UNIVERSIDADE FEDERAL FLUMINENSE

INSTITUTO DE COMPUTAÇÃO

CIÊNCIA DA COMPUTAÇÃO

Maria Gabriela Corrêa Netto de Alcântara

Mariana de Sousa Marinho

UMA FERRAMENTA PARA MODELAGEM DE NORMAS

Niterói

2010

Maria Gabriela Corrêa Netto de Alcântara
Mariana de Sousa Marinho

UMA FERRAMENTA PARA MODELAGEM DE NORMAS

Trabalho de conclusão de curso apresentado à Universidade Federal Fluminense como requisito parcial para obtenção do título de graduado em Ciência da Computação

Professora Orientadora: Viviane Torres da Silva

Niterói
2010

Maria Gabriela Corrêa Netto de Alcântara
Mariana de Sousa Marinho

UMA FERRAMENTA PARA MODELAGEM DE NORMAS

Trabalho de conclusão de curso apresentado à Universidade Federal Fluminense como requisito parcial para obtenção do título de graduado em Ciência da Computação

Aprovada em Dezembro de 2010.

BANCA EXAMINADORA

Prof[a]. Viviane Torres da Silva – Orientadora
UFF

Prof. Leonardo Murta
UFF

Prof. Christiano Braga
UFF

Niterói
2010

AGRADECIMENTOS MARIANA

A Deus por sempre iluminar o meu caminho.

A minha mãe que é a minha melhor amiga e o meu porto seguro.

A toda a minha família, em especial a minha avó, avô, tias e tio pelo suporte e apoio de sempre. Vocês são a minha base.

As minha amigas Vanessa e a Luana pela paciência desde o vestibular até a monografia. Amo poder contar com vocês.

A todos os meus amigos, que são a família que Deus me permitiu escolher. Obrigada pelos conselhos, conversas, risadas, saídas, abraços e apoio.

Aos professores da UFF pelo conhecimento adquirido ao longo do curso de graduação.

Ao professor José Raphael pelas conversas e orientação nos últimos períodos do curso.

Aos meus amigos da UFF, em especial a Lays, pelas chopadas, horas no laboratório, ajuda na monografia e conselhos. Vocês estarão no meu coração pra sempre.

A Marister pelos cafés, abraços e apoio. Obrigada!

A minha amiga Gabi por quatro anos de provas, estudos, conselhos, festas, torpedos e principalmente por me complementar tão bem nesse trabalho. Seria realmente impossível sem você!

A professora Viviane, pela orientação neste trabalho.

AGRADECIMENTOS MARIA GABRIELA

A Deus por me guiar nas minhas escolhas.

A minha mãe, meu irmão, minhas irmãs e minha sobrinha, pelo amor incondicional. Afinal nós somos UM.

Ao meu pai pelo amor e por me ensinar que “A vitória é certa”.

Ao meu “paidrasto”, o segundo pai que eu escolhi.

Ao meu amor, a pessoa que escolhi para estar ao meu lado em um momento tão especial como este, obrigada por acreditar em mim e por me dedicar tanto amor e carinho.

A toda a minha família, obrigada por tudo.

As minha amigas Joelma, Tayná e Lays, e ao meu amigo Vitor. Pessoas que posso contar a qualquer momento, AMIGOS!

A “galere” obrigada pelos dias repletos de conversas, zoações, risadas, saídas, pizzas e apoio. Aprendi muito com vocês!

A todos os amigos, obrigada pela paciência e carinho!

Aos professores da UFF pelo conhecimento e dedicação .

Ao professor do coração, José Raphael, pela amizade.

A Marister por oferecer seu colo e abraço de mãe.

A minha amiga Mari pela amizade verdadeira e sincera, vou sentir saudades dos milhares de torpedos, de assistir todas as aulas juntas, passar horas e horas conversando...Esses 4 anos seriam impossíveis sem você!!!

A professora Viviane, pela orientação neste trabalho.

RESUMO

O objetivo deste trabalho é construir uma ferramenta baseada na linguagem NormML, para modelagem de normas em sistemas multiagentes e para verificação de conflitos entre normas, respeitando as regras de boa formação previamente definidas.

Na modelagem, fazemos o uso de um meta-modelo que define os elementos da linguagem de normas utilizados na construção dos modelos. A validação dos mesmos, de acordo com as regras de boa formação, é feita através de operações implementadas na linguagem OCL (Object Constraint Language).

A ferramenta EOS (Eye OCL Software) é utilizada na execução das operações descritas em OCL. Essa possibilita a descrição de diagramas de classe e objeto, a implementação de operações e a execução das mesmas sobre o diagrama de objeto. No nosso contexto, o diagrama de classes foi utilizado para implementar o meta-modelo, o diagrama de objetos para descrever as normas e as operações OCL para descrever as regras de boa formação e as de verificação de conflitos.

Palavras-chave: Normas, Sistemas Multiagentes, OCL, NormML, Conflitos entre Normas.

ABSTRACT

The main goal of this work is to build a tool, based on the language called NormML for modeling norms in Multi-Agent Systems and for the verification of conflicts between norms, respecting the well formed rules previously established.

In the modeling we use a metamodel that defines the main elements of the normative language used to create the models. The validation of these models, according to the well formed rules and their properties, is done through operations implemented in the OCL language (Object Constraint Language).

The EOS tool (Eye Software OCL) is used to assist in the implementation of the operations described in OCL. The tool allows the description of class and object diagrams, the implementation of operations and their execution over the object diagram. In our context, the class diagram was used to implement the metamodel, the object diagram to describe the norms and the operations in OCL to describe the well formed rules and the rules for the verification of conflicts.

Keywords: Norms, Multi-Agent Systems, OCL, NormML, Conflicts between Norms.

LISTA DE FIGURAS E TABELAS

LISTA DE SIGLAS

NormML – *Normative Modeling Language*

OMG – *Object Management Group*

SMA – Sistemas MultiAgentes

UML – *Unified Modeling Language*

OCL – *Object Constraint Language*

EOS – *Eye OCL Software*

MOF – *Meta-Object Facility*

RBAC – *Roles-Based Access Control*

SUMÁRIO

CAPÍTULO 1: INTRODUÇÃO

Sistemas Multiagentes (SMA) são sistemas de computador compostos por agentes, ou seja, pedaços de software, que podem interagir com o propósito de atingir os objetivos de todo o sistema. Agentes de software são entidades autônomas, adaptativas, interativas e orientadas a objetivos. Os agentes são capazes de executar (seus planos de ação) sem a necessidade da intervenção humana ou de outros agentes, caracterizando assim sua propriedade autônoma [Figueiredo, K. and Silva, V. T. (2010)]. O comportamento dos agentes é orientado a objetivos, pois sua execução se dá pela seleção de objetivos e execução de planos capazes de alcançar estes objetivos. Durante a execução dos planos, os agentes podem adaptar seu comportamento às mudanças no ambiente no qual estão inseridos criando, por exemplo, novos planos, novos objetivos e adquirindo novos conhecimentos sobre o ambiente e sobre outros agentes. Os agentes são capazes de interagir entre eles trocando mensagens com o objetivo de cooperar, colaborar ou competir. Os agentes também são capazes de tratar ou ignorar as mensagens recebidas de acordo com os seus objetivos.

Os agentes destes sistemas são agentes heterogêneos e tipicamente implementados por diferentes desenvolvedores. Com objetivo de lidar com a diversidade de implementações dos agentes e os possíveis problemas que os diferentes comportamentos podem causar ao sistema, os sistemas multiagentes definem um conjunto de normas (ou leis) que devem ser seguidas pelos agentes. As normas são utilizadas para regular o comportamento de agentes em um SMA através da descrição de obrigações, proibições e permissões para estes agentes. Para que as normas possam fazer parte do sistema, é válido destacar que a definição das mesmas deve ser feita durante a especificação do sistema. Baseado no contexto acima descrito, NormML [Figueiredo, K. and Silva, V. T. (2010)] é uma linguagem de modelagem para a especificação de normas em SMA baseada na linguagem UML.

Neste trabalho temos como objetivo abordar a modelagem de normas em sistemas multiagentes, utilizando assim, a linguagem de modelagem de normas

NormML. O principal intuito é construir uma ferramenta que faça não apenas a modelagem das normas mas também a validação do modelo onde estas estão descritas. A validação de um modelo compreende, no contexto deste trabalho, a verificação da conformidade do mesmo em relação as regras de boa formação assim como a verificação da existência de conflitos entre as normas. As regras de boa formação especificam regras que os modelos instâncias do meta-modelo que descreve a linguagem NormML devem seguir para serem considerados modelos bem formados nesta linguagem. Já os conflitos entre as normas ocorrem quando (i) uma norma proíbe e outra permite um agente de executar uma ação, no mesmo intervalo de tempo e em um mesmo contexto de execução; (ii) uma norma proíbe e outra obriga um agente a executar uma ação, no mesmo intervalo de tempo e em um mesmo contexto de execução; e (iii) uma norma permite e outra obriga um agente a executar uma ação, em um mesmo contexto de execução porém em intervalos de tempo que não se interceptam.

Para construir o modelo, primeiramente o usuário modela as normas e todos os elementos utilizando a interface provida pela ferramenta. Após esta etapa, é gerado um arquivo, no formato XMI, que após sofrer duas transformações, gera um diagrama de objetos que juntamente com o diagrama de classes que representa o meta-modelo de NormML servirá de entrada para a ferramenta EOS (Eye OCL Software) [Clavel, M., Egea, M. and Dios, M., (2008)] que será utilizada para realizar as validações necessárias seguindo as regras de boa formação e as regras de verificação de conflitos. .Para descrição de todas as regras, fizemos o uso da linguagem OCL (Object Constraint Language), conceituada como notação da UML utilizada para definir restrições sobre objetos.

O EOS é um componente Java para avaliação de OCL que permite definir um diagrama de classes, um diagrama de objetos instâncias destas classes e um conjunto de operações descritas em OCL que são executadas sobre os objetos, neste nosso caso, sobre as normas.

Este trabalho está organizado nos seguintes capítulos. Capítulo 2 contendo a fundamentação teórica necessária para compreender o objetivo deste trabalho, onde podemos entender os conceitos de meta-modelo, da linguagem OCL, agentes e normas. Capítulo 3 consiste em apresentar a linguagem NormML, o seu meta-modelo, a geração de normas com base nos conceitos anteriormente descritos, bem

como o detalhamento das regras de boa formação e conflitos. Capítulo 4 apresenta a ferramenta construída neste trabalho, bem como os componentes utilizados detalhando assim, todo o seu fluxo de funcionamento. Capítulo 5 descreve de forma breve um quadro comparativo entre essa e outras ferramentas de modelagem de normas. Capítulo 6 contém a conclusão deste trabalho e possíveis melhorias para o mesmo.

1.1 CONTRIBUIÇÕES

É importante destacar, de maneira geral, que tivemos como contribuição para este trabalho:

- A implementação de todas as operações OCL que representam as regras de boa formação e as regras de verificação de conflitos;
- A representação do meta-modelo de NormML como um diagrama de classes seguindo as características da ferramenta EOS;
- A implementação de um transformador que recebe um arquivo XMI descrito seguindo o meta-modelo de UML e o transforma em um diagrama de classes seguindo as características da ferramenta EOS;
- A implementação da ferramenta para a modelagem de normas em si.

É preciso ressaltar que tanto a elaboração das regras, quanto a descrição do meta-modelo, foram definidos em [Figueiredo, K. and Silva, V. T. (2010)].

CAPÍTULO 2: FUNDAMENTAÇÃO TEÓRICA

2.1 META-MODELO

O meta-modelo de uma linguagem de modelagem fornece o vocabulário (conceitos e relações) que será utilizado na criação dos modelos, bem como uma notação gráfica para descrevê-los como diagramas. O meta-modelo pode ser representado como um diagrama cujos elementos formalizam os conceitos e relações fornecidos pela linguagem de modelagem e cujas restrições (regras de boa formação ou invariantes) especificam regras que os modelos instâncias do meta-modelo devem seguir para serem considerados modelos bem formados.

Muitas vezes, o meta-modelo da linguagem de modelagem também fornece um vocabulário para descrever instâncias do modelo, juntamente com a notação gráfica para descrevê-los. No caso da UML (Unified Modeling Language) [UML - Object Management Group - OMG], por exemplo, tanto o vocabulário que deverá ser utilizado nos diagramas de classe quanto nos diagramas de objetos está presente no meta-modelo de UML.

O Object Management Group (OMG) [OMG - Object Management Group] propôs a arquitetura clássica de quatro camadas chamada MOF (Meta-Object Facility) com o objetivo de definir uma linguagem abstrata para a especificação de meta-modelos. As camadas propostas por esta arquitetura e exemplificadas na figura 2.1 são:

- A camada de meta-metamodelo (M3) contém os elementos do meta-metamodelo que são utilizados para definição dos meta-modelos. A própria MOF define um conjunto de elementos que devem ser utilizados para a instanciação de meta-modelos. Sendo assim, MOF é um exemplo de meta-metamodelo.

- A camada de metamodelagem (M2) contém os elementos do meta-modelo que são usados para definir modelos. O meta-modelo é uma instância do meta-

metamodelo, significando assim, que cada elemento do primeiro é uma instância de um elemento do segundo. O meta-modelo de UML, que é uma instância de MOF.

- A camada M1 contém elementos de modelos utilizados para modelar diferentes domínios de problemas. O modelo de usuário é uma instância do meta-modelo de UML e contém os elementos do modelo e os snapshots das instâncias desses elementos do modelo.

- A camada inferior (M0) contém as instâncias de tempo de execução dos elementos definidos no modelo.

Conforme exemplificado na figura 2.1 a estrutura de quatro camadas pode ser implementada de forma recursiva, ou seja, o que é um meta-modelo em um caso pode ser considerado um modelo em outro. Além disso, todo modelo é instância de um meta-modelo no qual este é baseado.

No exemplo mostrado na Figura 2.1, a camada de meta-metamodelo M3 define o elemento Class. Instâncias deste elemento são utilizadas na camada inferior de meta-modelagem (Class, Attribute, Instance). Essa mesma correspondência, ocorre entre as camadas M2 e M1, onde Wagon é um elemento instância da meta-classe *Class* definida no meta-modelo do nível M2. Na camada M0 representamos uma instância do elemento Wagon definido em M1. Sendo assim, para cada elemento de um modelo em um determinado nível M_x é instancia de um elemento definido no nível M_{x+1}.

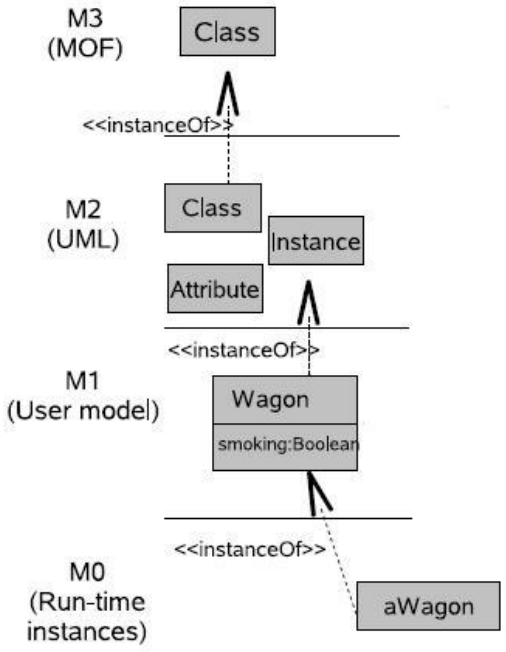

Figura 2.1 A Arquitetura MOF [OMG - Object Management Group]

2.2 OCL

Geralmente, os diagramas descritos utilizando a linguagem de modelagem UML não são suficientemente detalhados para fornecer todos os aspectos relevantes do sistema. Existe então a necessidade de descrever restrições adicionais aos elementos presentes nestes modelos com o objetivo de definir tais aspectos [OMG (2003)]. Estas restrições são, usualmente, descritas em linguagem natural, o que pode vir a causar problemas de ambiguidade. A possível solução para este fato é usar uma linguagem livre de ambiguidades onde tais restrições pudessem ser descritas. Linguagens com estas características são chamadas de linguagens formais. Porém, as linguagens formais tradicionais exigem do usuário uma formação matemática forte, o que pode não ser o caso dos projetistas e desenvolvedores de sistemas.

