3
Kuaba: Uma Ontologia para Design Rationale

Para que o conhecimento registrado durante o design possa ser automaticamente processado, é desejável representar o *design rationale* de uma maneira formalmente precisa e computável. Uma forma de obter isto é por meio de ontologias, uma vez que, como definido em (Gruber, 1993), elas representam "uma especificação explícita de uma conceitualização compartilhada". Ontologias são representações de conhecimento, onde um conjunto de objetos e seus relacionamentos são descritos por um vocabulário definido.

Neste trabalho, propomos a ontologia Kuaba como um modelo de representação de conhecimento para o domínio de *design rationale*. Nosso objetivo ao propor esta ontologia é fornecer um vocabulário que permita atribuir semântica ao conteúdo do *design rationale* registrado, e definir um conjunto de regras que possibilite a realização de inferências e operações computáveis para apoiar o uso de *design rationale*. A versão atual da ontologia Kuaba está descrita em F-logic (Kifer & Lausen, 1989). A primeira versão da ontologia foi especificada em OWL (Web Ontology Language) (W3C, 2004) e o processamento das regras de inferência realizado no sistema comercial OntoBroker[4]. No entanto, concluímos que o uso de um sistema comercial poderia prejudicar a implementação e a utilização futura do ambiente de design proposto nesta tese. Por isso, decidimos utilizar a linguagem F-logic devido à maior disponibilidade de máquinas de inferência gratuitas para o processamento das representações de *design rationale* geradas com Kuaba.

Neste capítulo, apresentamos inicialmente um resumo da sintaxe F-logic usada na definição da ontologia Kuaba (seção 3.1), visando facilitar a compreensão dos exemplos mostrados ao longo da tese. Em seguida, descrevemos os elementos que compõem o vocabulário da ontologia Kuaba (seção 3.2) e as regras definidas para apoiar a representação de *design rationale* (seção 3.3).

[4] http://www.ontoprise.com

3.1. A Linguagem F-logic

F-logic (ou *Frame logic*) é um formalismo lógico criado para descrever de forma declarativa a maioria dos aspectos estruturais de linguagens orientadas a objetos e de linguagens baseadas em *Frame*, usados na representação de conhecimento e de dados.

As características originais de F-logic incluem assinaturas, identidade de objetos, objetos complexos, métodos, classes e herança. Objetos são as primitivas básicas de F-logic. Assim, classes e métodos são também objetos. As informações sobre objetos tais como nome, classes e métodos são expressas por *F-atoms*. Os nomes de objetos e de variáveis são os elementos sintáticos básicos de F-logic e são chamados de *id-terms*. Os nomes de variáveis são sempre declarados usando os operadores lógicos “FORALL” e “EXISTS” para distingui-los dos nomes de objetos.

A descrição de uma ontologia em F-logic envolve a definição de um esquema de dados, de um conjunto de fatos, de regras e consultas sobre a base de fatos. O esquema de dados descreve o vocabulário da ontologia formado pelas classes de objetos, seus atributos e métodos. Os fatos representam instâncias dos conceitos definidos no vocabulário e podem ser considerados como a base de dados de um programa F-logic. Por fim, as regras definem o que pode ser derivado da base de fatos.

3.1.1. Esquema de Dados

O esquema de dados, ou vocabulário, é formado pela hierarquia de classes e assinaturas. A hierarquia de classes é expressa por relações de subclasse. As assinaturas definem que métodos são aplicáveis às instâncias das classes. O exemplo abaixo mostra o vocabulário de uma ontologia bem simples, que descreve algumas relações entre os membros de uma família.

```
/* Vocabulary */
mulher::pessoa.
homem::pessoa.
pessoa[pai=>homem].
pessoa[mae=>mulher].
pessoa[filho=>>homem].
pessoa[filha=>>mulher].
```

A relação de subclasse entre duas classes é expressa por *subclass-F-atoms* e denotada por um símbolo de dois pontos duplo (::). No exemplo acima, as duas primeiras definições expressam que as classes *homem* e *mulher* são subclasses da classe *pessoa*. As definições seguintes expressam os métodos aplicáveis às instâncias da classe "*pessoa"* (*signature-F-atoms*). Por exemplo, os métodos "*pai*" e "*mãe*" são aplicáveis aos objetos da classe "*pessoa*" e devem retornar, respectivamente, um objeto da classe "*homem*" e um objeto da classe "*mulher*". Métodos podem retornar um único valor (=>) ou podem retornar vários valores (=>>).

3.1.2. Fatos

Os fatos representam os objetos instanciados a partir dos conceitos definidos no vocabulário da ontologia. Objetos modelam entidades do mundo real e são internamente representados por identificadores de objetos que são independentes de suas propriedades. De acordo com o paradigma de orientação a objeto, as relações entre objetos são modeladas através da aplicação de métodos. Em F-logic, a aplicação de um método a um objeto é expressa por *data-F-atoms* que consistem de um objeto servidor, um método e um objeto resultante. O exemplo abaixo mostra uma pequena base de fatos formada por objetos instanciados com o vocabulário da ontologia definida no exemplo anterior.

```
/* Fatos */
manoel:homem.
pedro:homem.
maria:mulher.
carlos:homem[pai->manoel; mae->maria].
ana:mulher[pai->manoel; mae->maria].
joel:homem[pai->pedro; mae->maria].
maria[filho->>{carlos, joel}; filha->>ana].
```

Neste exemplo, temos a definição de vários objetos. O objeto "*carlos*" possui dois métodos (*pai* e *mãe*) que podem ter apenas um único valor associado (->). Já o objeto "*maria*" possui um método (*filho*) que pode ter vários valores associados (->>). Note que o relacionamento entre os objetos "*carlos*" e "*manoel*" é dado pela aplicação do método "pai". Ou seja, aplicando o método "*pai*" ao objeto "*carlos*" obtemos o objeto resultante "*manoel*".

