod1},
   L2=collectset{Q|Q:question@mod2},
   verify_equivalence_question(L,L2,List).
```