Impedir a modelagem de ciclos em heranças de classes representadas em diagramas de classes é um exemplo de aplicação de tais restrições a estes diagramas. Por exemplo, não deveríamos ser capazes de modelar a seguinte hierarquia: classe A herda da classe B, que por sua vez herda da classe C que herda da classe A, pois esta hierarquia caracteriza um ciclo. Utilizando apenas as notações disponíveis para descrever o meta-modelo de UML não é possível garantir que modelos que sejam instâncias do meta-modelo de UML não contenham estes ciclos.

Sendo assim, o OMG propôs a linguagem para descrição de restrições para objetos, chamada OCL [OCL - Object Management Group – OMG (2003)]. OCL é uma notação utilizada para descrever restrições sobre objetos e é aplicada a um modelo orientado a objetos ou a parte de um sistema modelado. Utilizando a linguagem OCL é possível então descrever restrições ou regras de boa formação garantindo que todo modelo instância do meta-modelo de UML será um modelo bem-formado somente se respeitar estas restrições. Seguindo o exemplo mencionado anteriormente, para garantir que ciclos de herança não ocorram em diagrama de classes, é necessário associar uma restrição ao meta-modelo de UML.

Embora OCL seja uma linguagem formal, ela é de fácil escrita e leitura além de possuir uma sintaxe bem definida e semântica precisa. OCL não é uma linguagem de programação, mas sim de modelagem. Desta forma, expressões OCL não são diretamente executáveis.

Uma característica importante da linguagem citada é o fato das expressões especificadas não possuírem efeitos colaterais, não causando assim, mudança no estado do objeto, modificações nos modelos e alterações no fluxo de controle. Além disso, as expressões OCL têm avaliação instantânea, ou seja, quando uma expressão OCL é avaliada ela simplesmente retorna um valor [Martins, E. (2003)].

Ser fortemente “tipada” é outro ponto fundamental na definição da linguagem OCL. Uma expressão OCL bem formada deve seguir as regras de tipo da linguagem. Toda expressão tem um tipo e os termos presentes nesta devem ser usados de acordo com o seu tipo. Por exemplo, não é possível comparar uma String com um inteiro. Os tipos inicialmente definidos em OCL podem ser divididos em quatro grupos: tipos básicos como Integer, Real, Boolean e String; tipos

enumerados; tipos provenientes da UML como classes, associações e generalizações; e tipos compostos como Set, Bag e Sequence.

Devemos destacar também que OCL é uma linguagem de navegação, o que significa que também pode ser usada para se navegar através das associações [Martins, E. (2003)]. Outro aspecto a ser considerado é que esta linguagem permite especificar quatro tipos de restrições: invariantes (*inv*) de classe, pré (*pre*) e pós (*post*) condições de métodos e condições de guardas.

A primeira anteriormente citada é um predicado que se aplica às classes com o objetivo de garantir que uma determinada restrição seja válida para todos os objetos instâncias da classe. Pré-condição é o fato que deve ser satisfeito antes da execução de uma operação, e pós-condição é o fato que deve ser satisfeito após a execução de uma operação. Já a condição de guarda é a condição que deve ser satisfeita para que uma transição de estado ocorra.

O exemplo representado na figura 2.2 descreve a relação entre as classes Pessoa, Empresa, e Emprego onde essa é representada como a classe de associação da relação entre Pessoa e Empresa. Uma empresa pode contratar vários empregados e cada um deles pode pertencer a várias empresas.

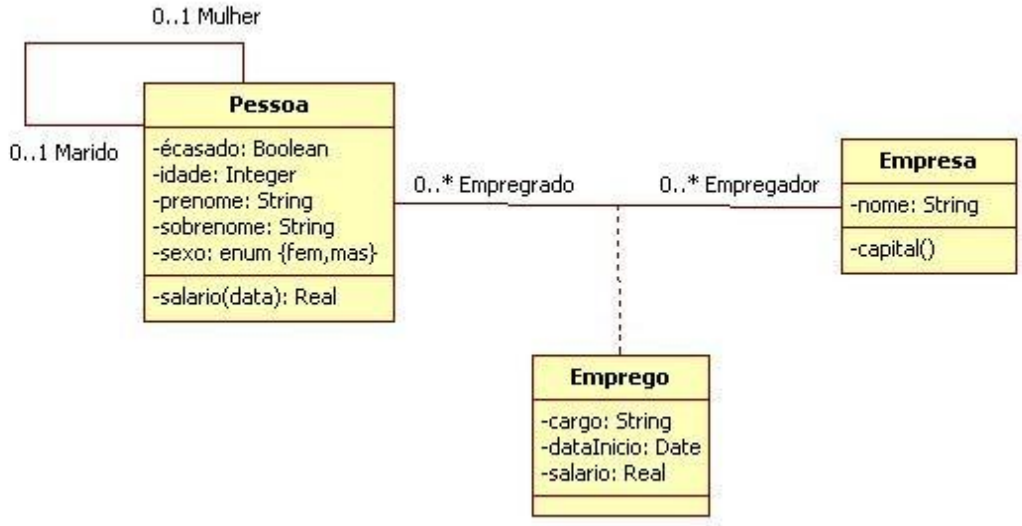

Figura 2.2 Exemplo de Modelagem

Alguns exemplos de expressões em OCL.

Contexto e Invariante:

Contexto: Pessoa

Invariante: (((self.mulher->isEmpty()) and (self.marido->isEmpty()))
or ((self.mulher->notEmpty()) and (self.marido->isEmpty()))
or ((self.mulher->isEmpty()) and (self.marido->notEmpty())))

A invariante descrita acima garante que uma pessoa não pode ter um marido e uma esposa ao mesmo tempo.

Pré e pós-condições:

Context: Pessoa:: getEmpresas(): Empresa

pre maiorIdade: self.idade>=18

post: (Set(Empresa) or null)

Neste caso, foi estabelecida uma pré-condição para garantir que apenas pessoas com idade superior ou igual a 18 anos participem da operação "getEmpresas", já que apenas pessoas com esta idade podem ter empregos. Como uma Pessoa pode ter relacionada a ela 0..* Empresas, a condição de *post* é dada por um conjunto de Empresas ou um conjunto vazio (*null*).

2.3 AGENTES E NORMAS

Agente é: "Pessoa ou coisa que atua ou que tem o poder de atuar; aquele que atua em benefício de outro" [Dictionary]. Mas se analisarmos no ponto de vista científico, podemos considerar a seguinte definição para agente de software: "Um agente é um sistema informático situado em um ambiente que é capaz de realizar ações de forma autônoma/independente para conseguir seus objetivos" [Wooldridhe, M. (1997)]. Podemos citar ainda a definição apresentada em [Jennings, N. R. (1999)]: "Um agente é um sistema de computador encapsulado que está situado em

algum ambiente e que é capaz de agir de maneira flexível e autônoma neste ambiente, a fim de cumprir os seus objetivos". Embora várias definições para agentes de software tenham sido propostas, ainda não existe uma única definição aceita e utilizada por todos os pesquisadores da área.

Algumas propriedades de agentes comumente aceitas por vários pesquisadores foram apresentadas em [Agent - Object Management Group (2000)] e estamos descrevendo a seguir:

- **Autonomia** – Capacidade de agir sem que haja intervenção externa e controle sobre seu estado e suas ações;

- **Interatividade** – Capacidade de se comunicar com o ambiente em que está inserido e com os outros agentes;

- **Adaptativo** – Capacidade de mudança no comportamento, baseado na sua experiência;

- **Socialmente interativo** – Capacidade de interagir em uma sociedade com o objetivo de colaborar ou competir com os outros agentes inseridos na mesma sociedade;

- **Proatividade** – Capacidade de tomar iniciativa e reconhecer as oportunidades, e não simplesmente reagir às informações vindas do ambiente;

- **Racionalidade** – Capacidade de selecionar uma ação considerando seus objetivos internos, sabendo que determinada ação irá aproximá-lo de seus objetivos.

Outro ponto importante na definição de agentes é o fato do comportamento destes ser orientado a objetivos, ou seja, sua execução se dá pela seleção de objetivos e execução de planos capazes de alcançar estes objetivos. Durante a execução dos planos, os agentes podem adaptar seu comportamento às mudanças

do ambiente no qual estão inseridos criando, por exemplo, novos planos, novos objetivos e adquirindo assim, novos conhecimentos sobre o ambiente e sobre outros agentes. Os agentes também são capazes de possuir crenças, que representam o que os agentes sabem e acreditam, bem como ações que são executadas.

Um Sistema Multiagentes (SMA) é composto por um conjunto de agentes autônomos que podem interagir cooperando, competindo ou ambos com o propósito de atingir objetivos individuais ou coletivos. Desta forma, a utilização de SMA ao invés de sistemas com um único agente, tem um grande potencial, visto que beneficia as seguintes propriedades descritas anteriormente: autonomia, reatividade, proatividade e interatividade social [Wooldridge, M. and Ciancarini, P. (2001)]. Para que os agentes entendam as suas responsabilidades e as dos outros agentes do sistema, normas são definidas. Estas são mecanismos utilizados para se restringir o comportamento dos agentes descrevendo as ações que devem ser executadas (i.e., suas obrigações), as ações que podem ser executadas (i.e., suas permissões) e as ações que não podem ser realizadas (i.e., suas proibições) [Silva, V. T., Braga, C. and Figueiredo, K. (2010)].

Uma norma pode ser aplicada a um agente em si, a um papel (caracterizando que todos os agentes que desempenham o papel devem cumprir com a norma) e a um conjunto ou grupo de agentes. Uma norma é válida durante um determinado período chamado de período de ativação e é justo durante este intervalo que a entidade a qual a norma se aplica deve cumpri-la. [Vazquez-Salceda, J., Aldewereld, H. and Dignum, F., (2005) e Silva, V. T. (2008)].

A definição das normas pode ocorrer em tempo de execução pelos agentes do sistema ou pode acontecer em tempo de design/projeto, ou seja, em conjunto com a modelagem do sistema. Como as normas mencionam elementos presentes na modelagem do sistema, como os agentes, suas ações e papéis que executam, é essencial que haja um alinhamento entre as normas e estes elementos porque alterações no sistema podem afetar as especificações das normas, assim como alterações nas normas podem influenciar um conjunto de elementos do sistema.

Outro aspecto relevante referente às normas são as sanções, utilizadas pelos sistemas de governança para punir ou premiar os agentes. Se uma norma for seguida, prêmios podem ser fornecidos, e se for violada, punições devem ser

aplicadas. Um prêmio ou uma punição são descritos como outra norma a ser ativada quando o cumprimento ou a violação da norma ocorre.

Como exemplo, podemos citar a seguinte norma para um agente que exerce um papel de Aluno: *O Aluno deverá entregar um livro antes da data de entrega prevista.* Caso ele entregue no período desejado ele será *permitido* a executar a ação de pegar outros livros para consulta na Biblioteca. Caso contrário, ele será punido com a *proibição* de empréstimos por um período pré-determinado pela Instituição.

Alguns exemplos de normas para o sistema de uma biblioteca:

- **O Aluno deve entregar o livro antes ou na data de entrega prevista.**
  Os agentes que executam o papel de Aluno são obrigados a entregar o livro dentro de um prazo estipulado pela Biblioteca.

- **O Aluno não pode danificar o livro.**
  Os agentes que executam o papel de Aluno são proibidos de executarem a ação de danificar um livro. Neste caso, nenhuma restrição de tempo foi declarada. Uma sanção de punição possível seria proibir o agente de executar a ação de pegar livros emprestados por um determinado período de tempo ou mesmo de se desligar da Biblioteca.

- **O Bibliotecário não poderá trocar o status do livro quando este estiver emprestado.**
  Aos agentes que exercem o papel de Bibliotecário é aplicada uma norma de proibição a ação de trocar o status do livro se este estiver emprestado.

Note que as normas definidas em um determinado sistema podem estar em conflito. Conflitos entre normas ocorrem quando (i) uma norma proíbe e outra permite um agente de executar uma ação, no mesmo intervalo de tempo e em um mesmo contexto de execução; (ii) uma norma proíbe e outra obriga um agente a executar uma ação, no mesmo intervalo de tempo e em um mesmo contexto de

execução; e (iii) uma norma permite e outra obriga um agente a executar uma ação, em um mesmo contexto de execução porém em intervalos de tempo que não se interceptam. Assumimos que fora do escopo da permissão o agente é proibido de realizar a ação.

Quando as normas são definidas em tempo de design, alguns desses conflitos podem ser detectados e resolvidos. Resolver uma parte dos conflitos no tempo de design implica em reduzir o tempo em que os agentes vão gastar executando essa tarefa em tempo de execução.

Assim, é importante que as linguagens de modelagem, as notações utilizadas nas metodologias e os modelos organizacionais que modelam os sistemas multiagentes possibilitem a modelagem de normas juntamente com a modelagem de todo o sistema e que ainda tenham mecanismos para detectar e se possível solucionar conflitos em tempo de design.

CAPÍTULO 3: NORMML

3.1 A LINGUAGEM NORMML

NormML [Figueiredo, K. S., and Silva, V. T. (2010)] é uma linguagem de modelagem para a especificação de normas em SMA baseada na linguagem UML [UML – OMG (2005)]. A escolha por UML como meta-modelo permite uma fácil integração de NormML com linguagens de modelagens para SMA baseadas em UML como AUML[Odell, J., Parunak, H., and Bauer, B.(2000)], AML[Danc, J. (2008)] e MAS-ML [Silva, V. T., Choren R. and Lucena, C. (2008)].

A linguagem de modelagem NormML foi implementada levando em consideração que a especificação de normas no design de SMA e a especificação de políticas de segurança no design de RBAC (Roles-Based Access Control) [Basin, D., Clavel, M., Doser, J. and Egea, M. (2009)] são problemas fortemente acoplados. As políticas de segurança no RBAC especificam as permissões que os usuários possuem quando desempenham um determinado papel. Nos SMA especificamos as normas que regulam o comportamento (ou as ações) de um agente que exerce um determinado papel. Embora consideremos as políticas de segurança e os elementos relacionados, as normas podem ser violadas.

3.2 O META-MODELO DE NORMML

A especificação de normas em SMA é feita através da modelagem utilizando a linguagem NormML. Visto que esta se baseia na linguagem SecureUML, o meta-modelo de NormML se baseia no meta-modelo de SecureUML, porém não é

conservativa já que algumas invariantes de SecureUML não foram respeitadas. Novos conceitos foram incluidos, como: agentes, normas, ações de agentes e uma série de invariantes [Silva, V. T., Braga, C. and Figueiredo, K. (2010)]. A escolha pela SecureUML foi feita porque esta linguagem possui uma sintaxe bem definida, dada pelo seu meta-modelo, uma semântica formal [Basin, D., Clavel, M., Doser, J. and Egea, M. (2009)] e é projetada especificamente para a modelagem RBAC.

O meta-modelo de NormML tem (algumas) meta-classes originais de SecureUML, destacadas em cinza nas figuras desta subseção, e as novas meta-classes definidas de acordo com os conceitos relacionados às normas e apresentados em [Silva, V. T., Braga, C. and Figueiredo, K. (2010)]. A seguir, este meta-modelo será melhor detalhado.

3.2.1 Conceito Deôntico

As normas em NormML, como já citado, definem uma permissão (*NormPermission*), uma obrigação (*NormObligation*) ou uma proibição (*NormProhibition*), para uma determinada entidade. A figura 3.2 apresenta parte do meta-modelo de NormML que permite a criação destes três tipos de normas instanciando as devidas meta-classes.