A relação entre um objeto e a classe a qual ele pertence é expressa por *Isa-F-atoms* e denotada pelo símbolo de dois pontos (:). No exemplo, os três primeiros *Isa-F-atoms* expressam que Manoel e Pedro são objetos da classe "*homem*" e Maria é um objeto da classe "*mulher*".

As informações sobre um objeto podem ser agrupadas em moléculas (*F-molecules*), ao invés de serem apresentadas em várias definições individuais. Por exemplo, a seguinte molécula denota que Carlos é homem cujo pai é Manoel e Marcos e José são seus filhos.

```
carlos:homem[pai->manoel; filho->>{marcos, jose}].
```

Esta molécula também pode ser expressa por um conjunto de várias definições (*F-atoms*), como mostra o exemplo abaixo.

```
carlos:homem.
carlos[pai->manoel].
carlos[filho->>marcos].
carlos[filho->>jose].
```

3.1.3. Regras e Consultas

As regras derivam nova informação de uma dada base de objetos. Avaliando um conjunto de regras, novas relações entre objetos podem ser adicionadas à base de objetos como informações intencionais. Regras codificam informações genéricas da forma:

Se a pré-condição é satisfeita, então a conclusão é verdadeira.

A conclusão, *cabeçalho da regra*, e a pré-condição, chamada *corpo da regra*, são formadas por conjunções de moléculas. Sintaticamente, o cabeçalho da regra é separado do corpo da regra pelo símbolo "<-" e toda regra termina com um ponto. Toda variável no cabeçalho da regra deve também ocorrer em um *F-atom* positivo no corpo da regra. Fórmulas lógicas também podem ser negadas, usando o operador NOT. Considerando a base de fatos exemplificada no item anterior, as seguintes regras podem ser definidas:

```
/* Regras consistindo de um cabeçalho e corpo */
FORALL X,Y X[filho->>Y] <- Y:homem[pai->X].
FORALL X,Y X[filho->>Y] <- Y:homem[mae->X].
FORALL X,Y X[filha->>Y] <- Y:mulher[pai->X].
FORALL X,Y X[filha->>Y] <- Y:mulher[mae->X].
```

A primeira regra descreve que se X é pai de Y e Y é um homem, então Y é o filho de X. Uma relação similar ocorre para filhos e mães, para filhas e pais e filhas e mães.

Uma consulta pode ser considerada um tipo especial de regra com cabeçalho vazio. A seguinte consulta solicita todas as mulheres e seus filhos, cujo pai seja Manoel.

```
FORALL X,Y <- X:mulher[filho->>Y[pai->manoel]].
```

Esta consulta mostra a habilidade de F-logic aninhar aplicações de métodos. A mesma consulta poderia ser escrita como uma conjunção de expressões atômicas, como mostra o exemplo abaixo.

```
FORALL X,Y <- X:mulher AND X[filho->>Y] AND Y[pai->manoel].
```

3.2. O Vocabulário da Ontologia Kuaba

O vocabulário da ontologia Kuaba é composto por um conjunto de elementos (classes, propriedades, relações e restrições) que expressam o domínio de *design rationale*. Este vocabulário estende a estrutura de argumentação da notação IBIS, enriquecendo-a com a representação explícita das decisões tomadas durante o design e suas justificativas, e das relações entre os elementos de argumentação e os artefatos gerados. A extensão inclui também a integração da estrutura de argumentação com as descrições dos artefatos produzidos e com informações sobre a história de design do artefato (quando as decisões foram tomadas, quem tomou as decisões, que método de design foi utilizado, etc).

A Figura 5 ilustra os elementos do vocabulário definido pela ontologia Kuaba, usando uma notação gráfica no estilo da UML para auxiliar sua visualização. Note que esta representação orientada a objetos é usada apenas como uma representação gráfica do vocabulário. Algumas relações e restrições foram suprimidas por questões de legibilidade e serão apresentadas ao longo desta seção.

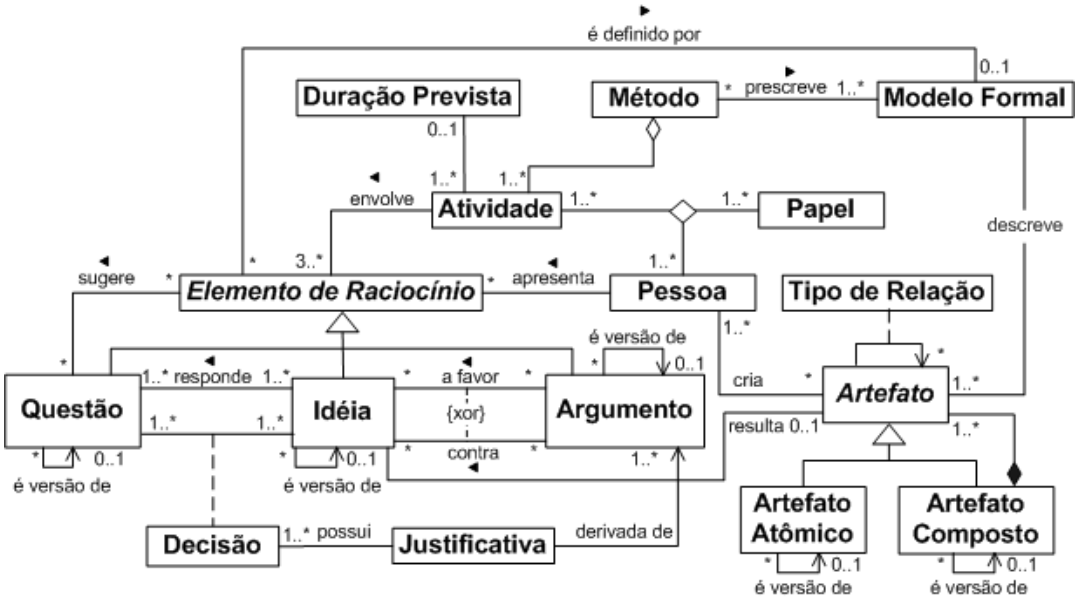

Figura 5 – Elementos do vocabulário da ontologia Kuaba

Brevemente descrito, o vocabulário da ontologia Kuaba representa os elementos de raciocínio usados pelos projetistas durante o design; as decisões tomadas por eles; as informações sobre os artefatos que resultam desse raciocínio; e as informações sobre a atividade de design realizada.

3.2.1. Elementos de Raciocínio

O design de um artefato de software envolve uma série de elementos de raciocínio. Estes elementos são usados pelo projetista na formulação de uma solução final para o problema de design que ele tem em mãos. De forma similar à notação IBIS, os elementos de raciocínio incluem as questões relacionadas ao design do artefato, as idéias de solução para essas questões e os argumentos contra ou a favor das idéias apresentadas. Cada elemento possui um conjunto de propriedades e relações que formam a estrutura do *rationale* desenvolvido pelo projetista durante o design.

A Figura 6 ilustra as propriedades dos elementos de raciocínio que podem estar envolvidos numa atividade de design e as relações entre eles.

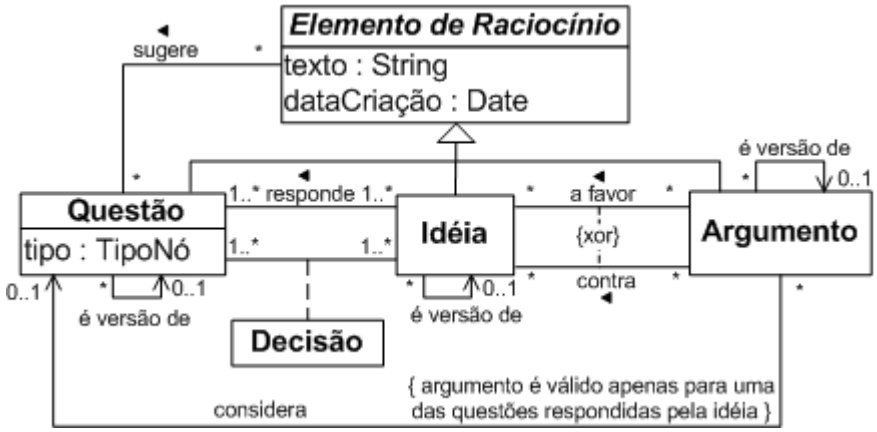

Figura 6 – Elementos de raciocínio e as relações entre eles

Uma questão representa um problema de design que precisa ser resolvido. Além do texto que indica qual problema deve ser tratado e a data de criação da questão, o elemento *Questão* possui também um atributo *tipo* pelo qual podemos definir se as idéias propostas como possíveis soluções para o problema devem ser consideradas mutuamente exclusivas ou não. Esta informação permite a definição de regras que podem ser usadas para sugerir decisões sobre a aceitação ou não das idéias de solução apresentadas. Apresentamos algumas destas regras na seção 3.3. *TipoNó* é uma enumeração que contém os possíveis valores para o atributo *tipo*: *AND*, *OR* ou *XOR*. O valor "*XOR*" indica que todas as idéias que respondem a questão são mutuamente exclusivas, ou seja, apenas uma destas idéias poderá ser aceita como solução para a questão. O valor "*AND*" indica que o projetista deve aceitar todas as idéias que respondem a questão ou rejeitar todas elas. Finalmente, o valor "*OR*" indica que várias idéias podem ser aceitas como solução para a questão.

Uma idéia representa uma possível solução, ou parte de uma solução de design, para o problema apresentado no elemento *Questão*. Na ontologia Kuaba preferimos usar o termo *Idéia* no lugar de *Alternativa* (termo geralmente usado em grande parte dos modelos existentes para *design rationale*), para possibilitar a representação de uma solução formada por várias idéias, uma vez que o termo *Alternativa* já embute em sua definição o caráter mutuamente exclusivo das idéias de solução apresentadas.