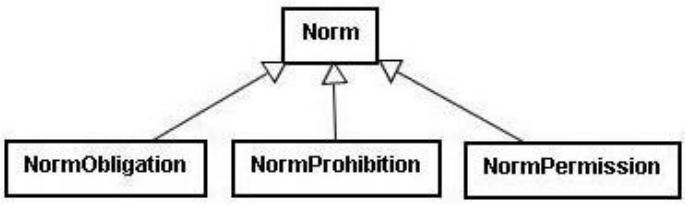

Figura 3.1 Conceito Deôntico relacionado às metaclasses do Meta-modelo de NormML [Meta-modelo NormML (2010)]

Para um melhor entendimento podemos ter uma norma de obrigação, *Um aluno é obrigado a frequentar 75% das aulas*; uma norma de proibição, *Um aluno é*

proibido de colar em provas; e uma norma de permissão, *Um aluno pode participar do concurso de monitoria*.

3.2.2 Contexto

A atual versão de NormML nos permite definir normas no contexto de uma organização ou de um ambiente. As organizações definem os papéis desempenhados pelos agentes. E tanto as organizações e quanto os agentes, por sua vez, habitam um ambiente [Figueiredo, K. and Silva, V. T. (2010)]. Na figura 3.3 é apresentado uma parte do meta-modelo de NormML que define as possíveis relações para o contexto de uma norma.

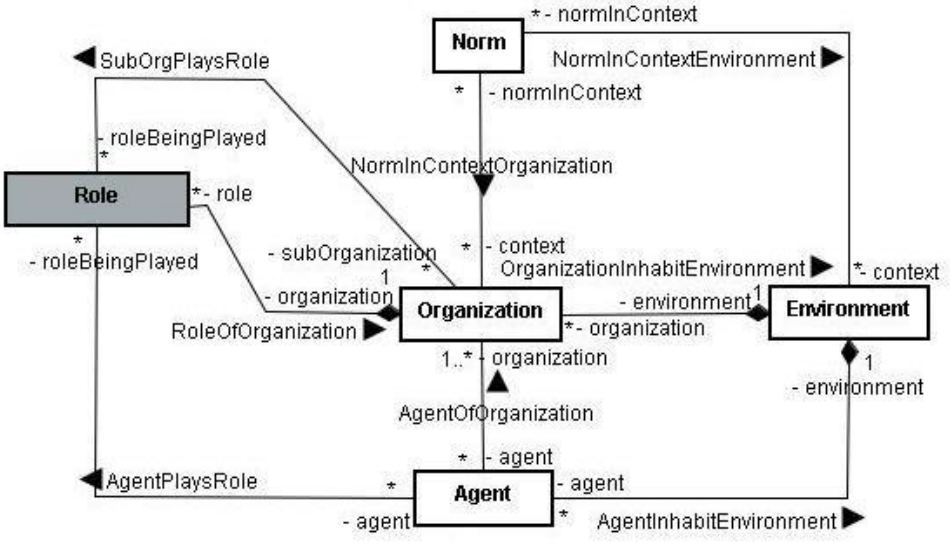

Figura 3.2 Contexto relacionado às metaclasses do Meta-modelo de NormML [Meta-modelo NormML (2010)]

3.2.3 Entidades Envolvidas

As normas podem ser aplicadas (i) a um papel (*Role*), caracterizando que todos os agentes que desempenham tal papel devem cumprir com a norma, (ii) a um agente (*Agent*), (iii) a um agente quando este desempenha um determinado papel (*Agent playing role*), (iv) a uma organização (*Organization*), isto é, para todos os agentes ou suborganizações da organização, (v) a um ambiente (*Environment*), isto é, a todos os agentes e organizações que habitam este ambiente, ou ainda (vi) a uma suborganização exercendo um papel (*Organization playing role*).

A Figura 3.3, apresentamos a parte do meta-modelo de NormML que define as relações entre uma norma e as suas possíveis entidades relacionadas.

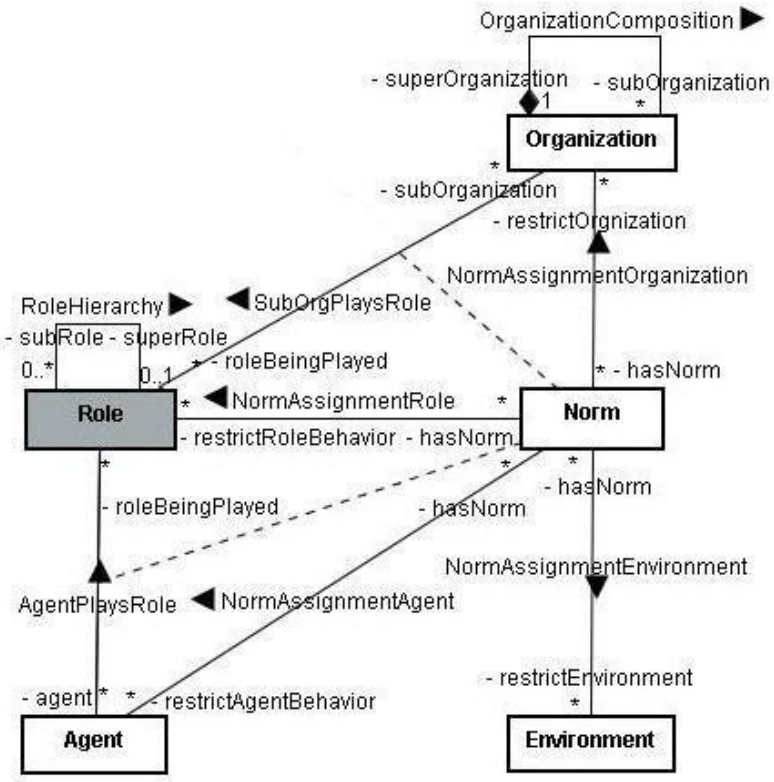

Figura 3.3 Entidades envolvidas relacionadas às metaclasses do Meta-modelo de NormML [Meta-modelo NormML (2010)]

3.2.4 Recursos e Ações

Uma norma restringe o acesso de uma entidade a um determinado recurso (*Resource*). A norma pode restringir o acesso a um determinado objeto do sistema (i.e., a seus atributos (*Attribute*), a métodos (*Method*) de um objeto, a todo objeto (*Entity*)) ou à relação entre o objeto e outra entidade (*AssociationEnd*)) . Uma norma pode ainda restringir o acesso a uma crença (*Belief*) ou objetivo do agente (*Goal*), a uma ação a ser executada pelo agente (*AgentAction*), a um plano do agente (*Plan*), a uma mensagem a ser enviada ou recebida pelo agente (*Message*), a um agente (*Agent*), organização (*Organization*), papel (*Role*), protocolo (*Protocol*) ou ambiente (*Environment*). A figura 3.4 apresenta as meta-classes utilizadas para a definição dos recursos.

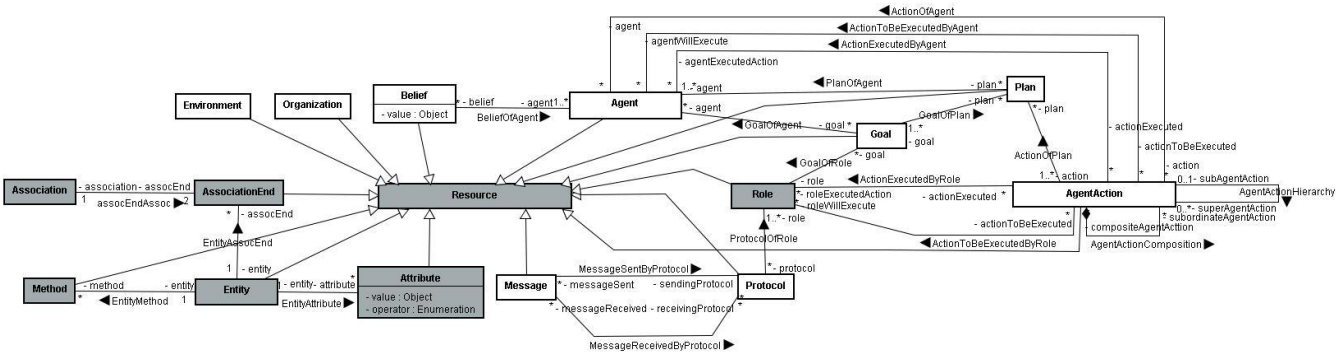

Figura 3.4 Recursos relacionados às metaclasses do Meta-modelo de NormML [Meta-modelo NormML (2010)]

O tipo de acesso a ser restringido em cada recurso depende da natureza do recurso. O aceso pode ser simples, representado por ações atômicas (*AtomicAction*) ou, complexos, representados por um conjunto de ações (*CompositeAction*). A seguir encontra-se uma tabela (tabela 3.1) que relaciona cada recurso com as ações que podem ser utilizadas para acessar o recurso. A figura 3.5 apresenta todas as ações propostas pelo meta-modelo de NormML. Com o objetivo de exemplificar a relação entre recurso e ação, podemos descrever uma norma para restringir a leitura (*read*), a atualização (*update*) ou qualquer acesso (*full access*) a um atributo de um objeto. Já no caso de uma ação de um agente, só podemos restringir sua execução (*execute*).

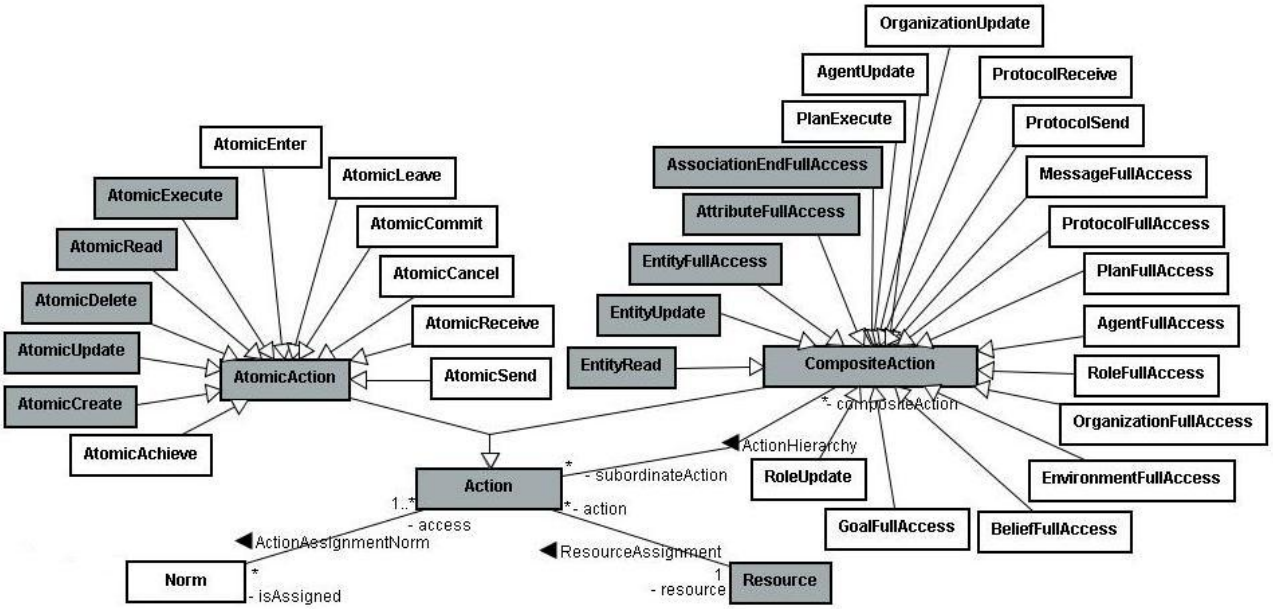

Figura 3.5 Ações relacionadas às metaclasses do Meta-modelo de NormML [Meta-modelo NormML (2010)]

Tabela 3.1 Recursos e suas ações

Recurso	Ações
Agente	Criação, exclusão, atualização e acesso total
Papel	Criação, compromisso, exclusão, cancelamento, atualização e acesso total
Organização	Criação, exclusão, entrada, saída, atualização e acesso total
Ambiente	Exclusão, criação, entrada, saída e acesso total
Atributo	Atualização, leitura, alcançado *e* acesso total
Entidade	Criação, exclusão, leitura, atualização e acesso total.
Ação de Agente	Execução
Ponta de Associação	Atualização, leitura e acesso total
Método	Execução
Mensagem	Recebimento, envio e acesso total
Protocolo	Criação, exclusão, recebimento, envio e acesso total
Crença	Criação, exclusão, atualização e acesso total
Objetivo	Alcançado, compromisso, cancelamento e acesso total
Plano	Criação, atualização, exclusão, execução e acesso total.

3.2.5 Restrições de Ativação

Como uma norma está ativa durante um determinado período de tempo, é necessário poder especificar este período através de restrições de escopo da norma. Essa definição é feita através da metaclasse *NormConstraint.* Uma norma pode ser condicionada através das cláusulas Before, After, Between e If, conforme mostrado na figura 3.6. No caso da primeira, significa que a norma estará ativa antes da execução de uma ação e/ou antes do alcance de uma determinada data. Já se a norma estiver associada a uma cláusula After, esta estará ativa após a execução de uma determinada ação e/ou após do alcance de uma determinada data. A cláusula Between indica que a norma estará ativa apenas entre as execuções de duas ações ou duas datas, delimitando assim, um intervalo de ativação. Por último, a cláusula If, a norma apenas estará ativa se uma data, um determinado valor de um atributo ou de uma crença for atingido.

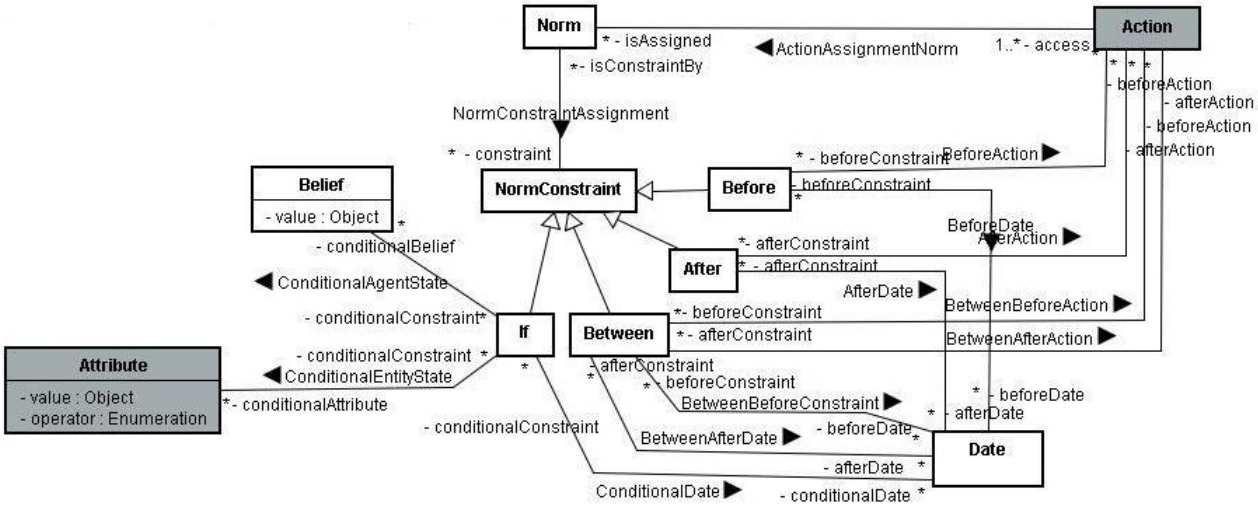

Figura 3.6 Restrições de ativação relacionadas às metaclasses do Meta-modelo de NormML [Meta-modelo NormML (2010)]

3.2.6 Sanções

Caso a norma seja violada ou cumprida, poderá ocorrer punição (*Punishment*) e premiação (*Reward*), respectivamente (como já discutido na seção 2.3). A figura 3.7 é uma parte do meta-modelo de NormML que define as relações estabelecidas entre as normas e as sanções. Conforme exemplificado na figura 3.7, uma norma pode possuir uma ou mais sanções (relacionamento *SanctionOfNorm*). Uma sanção, por sua vez, aplica uma ou mais normas à entidade relacionada (relacionamento *SanctionAppliesNorm*), ou seja, como recompensa ou punição, a entidade deverá executar a norma estabelecida pela sanção da norma.

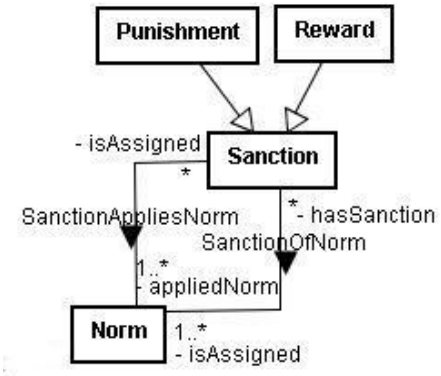

Figura 3.7 Sanção relacionada às metaclasses do Meta-modelo de NormML [Meta-modelo NormML (2010)]

3.3 GERANDO AS NORMAS

Nesta seção detalharemos exemplos de modelos de normas baseados em um sistema de gerenciamento de Biblioteca. Estes modelos foram criados através da

instanciação direta das meta-classes apresentadas na seção 3.2. Considere as seguintes normas:

- *O Aluno não pode danificar o livro.*

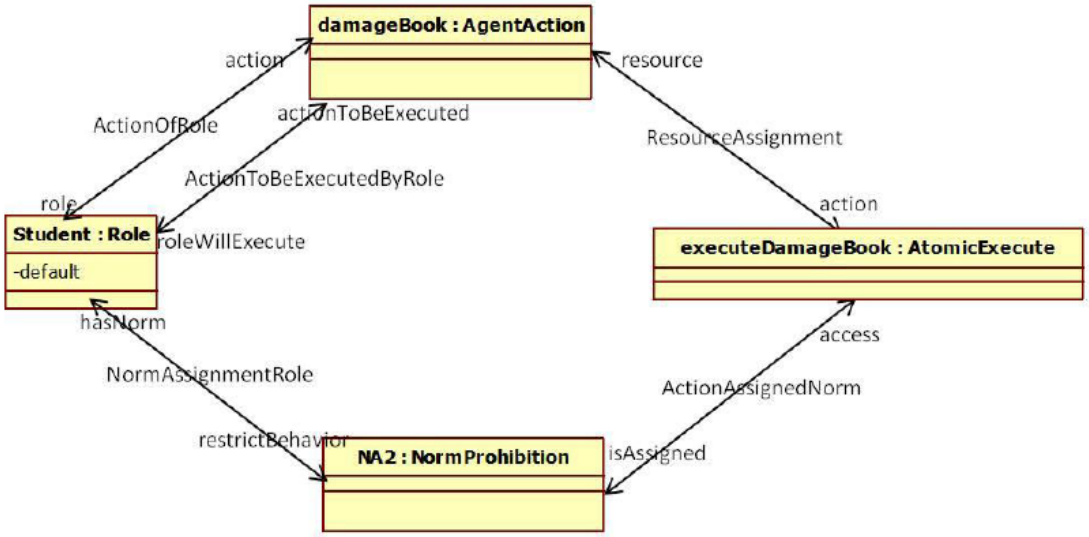

Figura 3.8 Exemplo de norma descrita na linguagem NormML

Neste modelo é notório que NA2 é uma instância da meta-classe de NormML "*NormProhibition*" que restringe o comportamento do Student (instância da classe "*Role*") proibindo entidades que desempenham este papel de executar (representado por executeDamageBook instância da classe "*AtomicExecute*") a ação do agente damageBook (instância da classe "*AgentAction*").

- O *Aluno deve entregar o livro antes ou na data de entrega prevista.*

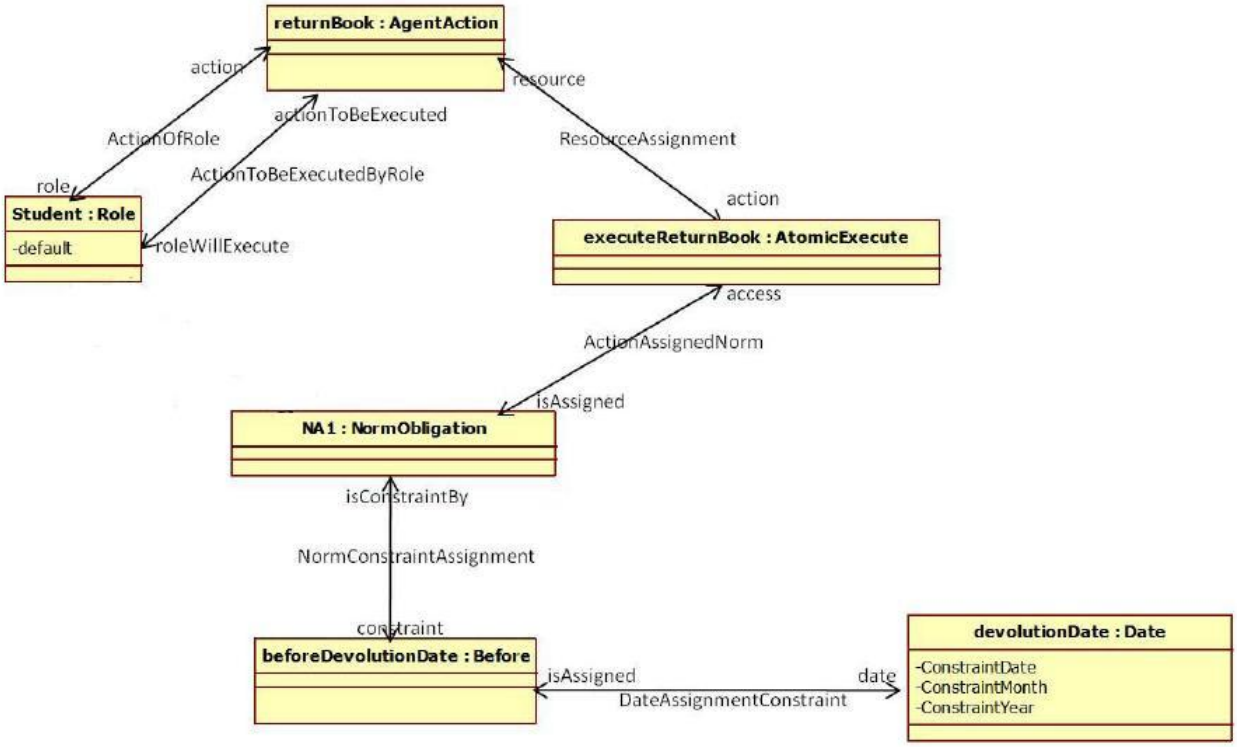

Figura 3.9 Exemplo de norma descrita na linguagem NormML

Já neste modelo é possível identificar que NA1 é uma instância da classe de NormML "*NormObligation*" que restringe o comportamento do Student (instância da classe "*Role*") obrigando as entidades que desempenham este papel de executar (representado pela ação executeReturnBook instância da classe "*AtomicExecute*") a ação do agente returnBook (instância da classe "*AgentAction*") no período descrito por beforeDevolutionDate (instância da classe "*Before*") com uma determinada data de devolução (devolutionDate instância da classe *"Date"*). Note que este modelo está incompleto pois o relacionamento entre NA1 e Student não foi representado propositalmente. Na seção 3.4 voltaremos a este diagrama analisando este problema.

3.4 REGRAS DE BOA FORMAÇÃO

Como apresentado na seção 2.2, definir os elementos do meta-modelo e as relações entre eles normalmente não é suficiente para garantir que os modelos instancias destes meta-modelos sejam bem formados. Este é o caso de NormML. Por tanto, foram definidas diversas regras de boa-formação associadas ao meta-modelo de NormML. Um dos passos para a validação das normas representadas como modelos em NormML é a verificação dos modelos de acordo com as regras de boa-formação. Como será visto na próxima subseção, o outro passo é a verificação da existência ou não de conflitos entre as normas de um sistema multi-agente.

Para realizar a verificação das regras de boa formação temos a modelagem da norma, como uma instância do meta-modelo NormML, sendo verificada de acordo com as invariantes do meta-modelo. Com as invariantes é possível certificar que o modelo onde a norma se aplica está bem formado baseado nas especificações do meta-modelo.

A seguir, apresentamos alguns exemplos de regras de boa formação, baseadas no meta-modelo utilizado neste trabalho.

- *As normas aplicadas a um agente devem restringir as ações desse agente.* A aplicação desta regra é útil pois uma norma aplicada a um agente não pode restringir a execução de uma ação que o agente não conhece.

Contexto: Agent :: normsRestrictActionAgentOne: Boolean
Ope: self.action->includesAll(self.hasNorm.oclAsType(NormObligation)->collect (norma:Norm|norma.access)->collect(action:Action|action.resource.oclAsType (AgentAction)))

Contexto: Set(Agent) :: rule4: Boolean
Ope: self->forAll(agente:Agent|agente.normsRestrictActionAgentOne())

- As Ações AtomicReceive e *AtomicSend só* podem estar relacionadas *a um recurso Message.* Apenas duas ações se aplicam ao uso do recurso *Message* (ou mensagem): o envio da mensagem (representado

pela ação *AtomicSend*) e o recebimento da mensagem (representado pela ação *AtomicReceive*).

Contexto: Action :: restrictActionMessageOne_: Boolean
Ope: self.resource.IsResourceMessage()

Contexto: Set(Action) :: rule17: Boolean
Ope: self->forAll(action:Action|if(action.IsActionAtomicSend())then
(action.restrictActionMessageOne())else(true)endif)

Contexto: Set(Action) :: rule18: Boolean
Ope: self->forAll(action:Action|if(action.IsActionAtomicReceive())then
(action.restrictActionMessageOne_())else(true)endif)

- *Uma recompensa para uma entidade não pode aplicar uma NormProhibition ou NormObligation para a mesma entidade.* A aplicação desta regra é necessária pois trata-se de uma recompensa, portanto não se pode proibir ou obrigar a entidade que está sendo recompensada a executar uma ação.

Contexto: Sanction :: IsReward: Boolean
Ope: self.oclIsTypeOf(Reward)

Contexto: Sanction :: Rule_26: Boolean
Ope: self.isAssigned->forAll(norms:Norm|self.appliedNorm->
forAll(norma:Norm|if(norma.IsNormProhibition() or
norma.IsNormObligation())then((((norma.restrictRoleBehavior)->
intersection(norms.restrictRoleBehavior))->isEmpty())
and((((norma.restrictEnvironmentBehavior)->
intersection(norms.restrictEnvironmentBehavior))->isEmpty()))
and((((norma.restrictOrganization)->intersection(norms.restrictOrganization))->isEmpty()))
and((((norma.restrictAgentBehavior)->intersection(norms.restrictAgentBehavior))->
isEmpty())))else(if(norma.IsNormPermission())then(true)else(false)endif)endif))

Contexto: Set(Sanction) :: rule26: Boolean
Ope: self->forAll(sanction:Sanction|if(sanction.IsReward())then(sanction.Rule_26())else
(true)endif)

- *Uma norma deve estar no contexto de uma organização ou de um ambiente.* É interessante a aplicação desta regra pois obriga que a norma tenha um contexto restringindo assim o escopo de execução da norma.

Context: Set(Norm) :: rule2: Boolean

Ope: self->forAll(norma:Norm|((((norma.context1)->isEmpty()) and ((norma.context2)->
notEmpty())) or (((norma.context1)->notEmpty()) and ((norma.context2)->isEmpty()))))

- *Uma norma deve restringir um agente, um papel, um agente exercendo um papel, um ambiente, uma organização ou uma suborganização exercendo um papel.* A aplicação desta regra é indispensável pois garante que a norma esteja restringindo o comportamento de pelo menos uma das entidades descritas acima, certificando assim que a modelagem de normas ocorra de forma esperada.

```
Context: Set(Norm) :: rule1: Boolean

Ope: self->forAll(norma:Norm|if((norma.restrictAgentBehavior)->
isEmpty())then(((norma.restrictRoleBehavior)->
notEmpty())or((norma.restrictOrganization)->
notEmpty())or((norma.restrictEnvironmentBehavior)->
notEmpty()))else(if((norma.restrictRoleBehavior)->
isEmpty())then(((norma.restrictAgentBehavior)->
notEmpty())or((norma.restrictOrganization)->
notEmpty())or((norma.restrictEnvironmentBehavior)->
notEmpty()))else(if((norma.restrictOrganization)->
isEmpty())then(((norma.restrictAgentBehavior)->
notEmpty())or((norma.restrictRoleBehavior)->
notEmpty())or((norma.restrictEnvironmentBehavior)->
notEmpty()))else(if((norma.restrictEnvironmentBehavior)->
isEmpty())then(((norma.restrictAgentBehavior)->
notEmpty())or((norma.restrictRoleBehavior)->notEmpty())or((norma.restrictOrganization)->
notEmpty()))else(if(((norma.restrictAgentBehavior)->
notEmpty())and((norma.restrictRoleBehavior)->
notEmpty())and((norma.restrictOrganization)->
notEmpty())and((norma.restrictEnvironmentBehavior)->
notEmpty()))then(true)else(false)endif)endif)endif)endif)endif)
```

Para assimilar a aplicação das regras de boa formação, avaliaremos as figuras descritas na subseção 3.3 empregando essas regras. O modelo representado na Figura 3.8 está mal formado pois não segue a seguinte regra de boa formação: *Uma norma deve estar no contexto de uma organização ou de um ambiente.* Ao definir a norma não foi especificado o contexto de execução da mesma, sendo assim, não se sabe a qual organização o papel *Student* está relacionado.

Já no modelo produzido na Figura 3.9, além de infringir a mesma regra descrita anteriormente, o modelo não segue a seguinte regra: *uma norma deve restringir um agente, um papel, um agente exercendo um papel, um ambiente, uma organização ou uma suborganização exercendo um papel.* A norma de obrigação

não está associada a nenhuma entidade cujo comportamento ela deveria estar restringindo. Não foi modelada a relação direta entre a norma e o papel *Student*.

Todas as regras que foram definidas para este trabalho, estão disponíveis em [Regras de Boa Formação (2010)].

3.5 REGRAS DE VERIFICAÇÃO DE CONFLITOS

Após a verificação das regras de boa formação, é importante analisar se há conflitos entre as normas. Conforme já explicado na seção 3.2, uma norma em NormML é composta pelos seguintes elementos: seu contexto, as entidades envolvidas, o seu conceito deôntico, suas ações, restrições de ativação e suas possíveis sanções. Desta forma, para checarmos se existem conflitos entre as normas, é necessário verificar cada uma dessas partes, como será melhor detalhado a seguir.

3.5.1 Contexto

Para verificar um estado de possível conflito entre duas normas, é importante verificar (i) se ambas estão definidas em um mesmo contexto (se estão definidas para uma mesma organização ou um mesmo ambiente), (ii) se uma das normas está definida para um ambiente e a outra está definida no contexto de organizações que habitam este ambiente, ou ainda (iii) se as normas estão definidas em uma mesma hierarquia de organizações. Caso as duas normas estejam em contextos relacionados ou no mesmo contexto, elas podem estar em conflito. Se as normas

estiverem em contextos não relacionados, então seguramente não estarão em conflito pois o agente ao qual as normas se aplicam não terá que tomar duas atitudes contraditórias no mesmo escopo de execução. Esta operação verifica se as normas estão no contexto de uma mesma organização.

Contexto: Norm :: NormsAppliedOnTheSameOrganization(n2:Norm): Boolean
Ope: self.context1->exists(org1:Organization|n2.context1->exists(org2:Organization|org1=org2))

3.5.2 Entidades Envolvidas

Caso as normas estejam definidas para contextos de execução relacionados, é importante verificar se os agentes aos quais estas normas se aplicam são os mesmos. Para tanto é necessário verificar (i) se as normas estão aplicadas a uma mesma entidade, (ii) se uma está definida para um papel e a outra a um agente ou suborganização que pode exercer esse papel, (iii) se as normas estão aplicadas a diferentes papéis que são exercidos pelo mesmo agente ou pela mesma suborganização, (iv) se as normas estão aplicadas a uma mesma hierarquia de papéis, ou ainda (v) se uma norma está aplicada a uma organização e a outra norma a papéis, agentes ou suborganizações dessa organização. A operação, descrita no quadro, em OCL que verifica se duas normas estão aplicadas a um mesmo agente (ou ainda, possuem mais de um agente em comum).