Um argumento representa uma razão contra ou a favor da adoção de uma idéia como uma solução para a questão que está sendo tratada. É através do

elemento *Argumento* que os projetistas podem registrar as experiências e o conhecimento que eles estão empregando no design do artefato.

Como podemos observar na Figura 6, uma idéia responde a uma ou mais questões e deve possuir pelo menos um argumento (contra ou a favor) associado a ela[5]. Uma instância de *Argumento* só pode participar de uma das relações possíveis com uma dada instância do elemento *Idéia*. Isto é representado no modelo pela restrição *{xor}*. Caso um argumento seja válido apenas para uma das questões respondidas pela idéia à qual ele se refere, este argumento deve estar associado também à questão respondida pela idéia através da relação *considera*.

Durante o processo de design novas questões podem ser sugeridas a partir de um elemento de raciocínio (questão, idéia ou argumento). Estas questões indicam que outros problemas de design precisam ser resolvidos para que o projetista possa chegar a uma decisão sobre uma solução para a questão inicial. A sugestão de novas questões é representada no modelo pelas relações *sugere* e *é sugerida por* (a relação inversa de *sugere*). Estas relações também podem ser usadas para representar a decomposição de uma questão em questões mais simples, para facilitar a exploração de idéias sobre as possíveis soluções para o design do artefato a ser construído.

Um outro tipo de relação ilustrada na Figura 6 é a relação "*é versão de*", definida para os elementos *Questão, Idéia* e *Argumento*. Esta relação permite representar que um elemento de raciocínio foi aproveitado da representação do *rationale* de um outro design. Esta representação pode ser de uma versão anterior do design, que está sendo usada para evoluir o artefato, ou de um design diferente que está sendo reusado em uma nova situação de design.

Abaixo apresentamos o trecho do vocabulário[6] da ontologia Kuaba, expresso em F-logic, que descreve os elementos de raciocínio ilustrados na Figura 6.

[5] A multiplicidade zero ou mais representada na Figura 6 pelo símbolo "*" é usada apenas para manter a representação UML correta, devido ao uso da restrição "{xor}".

[6] O vocabulário da ontologia foi definido em inglês para facilitar sua divulgação em âmbito internacional.

```
// CONCEPTS ----------
question::reasoning_element.
idea::reasoning_element.
argument::reasoning_element.
reasoning_element[hasText=>STRING; hasCreationDate=>STRING;
                  isInvolved=>activity; suggests=>>question;
                  isPresentedBy=>person; isDefinedBy=>formal_model].
question[hasType=>STRING; isAddressedBy=>>idea; hasDecision=>>decision;
        isSuggestedBy=>>reasoning_element; isVersionOf=>question].
idea[address=>>question; results=>artifact; hasArgument=>>argument;
     isConcludedBy=>>decision; isVersionOf=>idea].
argument[inFavorOf=>>idea;objectsTo=>>idea;
          considers=>question; isVersionOf=>argument].
                              ...
```

No trecho acima podemos observar a definição dos elementos "*Questão*", "*Idéia*" e "*Argumento*" como subclasses de "*Elemento de Raciocínio*", herdando todos os métodos definidos para este elemento.

3.2.2. Decisão

Durante a realização de uma atividade de design, uma pessoa ou um grupo pode decidir, com base nos argumentos apresentados, se uma idéia deve ou não ser aceita como uma solução para o design de um artefato. Na ontologia Kuaba, a aceitação ou a rejeição de uma idéia como uma solução para uma questão de design é registrada pelo elemento *Decisão*. Diferente de outras notações para *design rationale*, em nossa ontologia a aceitação ou a rejeição de uma idéia é representada como uma propriedade da relação entre os elementos *Questão* e *Idéia*, como ilustrado na Figura 7. Nós consideramos que a aceitação ou rejeição de uma idéia não é uma propriedade intrínseca do elemento *Idéia*, mas deve ser definida com relação a uma certa *Questão*, uma vez que a mesma idéia pode responder a mais de uma questão, e pode ser aceita para uma e ser rejeitada para outra. As propriedades do elemento *Decisão* e suas relações são ilustradas na figura abaixo.

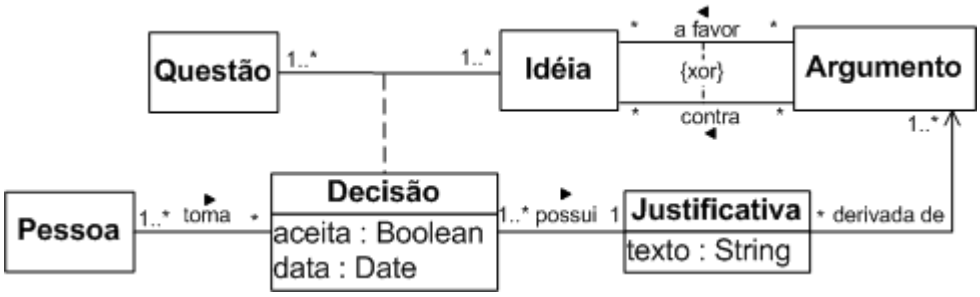

Figura 7 – Elemento Decisão e suas relações

Além da propriedade *aceita* que registra a aceitação ou a rejeição de uma idéia, o elemento *Decisão* também possui a propriedade *data* que registra quando a decisão foi tomada pelo projetista. A partir dos valores atribuídos a esta propriedade, será possível obter um histórico sobre a seqüência de decisões tomadas pelo projetista durante o design.