Contexto: Norm::NormsAppliedSameAgents(n2:Norm): Boolean
Ope: (self.restrictOrganization->intersection(n2.restrictOrganization))->notEmpty()

3.5.3 Conceito Deôntico

Caso as normas estejam em contextos relacionados e se apliquem a mesma entidade, duas normas estarão em estado de potencial conflito se uma das três situações a seguir acontecer: (i) uma norma é uma permissão e outra é uma proibição; (ii) uma norma é uma obrigação e outra é uma proibição; e (iii) uma norma é uma permissão e a outra é uma obrigação (onde estas não estão ativas em um mesmo período de tempo. O conflito ocorre porque a entidade é obrigada a executar uma ação que não tem permissão para executar). A operação que faz essa verificação de conflito deôntico é mostrada a seguir.

Contexto: Norm:: deonticInconsistence(n2:Norm): Boolean

Ope:((((self.oclIsTypeOf(NormObligation))and(n2.oclIsTypeOf(NormPermission)))or((n2.oclIsTypeOf(NormObligation))and(self.oclIsTypeOf(NormPermission))))or(((self.oclIsTypeOf(NormObligation))and(n2.oclIsTypeOf(NormProhibition)))or((n2.oclIsTypeOf(NormObligation))and(self.oclIsTypeOf(NormProhibition))))or(((self.oclIsTypeOf(NormPermission))and(n2.oclIsTypeOf(NormProhibition)))or((n2.oclIsTypeOf(NormPermission))and(self.oclIsTypeOf(NormProhibition)))))

3.5.4 Ações

Se uma das situações de potencial conflito listadas anteriormente for obedecida, é necessário verificar se: (i) as ações que estão sendo reguladas pelas normas são do mesmo tipo e atuam no mesmo recurso, (ii) uma das normas está relacionada a uma ação de *EntityRead*, *EntityUpdate* ou *EntityFullAccess* de uma entidade e a outra a uma ação de *AtomicRead* ou *AtomicUpdate* aos atributos ou pontas de associação desta mesma entidade, (iii) uma das normas está relacionada a uma ação de *EntityUpdate* ou *EntityFullAccess* a uma entidade e a outra a uma *AtomicExecute* aos métodos desta entidade, (iv) uma ação é de *AttributeFullAccess* e a outra de *AtomicRead* ou *AtomicUpdate* de um mesmo atributo de uma mesma

entidade, (v) ação é uma *AssociationEndFullAccess* e a outra uma *AtomicRead* ou uma *AtomicUpdate* de uma mesma ponta de associação de uma Associação, (vi) uma ação de *AtomicRead* e a outra uma *AtomicUpdate* aplicadas a um mesmo atributo de uma mesma entidade ou de uma mesma ponta de associação de uma associação, (vii) uma ação é uma *EntityRead* e a outra uma *EntityUpdate* de uma mesma entidade, (viii) uma é uma *MessageFullAccess* e a outra uma *AtomicSend* ou *AtomicReceive* para uma mesma mensagem, (ix) uma ação é *GoalFullAccess* e a outra um *AtomicAchieve*, a *AtomicCommit* ou *AtomicCancel* para um mesmo objetivo, (x) uma ação é *BeliefFullAccess e* a outra *AtomicCreate,* AtomicDelete ou *AtomicUpdate* para uma mesma crença, (xi) uma ação é *PlanFullAccess* e a outra *AtomicCreate, AtomicDelete, AtomicUpdate* ou *PlanExecute* para um mesmo plano, (xii) uma ação é *PlanExecute* e a outra *AtomicExecute* para as ações de um mesmo plano, (xiii) uma ação é *AgentFullAccess* e a outra *AtomicCreate, AtomicDelete* ou *AgentUpdate* para um mesmo agente, (xiv) uma ação é *AgentUpdate* e a outra *BeliefFullAccess, GoalFullAccess* ou *PlanFullAccess* para as crenças, objetivos ou planos de um mesmo agente, (xv) uma ação é *RoleFullAccess* e a outra *AtomicCreate, AtomicDelete, AtomicCommit, AtomicCancel* ou *RoleUpdate* para um mesmo papel, (xvi) uma ação é RoleUpdate e a outra GoalFullAccess ou ProtocolFullAccess para os objetivos e os protocolos de um mesmo papel, (xvii) uma ação é *OrganizationFullAccess* e a outra *AtomicCreate, AtomicDelete, AtomicEnter, AtomicLeave* ou *OrganizationUpdate* para uma mesma organização ou as suas suborganizações, (xviii) uma ação é *OrganizationUpdate* e a outra *BeliefFullAccess*, *GoalFullAccess, PlanFullAccess* ou *RoleFullAccess* para as crenças, objetivos, planos ou papéis para uma mesma organização ou as suas suborganizações, (xix) uma ação é *EnvironmentFullAccess* e a outra *AtomicCreate, AtomicDelete,* AtomicEnter ou *AtomicLeave* para um mesmo ambiente; (xx) uma ação é *ProtocolFullAccess* e a outra *AtomicCreate*, *AtomicDelete*, *ProtocolReceive*, *ProtocolSend* para um mesmo protocolo (xxi) uma ação é *ProtocolFullAccess* e a outra *AtomicSend* ou *AtomicReceive* para as mensagens de um mesmo protocolo, (xxii) uma ação é *ProtocolReceive* e a outra *AtomicReceive* para as mensagens de um mesmo protocolo, (xxiii) uma ação é *ProtocolSend* e a outra *AtomicSend* para as mensagens de um mesmo protocolo, ou (xxiv) se as ações de agente regulados pelas normas estiverem na mesma hierarquia de ação do agente.

Além destas, os seguintes casos especiais devem ser considerados: (i) uma ação de *AtomicCreate* e uma outra *AtomicDelete* para uma mesma entidade, agente, papel, plano, crença, protocolo, organização ou ambiente pode estar em situação de conflito se os conceitos deônticos associados as normas forem os mesmos. Por exemplo, um mesmo agente não pode ser obrigado a criar um plano e a apagar o mesmo plano ao mesmo tempo; (ii) uma ação de *AtomicEnter* e uma *AtomicLeave* aplicadas a uma mesma organização ou ambiente podem estar em conflito se os conceitos deônticos associados as normas forem os mesmos. Por exemplo, um mesmo agente não pode ser obrigado a entrar e a sair da mesma organização ao mesmo tempo; (iii) *AtomicCommit* e *AtomicCancel* para um mesmo papel ou objetivo podem estar em conflito se os conceitos deônticos associados as normas forem os mesmos. Por exemplo, o mesmo agente não pode ser obrigado a se comprometer a desempenhar um papel e a cancelar a execução deste mesmo papel ao mesmo tempo; (iv) entre normas relacionadas a ações de *AtomicAchieve* para um mesmo atributo/crença com diferentes valores se os conceitos deônticos associados as normas forem os mesmos. Por exemplo, um agente não pode ser obrigado a atingir dois valores diferentes para um mesmo atributo no mesmo intervalo de tempo.

Contexto: Action:: VerifyActionSameTypeResource(a2:Action): Boolean
Ope: (self.ActionsOfSameType(a2) and self.ActionsOfSameResource(a2))

3.5.5 Restrições de Ativação

Caso ao menos um caso ocorra em cada uma das seções de 3.5.1 até 3.5.4, duas normas estarão em situação de conflito se: (i) os períodos estabelecidos pelas invariantes *Before, After, Between* ou *If* se intersectam ou (ii) os valores relacionados a um mesmo atributo se intersectam, no caso de duas condicionais com a invariante If (ex: x>10 e x=20).

Contexto: If:: VerifyIfIntersectsValues(if2: IF): Boolean
Ope: self.conditionalAttribute->exists(att1:Attribute|if2.conditionalAttribute->exists (att2:Attribute|if((((att1.Operator='=')or(att2.Operator='='))and(att1.Value=att2.Value))or((att1.Operator ='>')and(att2.Operator='>'))or((att1.Operator='<')and(att2.Operator='<'))or(((att1.Operator='>')and(att2. Operator='<'))and(att2.Value>att1.Value))or(((att1.Operator='<')and(att2.Operator='>'))and(att1.Value> att2.Value)))then(true)else(false)endif))

Contexto: NormConstraint:: AttributesValuesIntersects(n2: NormConstraint): Boolean
Ope: if(self.IsIf() and n2.IsIf())then (self.oclAsType(If).VerifyIfIntersectsValues(n2.oclAsType(If)))else(false)endif

Contexto: Norm:: IfConditionalStateConflicts (n2: Norm): Boolean
Ope: self.constraint->exists(nc1:NormConstraint|n2.constraint->exists (nc2:NormConstraint|nc1.AttributesValuesIntersects(nc2)))

Todas as regras que foram definidas para este trabalho, estão disponíveis em [Regras de Verificação de Conflitos (2010)].

CAPÍTULO 4: FERRAMENTA PARA DEFINIÇÃO DE NORMAS

4.1 VISÃO GERAL DA FERRAMENTA

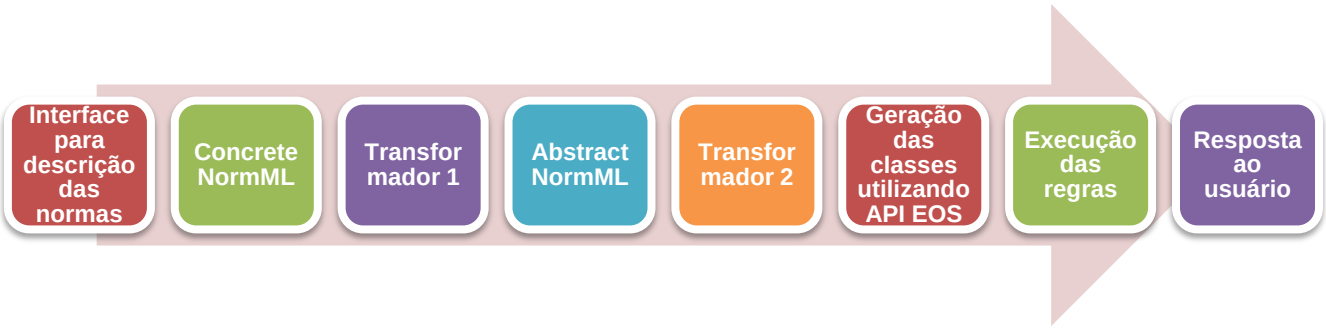

Figura 4.1 Visão geral da ferramenta

A ferramenta possui o seguinte fluxo. O usuário descreve os modelos das normas desejadas utilizando a interface provida pela ferramenta. A partir destes, é gerado um arquivo com extensão XMI (normML.xmi) contendo todas as informações modeladas. Em seguida, este primeiro arquivo sofre uma transformação que tem por objetivo gerar um XMI do modelo abstrato a partir do modelo concreto das normas (criando o arquivo uml.xmi). Após esta etapa, o arquivo é lido e a partir deste são gerados os objetos a que serão utilizados para a validação das normas de acordo com as regras de boa formação e as regras de verificação de conflito. Na fase de execução, as regras de boa formação são executadas, e caso o modelo esteja bem formado, realizamos a verificação de conflitos. A última etapa consiste na exibição de um feedback ao usuário, informando se o modelo foi considerado bem formado e ainda se as normas estão em situação de conflito.

Cada etapa será melhor definida nas subseções seguintes.

4.2 ARMAZENANDO AS NORMAS

4.2.1 Interface da Aplicação

Para esclarecer o uso da ferramenta vamos detalhar as principais telas. Primeiramente, vamos detalhar o menu da ferramenta. Neste temos as opções *File, Norm, Check for Conflicts, Elements, Help* e *Exit*, como ilustrado na Figura 4.2.

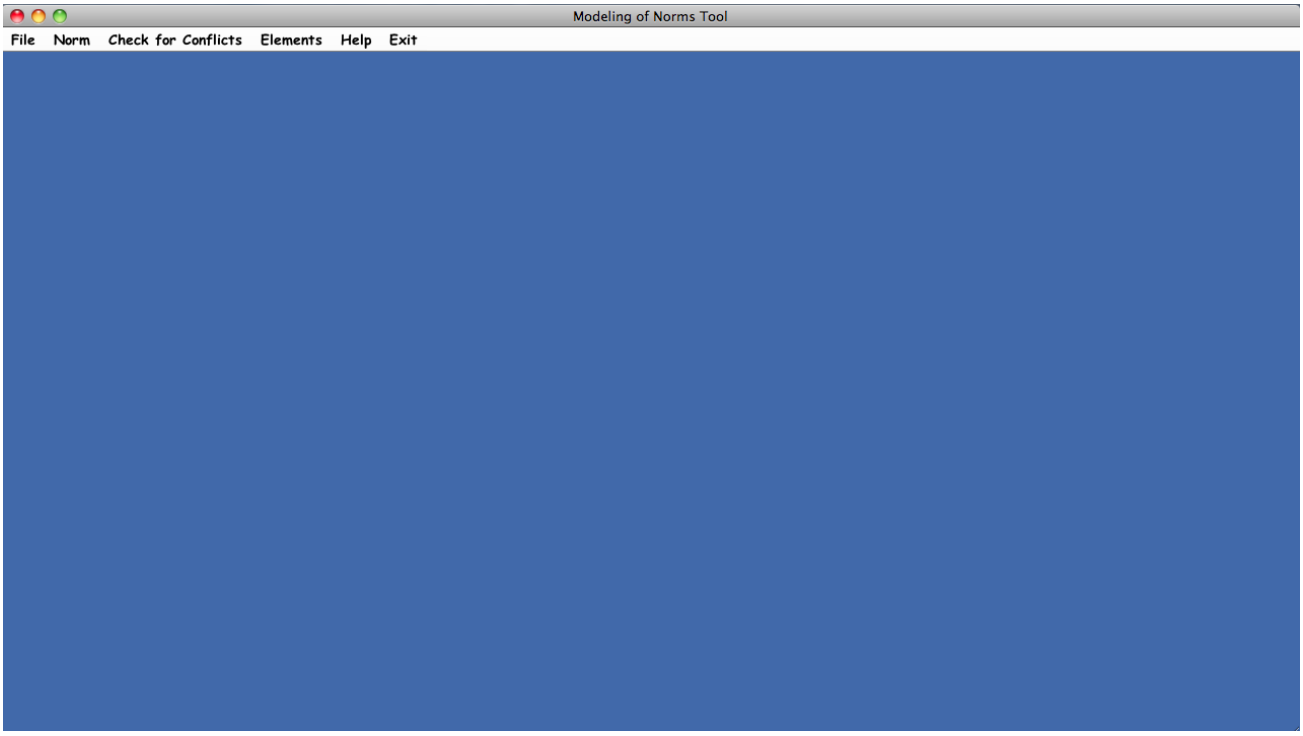

Figura 4.2 Menu da Ferramenta

Na primeira opção do menu, podemos salvar o modelo que está sendo criado, ou abrir um modelo criado em outro instante,como exemplificado na Figura 4.3.