Como podemos observar na Figura 7, uma decisão deve possuir uma justificativa sobre o porque da aceitação ou da rejeição de uma idéia como solução para uma determinada questão. Esta justificativa é sempre derivada de um ou mais argumentos apresentados durante a atividade de design. Embora uma justificativa tipicamente inclua os argumentos contra e a favor das idéias consideradas, decidimos representar justificativa e argumento como dois elementos distintos em nossa ontologia, a fim de permitir a definição e a representação da justificativa final de uma decisão, derivada dos argumentos apresentados durante o design. A justificativa pode, por exemplo, explicitar a forma como os argumentos a favor e contra a idéia aceita foram consolidados.

O trecho do vocabulário abaixo mostra a definição do elemento *Decisão* e suas relações.

```
// CONCEPTS ----------------
question[hasType=>STRING; isAddressedBy=>>idea; hasDecision=>>decision;
        isSuggestedBy=>>reasoning_element; isVersionOf=>question].
idea[address=>>question; results=>artifact;
    isConcludedBy=>>decision; isVersionOf=>idea].
decision[isAccepted=>BOOLEAN, hasDate=>STRING; isMadeBy=>>person,
        hasJustification=>justification; concludes=>idea].
justification[hasText=>STRING; isDerivedOf=>>argument].
            ...
```

Pela definição acima podemos notar que a associação do elemento *Decisão* aos elementos *Questão* e *Idéia* é feita pelas relações "*tem decisão*" e "*conclui*",

representadas em F-logic como métodos dos objetos *questão* e *decisão*, respectivamente. De acordo com esta especificação, várias decisões podem estar associadas a uma questão, mas cada decisão só pode estar associada a uma única idéia. Isto permite representar as diferentes decisões tomadas sobre as idéias de solução propostas para uma determinada questão de design.

3.2.3. Artefatos

A integração do *design rationale* com a descrição do artefato projetado, ou de seus componentes, torna a representação de *rationale* útil na evolução deste artefato, ou em seu reuso na criação de novos artefatos. Esta integração nos permite obter, a partir de um artefato, as decisões de design que lhe deram origem, e conhecer todo o raciocínio desenvolvido pelo projetista para chegar ao seu design final. Na ontologia Kuaba, esta integração é representada pela relação "*resulta*" entre o elemento *Idéia* e o elemento *Artefato*, como ilustra a Figura 8.

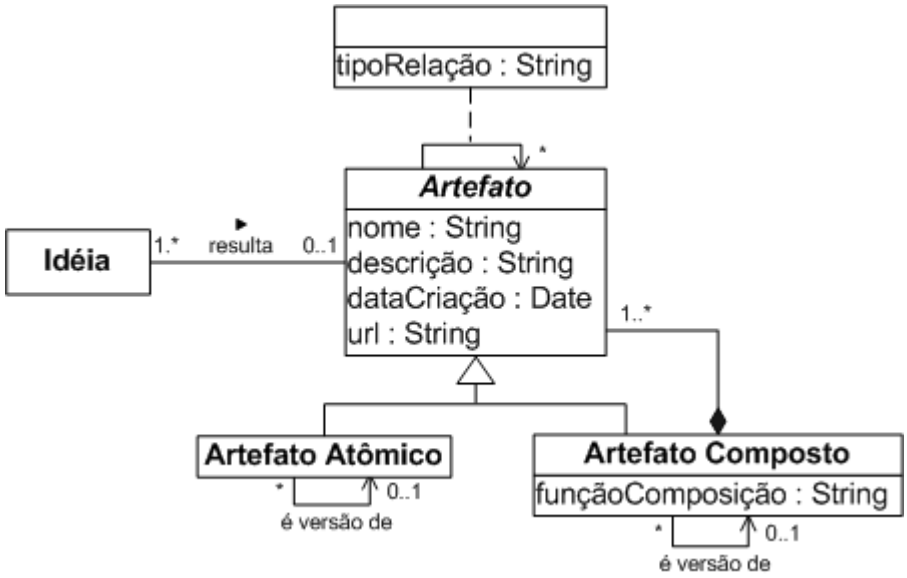

Figura 8 – Elemento Artefato e suas relações

Um artefato corresponde a uma solução de design final, formada pelo conjunto de idéias aceitas em uma representação de *design rationale*. Portanto, em uma representação de *design rationale* todo artefato deve estar associado a pelo menos uma idéia e esta deve ter sido aceita. Ou seja, um artefato não pode estar associado a uma idéia que foi rejeitada durante o processo de design.

No vocabulário da ontologia Kuaba, artefatos são representados por dois elementos, *Artefato Atômico* e *Artefato Composto*, dependendo da definição do

artefato no modelo formal do método de design utilizado. Por exemplo, no modelo formal da UML uma classe pode ser vista como uma composição de atributos, portanto, um item de informação modelado como uma classe pode ser representado como um artefato composto.

Além do *nome*, da *descrição* e da *data de criação*, o elemento *Artefato* contém uma propriedade *url* que permite registrar a localização do artefato construído. A propriedade *funcaoComposicao* do elemento *Artefato Composto* representa a função utilizada para compor o artefato. O objetivo desta função é permitir a construção de novos artefatos a partir de um artefato conhecido. O valor dessa propriedade pode ser expresso tanto em uma linguagem específica de ferramenta, para permitir a geração automática de novos artefatos, quanto em linguagem natural, que por sua vez pode ser usada por um projetista para guiar a construção de um artefato composto.

Um outro aspecto considerado no vocabulário da ontologia Kuaba é a representação da relação entre artefatos. A propriedade *tipoRelação* permite especificar a natureza desta relação que pode ser, por exemplo, a relação de inclusão entre dois casos de uso.

Da mesma forma que os elementos de raciocínio, um artefato pode ser uma versão de outro artefato, seja para registrar a evolução do artefato ao longo do processo de design, seja para registrar que o artefato no design atual é uma versão de um artefato projetado em algum outro design. Isto é representado no modelo pelas relações do tipo "*é versão de*".

Abaixo mostramos a definição dos elementos para a representação de artefatos e suas relações.

```
// CONCEPTS ----------------
atomic_artifact::artifact.
composite_artifact::artifact.
artifact[hasName=>STRING; hasDescription=>STRING; hasUrl=>STRING,
        hasCreationDate=>STRING; isResulted=>>idea;
        isCreatedBy=>>person; isDescribedBy=>formal_model].
atomic_artifact[isVersionOf=>atomic_artifact].
composite_artifact[compositionFunction=>STRING; compositionOf=>>artifact;
                    isVersionOf=>composite_artifact].
idea[address=>>question; results=>artifact; hasArgument=>>argument;
     isConcludedBy=>>decision; isVersionOf=>idea].
            ...
```

3.2.4. A Atividade de Design

Além dos elementos de raciocínio e das relações que permitem registrar o *design rationale* de um artefato, o vocabulário da ontologia Kuaba possui também elementos que permitem registrar o método de design utilizado pelos projetistas, as atividades de design realizadas de acordo com este método, as pessoas envolvidas no design e os papéis que elas exercem em cada atividade. O registro dessas informações é importante para facilitar o processamento e a interpretação do *rationale* representado, pois nos fornece parte do contexto no qual o artefato foi projetado. A Figura 9 ilustra estes elementos e suas relações.

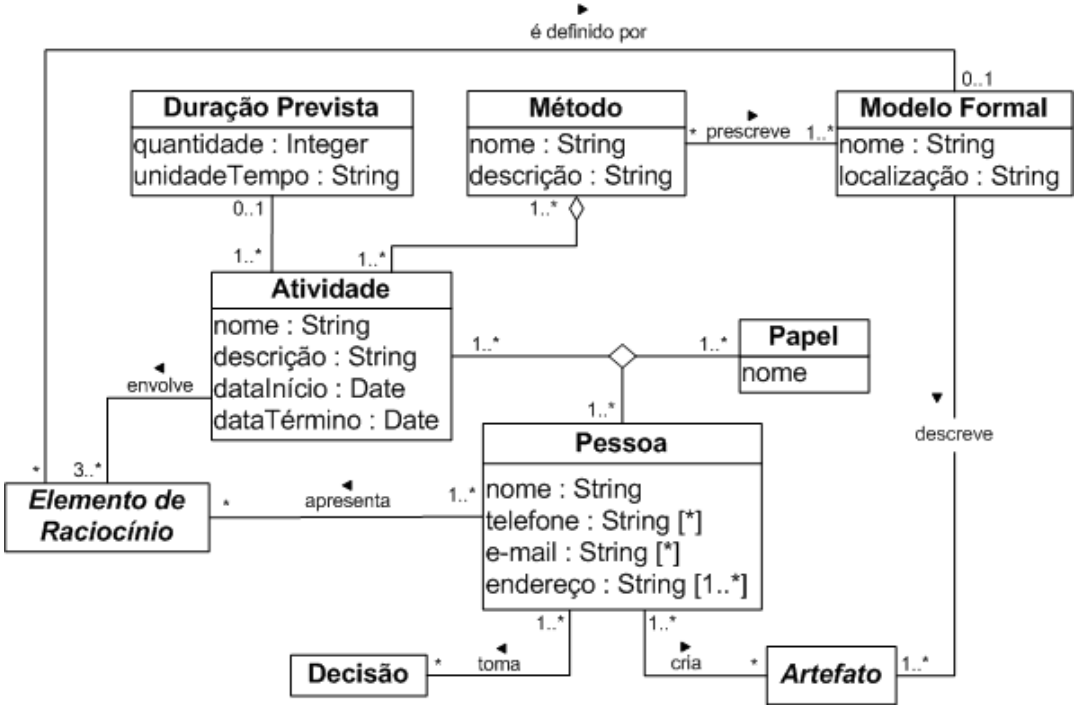

Figura 9 – Elementos para representar informações sobre atividade de design

Como podemos observar, uma pessoa pode apresentar um ou mais elementos de raciocínio, tomar decisões sobre as idéias de solução consideradas e construir o artefato final, de acordo com o modelo formal que descreve o artefato. Este modelo formal é prescrito pelo método de design utilizado pelo projetista, ambos representados pelos elementos *Modelo Formal* e *Método*, ilustrados na Figura 9.