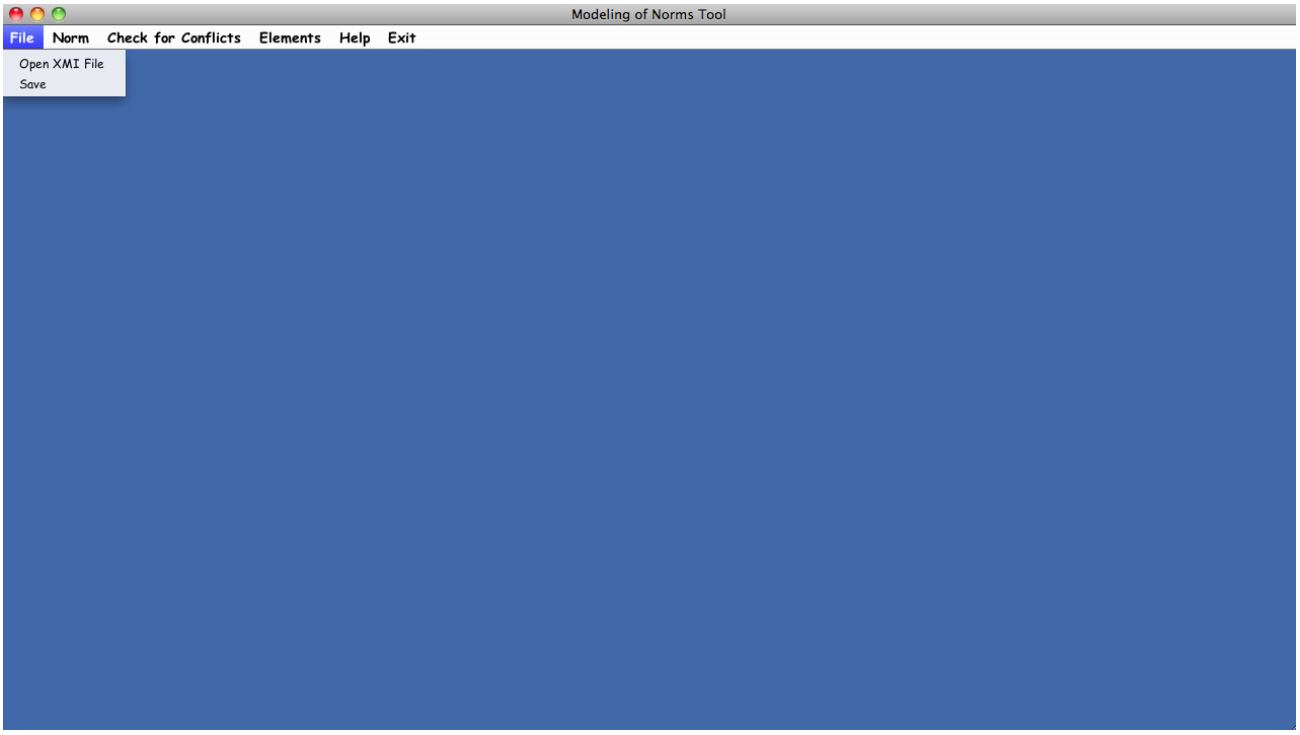

Figura 4.3 Menu File

Caso o usuário queira abrir um arquivo, uma nova tela abrirá para que ele possa digitar o nome do arquivo a ser aberto, Figura 4.4. Sendo que este arquivo estará no formato XMI.

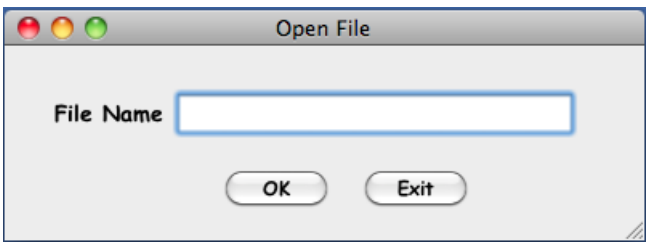

Figura 4.4 Open File

Caso não exista um arquivo com este nome, uma mensagem de erro é exibida, conforme Figura 4.5.

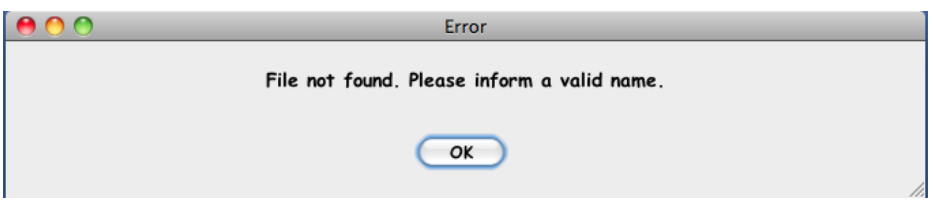

Figura 4.5 Erro Open File

No menu Norm podemos criar uma norma ou visualizar uma norma já criada, como ilustrado na Figura 4.6.

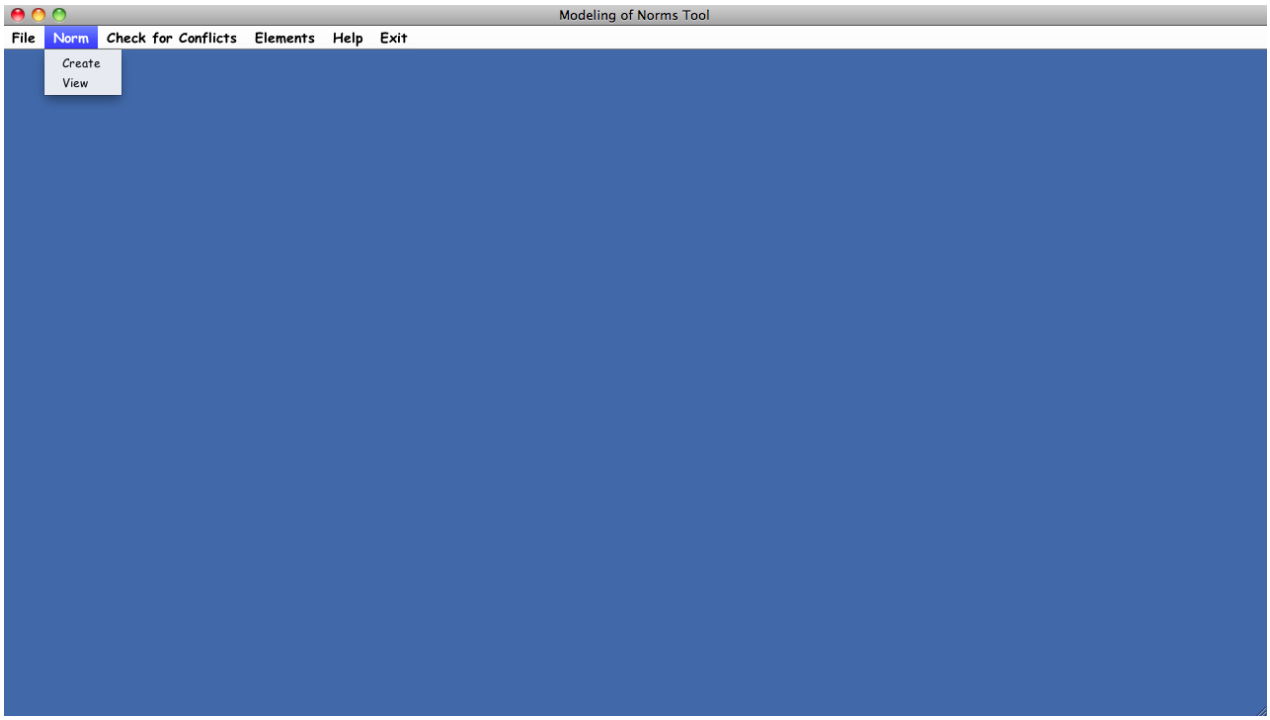

Figura 4.6 Menu Norm

Considerando que existe um modelo criado, é possível verificar a existência, ou não, de conflitos entre as normas contidas neste modelo, para isso basta escolher a opção Check for Conflicts presente no menu principal da ferramenta.

Além de criar normas, podemos criar elementos para o modelo, que poderão estar associados às normas modeladas. Conforme mostrado na Figura 4.7, podemos criar os seguintes elementos: *Environment, Organization, Agent, Role, Entity* e *Association End*.

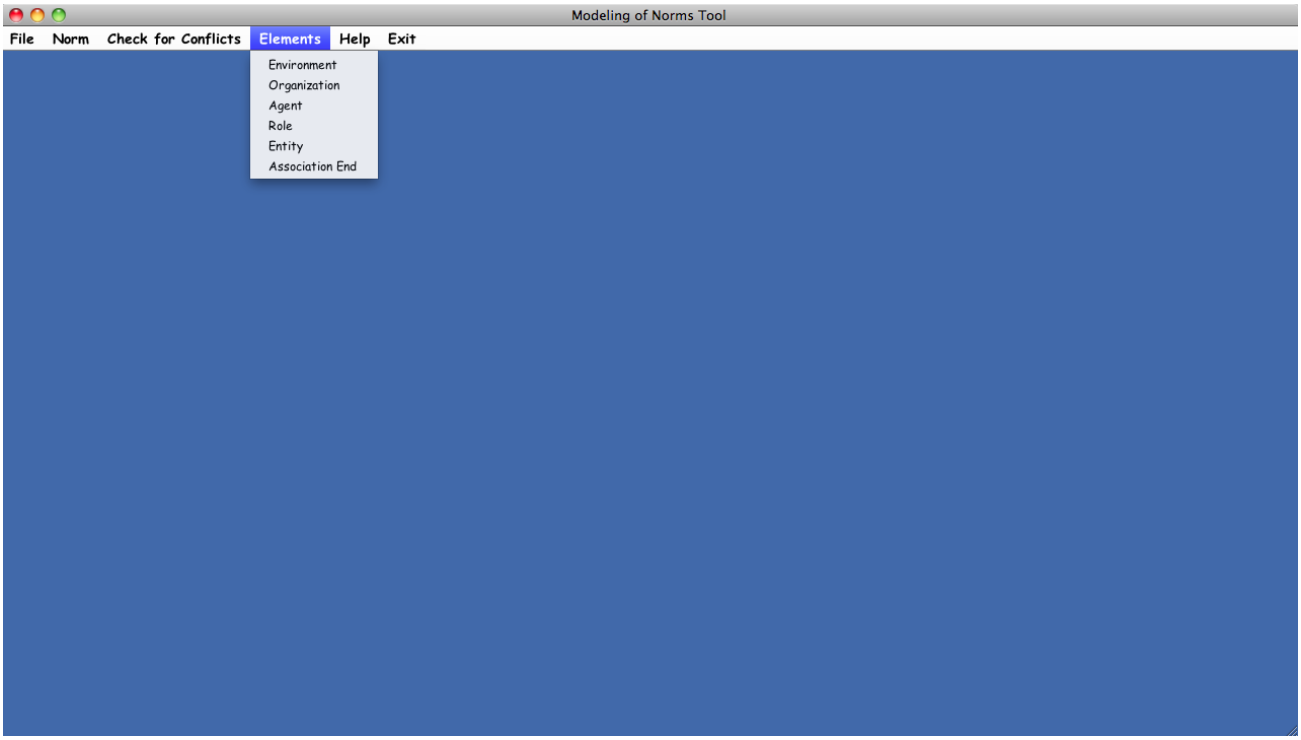

Figura 4.7 Menu Elements

Caso o usuário precise de ajuda no uso da ferramenta, existe a opção de, ao clicar em *Help*, abrir um Manual que possa auxiliar no uso da ferramenta. O usuário pode também desejar sair da ferramenta, para isso existe a opção *Exit*, que salva automaticamente o modelo que está sendo criado.

Para detalhar melhor a criação de uma norma, primeiramente precisamos fornecer um nome para a norma, em seguida escolher o tipo da norma (Permissão, Proibição ou Obrigação). Sendo assim podemos relacionar os elementos criados à norma, como mostrado na Figura 4.8. É importante destacar que para concluir a criação de uma norma é necessário associar a norma a um contexto, uma entidade e um recurso, pelo menos.

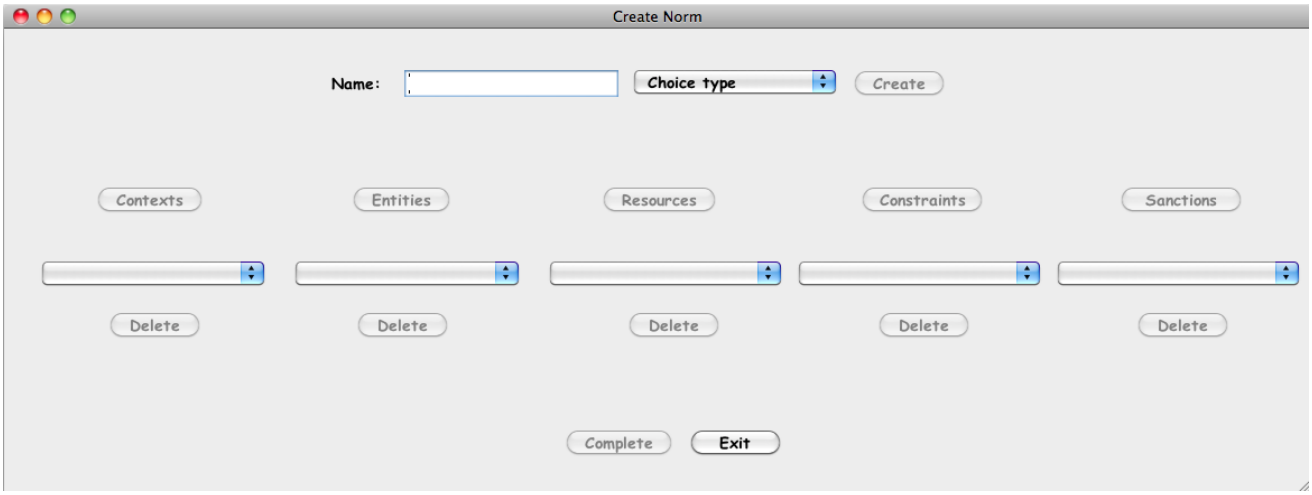

Figura 4.8 Criar uma Norma

Para associar um contexto, após clicar no botão *Contexts*, temos a opção de associar um ambiente ou uma organização, conforme Figura 4.9. Para isso, precisamos selecionar uma das opções no combo *Type*, e caso queira relacionar a um contexto existente o usuário escolhe em seguida o nome. Mas caso queira criar um novo contexto, após clicar em *Create New*, uma nova janela é aberta de acordo com a opção escolhida, o caso de *Environment* está ilustrado em Figura 4.10, e o de *Organization* em Figura 4.11.

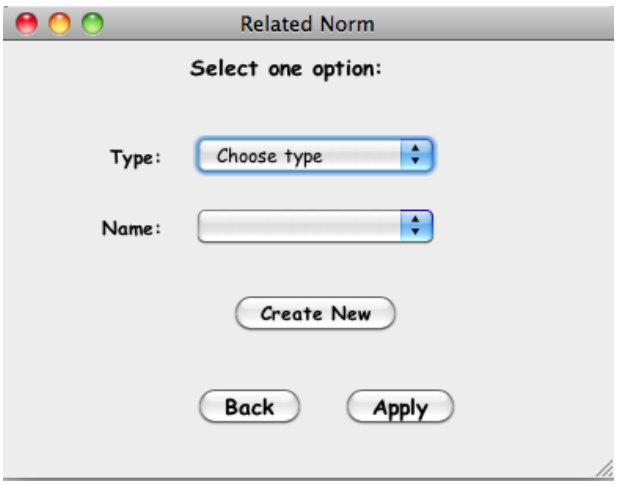

Figura 4.9 Associar a norma a um contexto

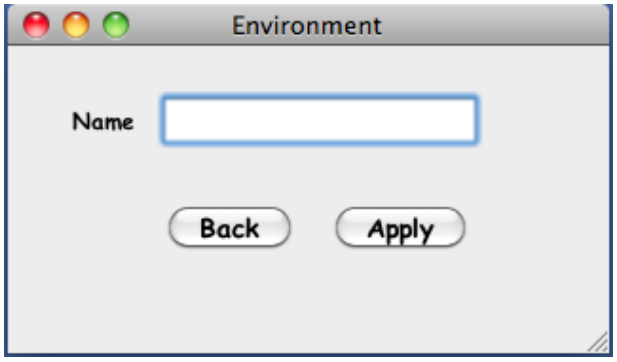

Figura 4.10 Criar um ambiente

Figura 4.11 Criar uma organização

Para associar entidades à norma, após clicar no botão *Entities* (Figura 4,8), temos a opção de associar um agente, um papel, uma organização, um ambiente, um agente exercendo um papel ou uma suborganização exercendo um papel, conforme Figura 4.9.

Neste caso, no combo *Type*, temos apenas as entidades que fazem parte do contexto da norma. Por exemplo, caso o contexto seja uma organização, só estarão disponíveis agentes dessa organização, papéis dessa organização e a própria organização e suas suborganizações. Caso o contexto seja um ambiente, só estarão disponíveis agentes desse ambiente, organizações desse ambiente, papéis das organizações desse ambiente, o próprio ambiente e outros ambientes (para permitir criar ambientes novos). Sendo assim, o usuário realiza a escolha da entidade e caso o usuário queira relacionar uma entidade existente, escolhe o nome da mesma.

Caso deseje associar uma nova entidade, de acordo com a opção escolhida temos as seguintes opções: *Environment* (Figura 4.10), *Organization* (Figura 4.11), *Agent* (Figura 4.12), *Role* (Figura 4.13), *Agente Playing a Role* (Figura 4.14) e

SubOrganization Playing a Role (Figura 4.15). Nos dois últimos casos, só podemos relaciona agentes, roles e suborganizações existentes.

Figura 4.12 Criar um agente

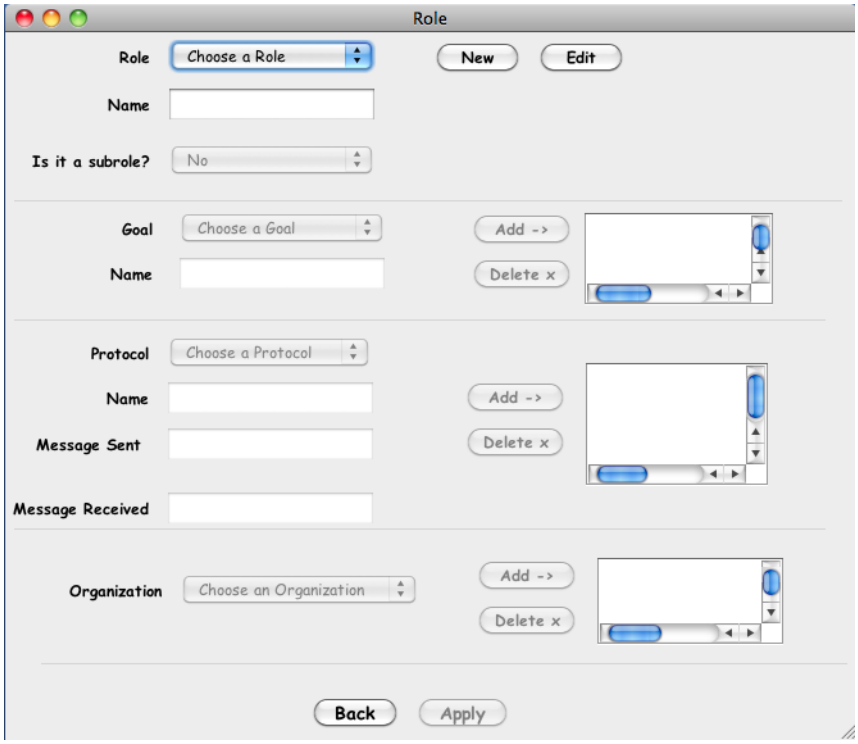

Figura 4.13 Criar um papel

Figura 4.14 Associar um agente exercendo um papel

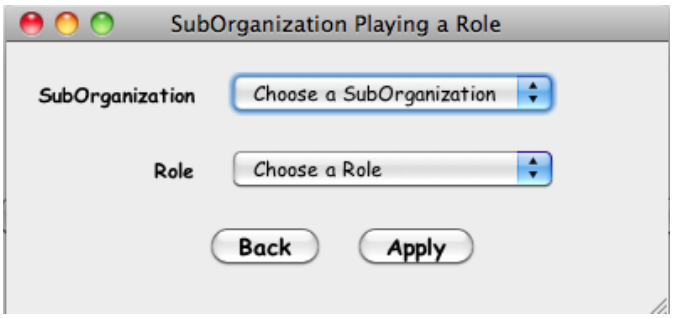

Figura 4.15 Associar uma suborganização exercendo um papel

Para associar recursos, após clicar no botão *Resources*, temos a opção de associar todos os recursos do meta-modelo com a norma, conforme Figura 4.16.

Neste caso, no combo *Resource Type*, temos todos os recursos descritos. Sendo assim, o usuário realiza a escolha do recurso e no combo *Object*, são listados os objetos, já existentes, do tipo escolhido. Em seguida o usuário deve escolher a ação desejada, no combo *Action*.

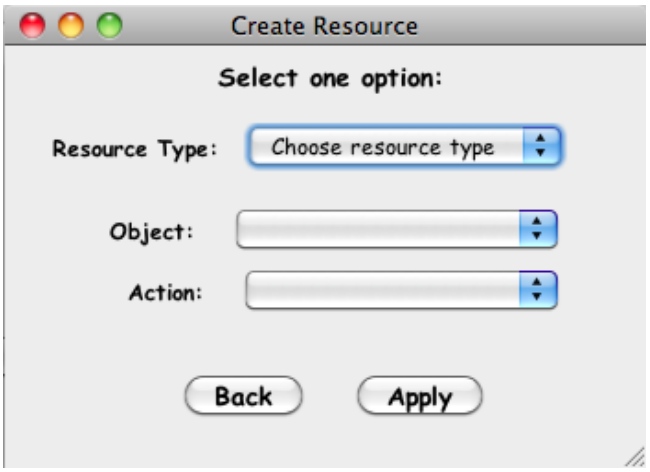

Figura 4.16 Associar um recurso

Caso a opção escolhida seja um atributo, temos uma tela diferenciada, conforme Figura 4.17. Nesta é necessário além das informações anteriormente descritas, selecionar a entidade “dona” do atributo.

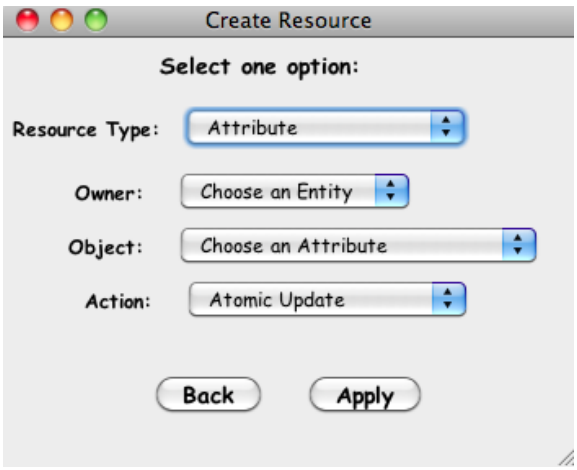

Figura 4.17 Associar um recurso - Atributo

Para uma restrição, após clicar no botão *Constraints*, temos a opção de associar uma restrição a norma, conforme Figura 4.18.

Neste caso, no combo *Type*, temos: *After* (*Action* e *Date*), *Before* (*Action* e *Date*), *Between* (*Action* e *Date*) e *If* (*Attribute, Belief* e *Date*). Sendo assim, o usuário realiza a escolha do tipo e em seguida a condição.

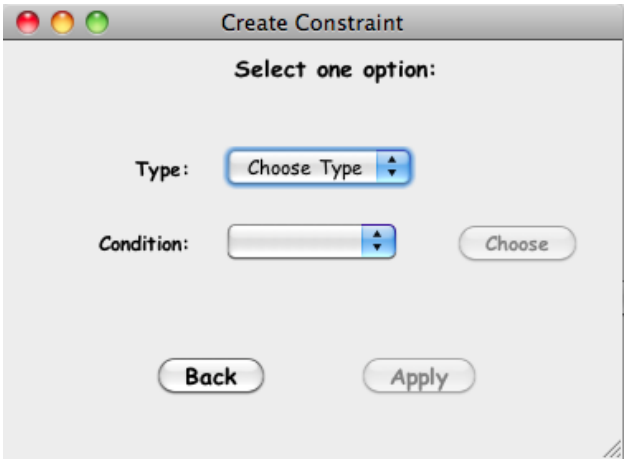

Figura 4.18 Associar uma restrição

Caso o usuário escolha *Between* temos as seguinte opções, conforme as Figuras 4.19 e 4.20. No caso de data e escolha de *After, Before* ou *If*, temos uma tela semelhante a Figura 4.19. Porém com apenas um campo de data. No caso de ação e escolha de *After* ou *Before*, temos uma tela parecida com a Figura 4.20. Porém sem a opção de escolher *After* ou *Before*.

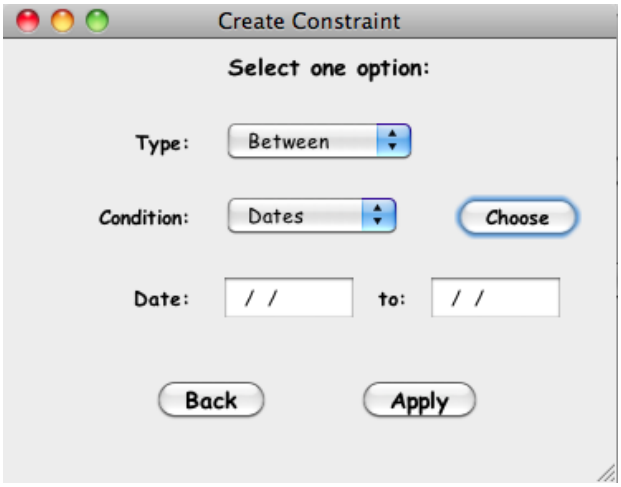

Figura 4.19 Restrição com data

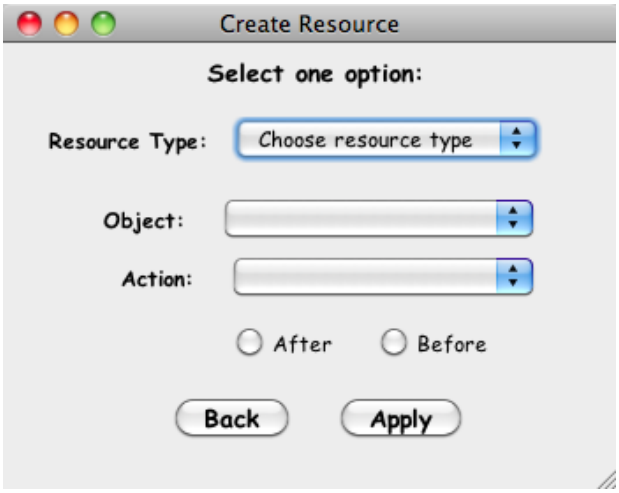

Figura 4.20 Restrição com ação

Caso o usuário escolha *If* além da opção de data, temos as seguintes opções, Attribute ou Belief. No caso de Belief, após escolher o tipo, devemos dizer o "dono" e o valor, e no caso do atributo, também um operador, como mostrado na Figura 4.21.

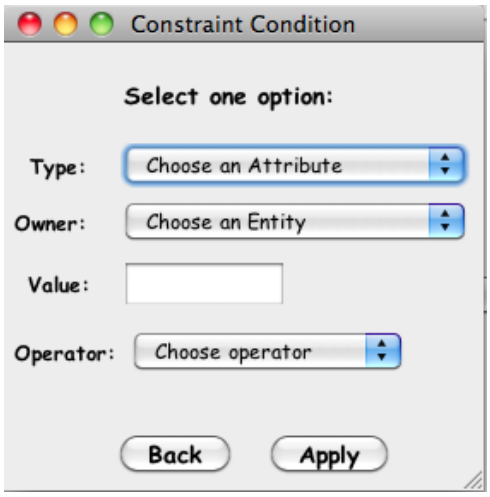

Figura 4.21 Restrição com atributo

Para associar uma sanção a norma, após clicar no botão *Sanctions*, temos a opção de associar uma punição ou uma recompensa, conforme Figura 4.22, e selecionar uma norma já criada. Podemos ainda criar uma nova norma para então definir uma sanção para a primeira norma. Para isso, basta clicar no botão Create.

Figura 4.22 Associar sanção à norma

Para detalhar a criação de elementos, no menu principal, falta apenas detalhar a criação de uma entidade e uma ponta de associação. Como mostrado nas figuras 4.23 e 4.24.

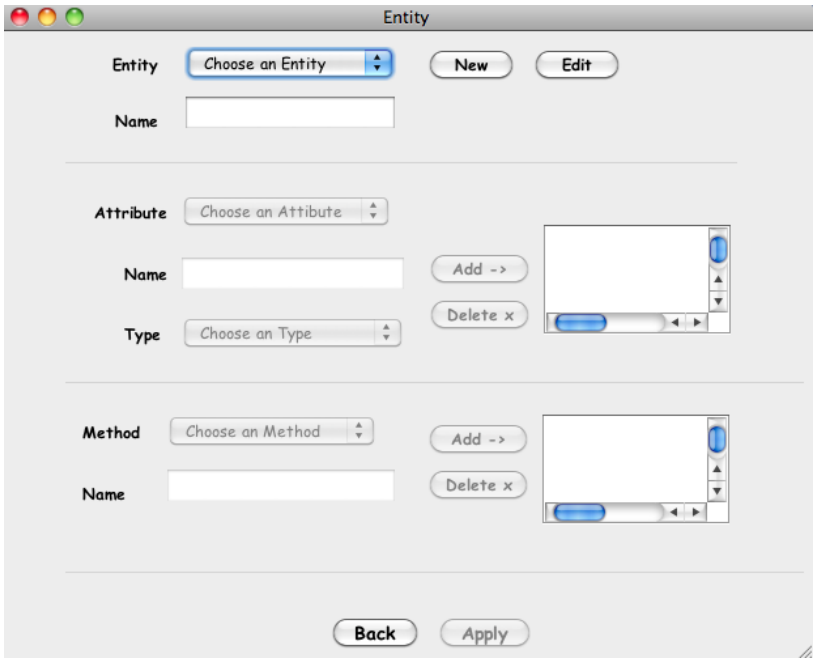

Figura 4.23 Criar uma Entidade

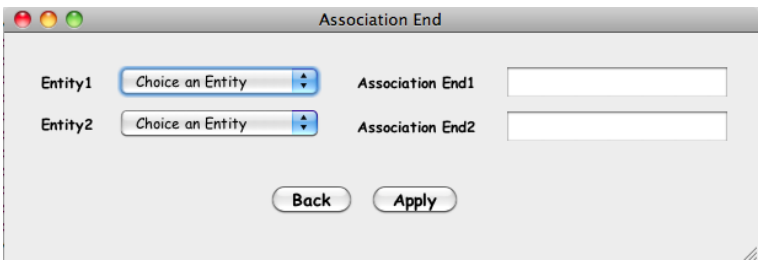

Figura 4.24 Criar uma ponta de associação

Conforme foi citado anteriormente, temos a opção de visualizar e editar uma norma, caso seja desejado, exemplificado na Figura 4.25.

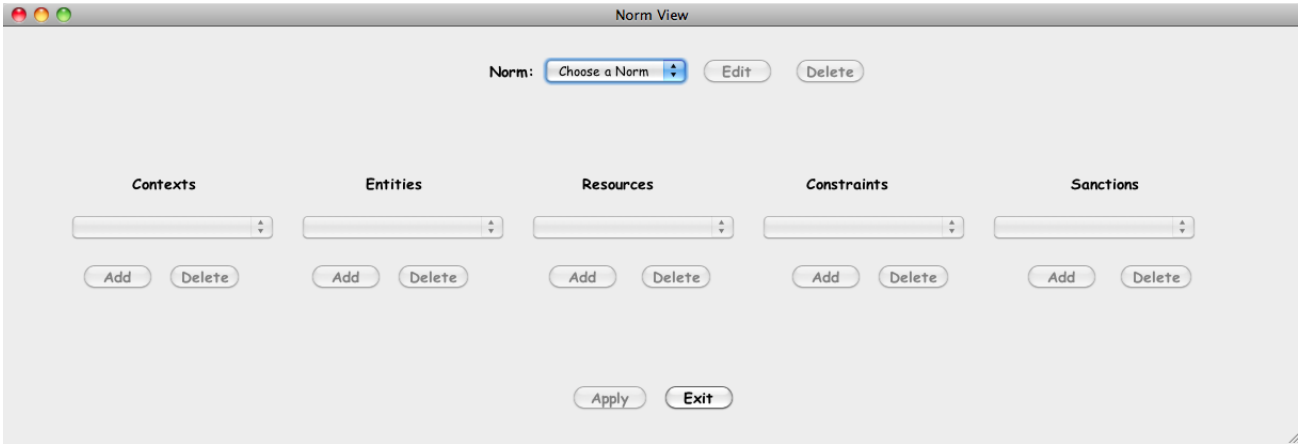

Figura 4.25 Visualizar uma norma

4.2.2 Gerando o XMI do modelo concreto

Após a criação das normas a ferramenta pode (fica a critério do usuário) gerar um arquivo com extensão XMI do modelo concreto da linguagem [Meta-modelo Concreto (2010)]. O modelo concreto se diferencia do modelo abstrato por representar os elementos criados na ferramenta de forma direta (como Normas, Agentes, Roles, Entidades, dentre outros) e não como instâncias destes elementos (Classes definidas no meta-modelo) como no modelo abstrato.