O registro do método de design utilizado e das atividades realizadas de acordo com este método é um aspecto importante relacionado ao *design rationale* de um artefato. Este registro em uma representação de *design rationale* ajuda a compreender melhor o raciocínio empregado pelos projetistas durante o design,

uma vez que grande parte das questões e idéias de solução propostas é definida com base no modelo formal prescrito por este método. Além disso, a informação sobre este modelo formal ajuda a empregar um formalismo maior na representação do *design rationale*, melhorando a verificação da consistência do *rationale* gerado e o seu processamento por um ambiente computacional.

O trecho do vocabulário da ontologia Kuaba abaixo mostra a especificação dos elementos definidos para representar as informações de uma atividade de design.

```
// CONCEPTS ----------------
method[hasName=>STRING; hasDescription=>STRING;
        aggregates=>>activity; prescribes=>>formal_model].
formal_model[hasName=>STRING; hasLocalization=>STRING;
              describes=>>artifact; isPrescribedBy=>>method].
activity[hasName=>STRING; hasDescription=>STRING;
        hasStartDate=>STRING; hasFinishDate=>STRING;
        hasExpectedDuration=>expected_duration; requires=>>role;
        involves=>>reasoning_element; isExecutedBy=>>person].
expected_duration[hasAmount=>integer; hasUnitTime=>STRING].
person[hasName=>STRING; presents=>>reasoning_element;
        creates=>>artifact; makes=>>decision; hasE_mail=>>STRING;
        hasAddress=>STRING; hasTelephone=>>STRING;
        performs=>>role; executes=>>activity].
role[hasName=>STRING; isPerformedBy=>>person;
    isRequiredBy=>>activity].
          ...
```

Algumas dessas informações nos permitem também obter alguns dados importantes sobre a história do design realizado. Por exemplo, podemos descobrir qual método de design foi utilizado e quais atividades foram necessárias para construir o artefato. Com base nos valores atribuídos às propriedades do elemento *Atividade*, podemos saber também se as atividades foram realizadas no prazo previsto ou se ocorreu algum atraso. Além disso, podemos conhecer as pessoas que participaram do design e os perfis profissionais que foram necessários neste design. Por fim, podemos conhecer os responsáveis pelas decisões tomadas e os criadores do artefato final. Estas informações podem apoiar algumas decisões sobre a manutenção do artefato ou sobre um novo design com características semelhantes ao design realizado.

Como o vocabulário da ontologia Kuaba pode ser usado para gerar representações de *design rationale* para diferentes designs, é necessário definir uma forma de distinguir estas representações. Uma forma possível seria definir um elemento "*Design*" na ontologia Kuaba e relacionar a ele todos os outros elementos definidos no vocabulário. Este elemento teria uma única instância em cada representação gerada, representando o design que está sendo realizado. Por questões de simplicidade, resolvemos não incluir este elemento nesta versão da ontologia Kuaba e armazenar as diferentes representações em arquivos distintos. Além de arquivos diferentes, usamos também o mecanismo de *namespace* definido pela linguagem usada para processar as representações de *design rationale*, como veremos no capítulo 6.

3.3. Regras da Ontologia Kuaba

As regras definidas com o vocabulário da ontologia Kuaba têm como objetivo permitir a realização de inferências e validações sobre o *design rationale* registrado durante o design.

O primeiro conjunto de regras, mostrado abaixo, especifica relações inversas para algumas das relações definidas no vocabulário da ontologia Kuaba. Por exemplo, a relação "*é respondida por*" é definida como a inversa da relação "*responde*", definida entre os elementos *Questão* e *Idéia*.

```
// RULES (INVERSE RELATIONS) ----------

FORALL X,Y X[address->>Y]   <-> Y[isAddressedBy->>X].
FORALL X,Y X[concludes->Y] <-> Y[isConcludedBy->>X].
FORALL X,Y X[prescribes->>Y] <-> Y[isPrescribedBy->>X].
FORALL X,Y X[suggests->>Y] <-> Y[isSuggestedBy->>X].
FORALL X,Y X[performs->>Y] <-> Y[isPerformedBy->>X].
FORALL X,Y X[describes->>Y] <-> Y[isDescribedBy->>X].
FORALL X,Y X[presents->>Y] <-> Y[isPresentedBy->>X].
FORALL X,Y X[executes->>Y] <-> Y[isExecutedBy->>X].
FORALL X,Y X[requires->>Y] <-> Y[isRequiredBy->>X].
FORALL X,Y X[involves->>Y] <-> Y[isInvolvedBy->X].
FORALL X,Y X[creates->>Y] <-> Y[isCreatedBy->>X].
FORALL X,Y X[results->Y] <-> Y[isResulted->>X].
FORALL X,Y X[makes->>Y] <-> Y[isMadeBy->>X].
                                ...
```

Estas regras permitem especificar relações entre classes em ambas direções. Geralmente, estas regras são usadas pelas máquinas de inferência para derivar informações quando as relações estão representadas apenas em um sentido. Por

exemplo, com base no fato “idéia A responde questão X”, o sistema pode usar a regra definida na ontologia para inferir que a questão X é respondida pela idéia A, sem que esta última relação tenha sido explicitamente declarada pelo projetista durante a representação do *rationale*. Este tipo de inferência é muito interessante, pois garante que uma consulta formulada com qualquer uma dessas relações terá uma resposta consistente com o *rationale* registrado.

Neste trabalho, definimos também algumas regras de inferência que permitem derivar novas informações da base de fatos e manter a consistência do *rationale* que está sendo representado durante o design. Por exemplo, a regra abaixo permite inferir o papel exercido por uma pessoa a partir das informações sobre a atividade que esta pessoa executa e do papel requerido por esta atividade.

```
// AXIOM ----------------
FORALL P, A, R  P[performs->>R]
        <-      P:person[executes->>A]
        AND   A:activity[requires->>R:role].
```

Com base nesta regra, se uma pessoa executa uma atividade de design e esta atividade requer um papel específico, então esta pessoa exerce este papel.

Algumas das regras de inferência definidas para a ontologia Kuaba são usadas pelo ambiente de design proposto nesta tese para sugerir decisões sobre a aceitação ou rejeição das idéias de solução propostas durante o design. Estas regras são definidas com base no atributo *tipo* do elemento *Questão*. Como vimos anteriormente, os valores para este atributo (“*AND*”, ”*OR*” ou “*XOR*”) definem se as idéias de solução propostas para uma questão de design particular são mutuamente exclusivas ou não. A seguinte regra é definida para sugerir decisões para questões do tipo “*XOR*”.

```
// AXIOM ----------------
FORALL Q,I1,I2,D1,D2  D2[isAccepted->"false"]
        <- Q:question[hasType->"XOR"; isAddressedBy->>{I1,I2};
                      hasDecision->>{D1,D2}]
        AND D1:decision[isAccepted->"true"; concludes->I1:idea]
        AND D2:decision[concludes->Idea2:idea]
        AND NOT (I1 = I2).
```

De acordo com esta regra, se uma idéia associada a uma questão do tipo “*XOR*” é aceita pelo projetista, então todas as outras idéias de solução propostas para esta questão podem ser rejeitadas. Baseado nesta regra, o ambiente de design pode sugerir ao projetista a rejeição dessas idéias. No próximo capítulo

comentamos a aplicação desta regra nos exemplos de representações de *design rationale* abordados.

Um outro exemplo de regra de inferência é mostrado abaixo. Com base nesta regra, o ambiente de design pode sugerir a rejeição de idéias de solução propostas para questões do tipo "*AND*".

```
// AXIOM ----------
FORALL Q,I1,I2,D1,D2  D2[isAccepted->"false"]
        <- Q:question[hasType->"AND"; isAddressedBy->>{I1,I2};
                      hasDecision->>{D1,D2}]
        AND D1:decision[isAccepted->"false"; concludes->I1:idea]
        AND D2:decision[concludes->Idea2:idea]
        AND NOT (I1 = I2).
```

Na regra acima, se uma idéia de solução associada a uma questão do tipo "*AND*" é rejeitada pelo projetista, então todas as outras idéias de solução propostas para esta questão também devem ser rejeitadas para manter a consistência do *rationale* gerado.

O conjunto de regras definidas para a ontologia Kuaba também inclui regras específicas para a validação do *design rationale* representado. Validar *rationale* envolve verificar se o *rationale* está estruturalmente completo e se existe alguma discrepância com relação às decisões tomadas. Esta validação é importante, uma vez que um *design rationale* completo pode indicar que os projetistas foram capazes de justificar explicitamente as razões para suas decisões de design. As principais validações para o *design rationale* que estamos tratando neste trabalho são:

- Verificar se existem idéias sem argumentos associados;
- Verificar se existem questões não resolvidas (sem decisões associadas);
- Verificar se existem idéias não concluídas (sem decisões associadas);
- Verificar se existem decisões não justificadas;
- Verificar se existem justificativas que não são derivadas de argumentos;
- Verificar se existem artefatos associados a idéias rejeitadas.

As validações relacionadas acima podem ser realizadas por meio de consultas (regras sem cabeçalho) à base de fatos formada pelo *rationale* que está sendo representado, como mostra o trecho de código abaixo. Observe que a terceira consulta faz uso da relação inversa "*isConcludedBy*" definida entre os elementos *Idéia* e *Decisão*.

```
// Queries ----------
FORALL X,Y  <- X:idea AND NOT (X[hasArgument->>Y:argument]).
FORALL X,Y  <- X:question AND NOT (X[hasDecision->>Y:decision]).
FORALL X,Y  <- X:idea AND NOT (X[isConcludedBy->>Y:decision]).
FORALL X,Y  <- X:decision ANT NOT (X[hasJustification->Y:justification]).
FORALL X,Y  <- X:justification AND NOT (X[isDerivedOf->>Y:argument]).
FORALL X,Y, Z <- X:artifact[isResulted->>Y] AND
                 Y:idea[isConcludedBy->>Z] AND
                 Z:decision[isAccepted->'false'].
                          ...
```