Para isso, após as definições do usuário, a ferramenta lê todas as listas de objetos e links que foram geradas anteriormente, criando assim, as *tags* dos elementos e de seus relacionamentos e as escrevendo no documento XMI. A estrutura do arquivo nos permite definir todas as entidades do Diagrama de Normas e os seus relacionamentos em *tags* específicas.

A fim de exemplificarmos essa transformação, detalharemos a seguir como ficaria o documento XMI para a norma NA2 já discutida na seção 3.3.

XMI do modelo concreto da linguagem

```
<?xml version="1.0" encoding="UTF-8"?>

<xmi:XMI xmlns:xmi="http://schema.omg.org/spec/XMI/2.1" xmi:version="2.1">

<NormDiagram name="NormsDiagram - UFF">

<environment name="UFF" />

<organization name="Computacao" />

<role name="Student" />

<Norm name="NA2-Role" type="Prohibition">
<resourceAction name="agentActionAction" resource="damageBook" actionType="AtoExe">
<agentActionExecuter name="Student" type="role" />
</resourceAction>
</Norm>

<Relationship name="RoleOfOrganization">
<element nameBegin="Computacao" associationEndBegin="organization"
 nameEnd="Student" associationEndEnd="role" />
</Relationship>

<Relationship name="OrganizationInhabitEnvironment">
<element nameBegin="UFF" associationEndBegin="environment"
 nameEnd="Computacao" associationEndEnd="organization" />
</Relationship>

<Relationship name="NormInContextOrganization">
<element nameBegin="NA2-Role" associationEndBegin="normInContext"
 nameEnd="Computacao" associationEndEnd="context" />
</Relationship>

<Relationship name="NormAssignmentRole">
<element nameBegin="Student" associationEndBegin="restrictRoleBehavior"
 nameEnd="NA2-Role" associationEndEnd="hasNorm" />
</Relationship>

<Relationship name="ResourceAssignmentNorm">
<element nameBegin="damageBook" associationEndBegin="access"
 nameEnd="NA2-Role" associationEndEnd="isAssigned" />
</Relationship>

</NormDiagram>

</xmi:XMI>
```

Em seguida, o transformador 1, [Transformador (2010)] que está implementado através da linguagem XSLT é executado com a finalidade de gerar um segundo arquivo, a partir do primeiro, também com extensão XMI, porém do modelo abstrato da linguagem.

A linguagem XSLT é um padrão para transformação de documentos que recebe como entrada um documento XML e gera na saída um outro documento em formato texto. No contexto deste trabalho, o transformador recebe como entrada um arquivo XMI que contém os modelos concretos das normas do sistema de acordo com o meta-modelo concreto da linguagem.

Posteriormente, as regras de transformação implementadas em XSLT são executadas, seguindo as definições de transformação dos modelos da linguagem NormML [Regras de Transformação - Transformador (2010)] e gerando desta forma como saída, um arquivo XMI nos padrões da UML 2.1 contendo os modelos abstratos das normas.

4.3 VALIDANDO AS NORMAS

Para a validação das normas, foi utilizado um componente Java denominado EOS que será melhor detalhado na seção 4.3.1.

4.3.1 Introdução a API EOS

É uma API descrita em Java que possibilitar a execução de operações OCL sobre diagramas de objetos instancias de diagramas de classes. Um componente, avaliador da linguagem OCL, foi projetado com o objetivo de realizar uma avaliação eficaz das expressões OCL sobre cenários de tamanho médio-grandes. Este componente foi denominado EOS.

Três aspectos foram estudados no desenvolvimento da EOS:

- A necessidade de uma implementação eficiente em OCL, a fim de lidar com os novos usos da língua;

- Os aspectos considerados para melhorar a eficiência do avaliador EOS em cenários médios e grandes;

- Os limites das implementações OCL atuais para lidar com cenários muito grandes.

O componente EOS não se baseia em um *framework* de modelagem (meta) em particular. Sua interface pública fornece métodos para criação dos elementos de um modelo através da definição de um modelo de classes, para criação dos elementos que são as instâncias através da definição de um modelo de objetos e para a definição de operações definidas utilizando os elementos do modelo e executas para a verificação das instâncias. Na modelagem do primeiro diagrama definimos as classes, descrevendo seus nomes; seus atributos (onde referenciamos a classe que contém este atributo, o nome e o tipo do atributo); as associações (onde definimos as classes associadas e as cardinalidades); e as generalizações (onde são definidos a que estamos generalizando e o nome da generalização). Com isso é possível projetar a estrutura de dados da EOS para armazenar internamente modelos de usuários e cenários, de forma que as propriedades dos objetos pudessem ser acessadas de maneira eficiente.

De modo geral, a implementação da EOS é bastante simples: expressões OCL de navegação são executadas usando loops For/while do Java; e operações OCL padrão, quando possível, são executadas utilizando as funções adequadas Java. Finalmente, as expressões são avaliadas através do componente EOS e uma estratégia é seguida: em particular, expressões coletivas são totalmente avaliados e seus elementos resultantes são alocados na memória.

Com o objetivo de exemplificar o uso de EOS, considere o modelo da figura 2.2. Este modelo foi representado em EOS através da criação de um diagrama de classes como apresentado no quadro.

Porém temos uma ressalva a fazer para a utilização da EOS nesta modelagem, já que nesse componente não existe a possibilidade de criar “classe de associação”. Para este exemplo, não iremos detalhar a solução proposta para a modelagem em EOS de “classes de associação”. Isso será feito na seção 4.3.3.

```
public class Metamodel {

        public static void createMetaModel(final IEOS environment) {

        //Creating the Metamodel classes
        environment.createClassDiagram();
        environment.insertClass("Pessoa");
        environment.insertClass("Empresa");
        environment.insertClass("Emprego");
        …

        // Attribute
        environment.insertAttribute("Pessoa", "eCasado", "Boolean");
        environment.insertAttribute("Pessoa", "idade", "Integer");
        environment.insertAttribute("Pessoa", "prenome", "String");
        environment.insertAttribute("Pessoa", "sobrenome", "String");

        environment.insertAttribute("Empresa", "nome", "String");

        environment.insertAttribute("Emprego", "cargo", "String");
        environment.insertAttribute("Emprego", "dataInicio", "Date");
        environment.insertAttribute("Emprego", "salario", "Real");
        …

        //Associations
        environment.insertAssociation("Pessoa", "Marido", "0..1", "0..1", "Mulher", "Pessoa");
        environment.insertAssociation("Pessoa", "Empregado", "0..*", "0..*", "Empregador",
"Empresa") ;
        …

        }

}
```

Para modelar o modelo de objetos descrevemos a inserção dos objetos, (que são instâncias das classes), a criação de *links* entre as instâncias como também o valor de atributos, caso haja. No quadro a seguir criamos os objetos {Mariana, Gabriela, Joaquim} e {Vale, Accenture} instâncias das classes Pessoa e Empresa, respectivamente. E relacionamos os objetos Mariana e Joaquim; Mariana e Accenture; e Gabriela e Vale.

```
public static void createObjectDiagram(final IEOS environment){

            //Objects
```

```
            environment.insertObject("Pessoa", "Mariana");
             environment.insertObject("Pessoa","Gabriela");
            environment.insertObject("Pessoa","Joaquim");
             environment.insertObject("Empresa","Vale");
            environment.insertObject("Empresa","Accenture");
            …

            //Attribute Objects
            environment.insertValue("Pessoa", "idade", "Mariana", "23");
            environment.insertValue("Pessoa", "eCasado", "Mariana", "True");

            environment.insertValue("Pessoa", "idade", "Gabriela", "22");
            environment.insertValue("Pessoa", "eCasado", "Gabriela", "False");

            environment.insertValue("Pessoa", "idade", "Joaquim", "28");
            environment.insertValue("Pessoa", "eCasado", "Joaquim", "True");
            …

           //Links
           environment.insertLink("Pessoa", "Mariana", "Empregado","Empregador", "Accenture",
"Empresa");
           environment.insertLink("Pessoa", "Gabriela", "Empregado", "Empregador", "Vale",
"Empresa");
           environment.insertLink("Pessoa", "Mariana", "Mulher", "Marido", "Joaquim", "Pessoa");
           …

}
```

Para descrever as operações definimos primeiramente parâmetros, se necessário, e em seguida definimos as operações em si. Nas operações declaramos o contexto da operação, o nome, a operação, e o parâmetro, caso haja. No quadro estamos criando a operação para garantir que uma pessoa não pode ter um Marido e uma Mulher ao mesmo tempo e uma operação que retorna o conjunto de empresas em que uma determinada pessoa trabalha, podendo esse ser nulo.

```
…
{
environment.insertOperation("Pessoa","verifyAssociationMaridoMulher", "Boolean", ” (((self.mulher-
>isEmpty()) and (self.marido->isEmpty())) or
 ((self.mulher->notEmpty()) and (self.marido->isEmpty()))
 or ((self.mulher->isEmpty()) and (self.marido->notEmpty()))))” ,
new Object[0]);

environment.insertOperation("Pessoa","returnCompanies", "Set(Empresa)", ” self.empresa”, new
Object[0]);
…
```

Para executar as operações devemos chamar a operação em um método de *call*, como feito a seguir:

```
public static void callOperations (final IEOS environment){

        String answer = null;


        try{

            answer = environment.query("Mariana.verifyAssociationMaridoMulher()");
            System.out.println(answer);
            System.out.println();

            answer = environment.query("Gabriela.returnCompanies()");
            System.out.println(answer);
            System.out.println();
…


        } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }

}
```

Podemos ainda chamar uma operação dentro de outra operação, como exemplificado. A operação *gotAJob* chama a operação *returnCcompanies*.

```
…
        try
        {

        environment.insertOperation("Pessoa","gotAJob","Boolean",
        "if (self.returnCompanies()->notEmpty()) then (true) else (false)" ,new Object[0]);
…
        } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
                System.exit(1) ;
        }
```

Suponha que estamos aplicando sobre o objeto *Mariana,* descrito no diagrama de objetos criado acima, a operação *returnCompanies.* Neste caso, como só havia um *link* entre o objeto *Mariana* (instância de Pessoa) e outro objeto *Accenture* (instância de Empresa), teremos como resultado o conjunto com um único elemento {Accenture}.

Considere agora a operação *gotAJob,* que possui como objetivo verificar se a pessoa esta empregada ou não. Neste exemplo, obtemos o resultado *true* para a

operação já que o conjunto retornado pela operação *returnCompanies* não é um conjunto vazio.

4.3.2 Do modelo abstrato para a validação

Conforme já descrito na seção 4.2.2, o Transformador 1 gera um segundo arquivo com extensão XMI. Este consiste em uma representação do modelo abstrato da linguagem. Este arquivo segue o padrão UML 2.1 e, assim como o primeiro, possui todas as informações sobre o modelo inserido pelo usuário na ferramenta. Após a sua criação, a ferramenta executa um método para a leitura desse arquivo. O objetivo desta etapa é ler e organizar os dados contidos no arquivo em estruturas de listas que serão, posteriormente, lidas para a criação dos elementos na EOS.

Assim, para cada elemento lido é gerado uma linha de código que ao ser processada criará um objeto, ou um link com o auxílio do componente EOS. Estes elementos darão origem ao Diagrama de Objetos. A utilização da EOS na ferramenta é melhor detalhada na seção 4.3.3.

Para um melhor entendimento, segue um exemplo do documento XMI, já transformado, para a norma NA2 já discutida na seção 3.3.

XMI do modelo abstrato da linguagem

```
<?xml version="1.0" encoding="UTF-8"?>

<uml:package>

<packagedElement type="uml:InstanceSpecification"
id="U-ObjEnvUFF" uuid="-ObjEnvUFF" name="UFF"
classifier="U-Env"/>

<packagedElement type="uml:InstanceSpecification"
id="U-ObjOrgComputer Departament" uuid="-ObjOrgComputer Departament"
name="Computer Departament" classifier="U-Org"/>

<packagedElement type="uml:InstanceSpecification"
id="U-ObjRolStudent" uuid="-ObjRolStudent"
name="Student" classifier="U-Rol"/>
```

```
<packagedElement type="uml:InstanceSpecification"
id="U-ObjAgeActdamageBook" uuid="-ObjAgeActdamageBook"
name="damageBook" classifier="U-AgeAct"/>

<packagedElement type="uml:InstanceSpecification"
id="U-ObjNorN2A-Role" uuid="-ObjNorProN2A-Role"
name="N2A-Role" classifier="U-NorPro"/>

<packagedElement type="uml:InstanceSpecification"
id="U-ObjIfN2A-Role" uuid="-ObjIfN2A-Role"
name="N2A-RoleIf" classifier="U-If"/>

<packagedElement type="uml:InstanceSpecification"
id="U-ObjAftN2A-Role" uuid="-ObjAftN2A-Role"
name="N2A-RoleAfter" classifier="U-Aft"/>

<packagedElement type="uml:InstanceSpecification"
id="U-ObjBefN2A-Role" uuid="-ObjBefN2A-Role"
name="N2A-RoleBefore" classifier="U-Bef"/>

<packagedElement type="uml:InstanceSpecification"
id="U-ObjBetN2A-Role" uuid="-ObjBetN2A-Role"
name="N2A-RoleBetween" classifier="U-Bet"/>

<packagedElement type="uml:InstanceSpecification"
id="U-ObjAtoExedamageBookN2A-Role"
uuid="-ObjAtoExedamageBookN2A-Role"
name="damageBookAtoExe" classifier="U-AtoExe"/>

<packagedElement type="uml:InstanceSpecification"
id="U-N2A-RoledamageBookAtoExe"
uuid="-U-N2A-RoledamageBookAtoExe"
classifier="U-NorAct">
<Extension extender="UModel">
<attributeExtension
instanceSpecification_LinkBegin="U-ObjNorN2A-Role"
 instanceSpecification_LinkEnd="U-ObjAtoExedamageBookN2A-Role"/>
</Extension>
</packagedElement>

<packagedElement type="uml:InstanceSpecification"
 id="U-N2A-RoledamageBookAtoExedamageBook"
 uuid="-U-N2A-RoledamageBookAtoExedamageBook"
classifier="U-ActRes">
<Extension extender="UModel">
<attributeExtension
instanceSpecification_LinkBegin="UObjAtoExedamageBookN2A-Role"
instanceSpecification_LinkEnd="U-ObjAgeActdamageBook"/>
</Extension>
</packagedElement>

<packagedElement type="uml:InstanceSpecification"
id="U-StudentComputer Departament"
uuid="-U-StudentComputer Departament"
classifier="U-RolOrg">
<Extension extender="UModel">
<attributeExtension
instanceSpecification_LinkBegin="U-ObjRolStudent"
```

```
instanceSpecification_LinkEnd="U-ObjOrgComputer Departament"/>
</Extension>

</packagedElement>
<packagedElement type="uml:InstanceSpecification"
id="U-UFFComputer Departament"
uuid="-U-UFFComputer Departament"
classifier="U-EnvOrg">
<Extension extender="UModel">
<attributeExtension
instanceSpecification_LinkBegin="U-ObjEnvUFF"
instanceSpecification_LinkEnd="U-ObjOrgComputer Departament"/>
</Extension>
</packagedElement>

<packagedElement type="uml:InstanceSpecification"
id="U-N2A-RoleComputer Departament"
uuid="-U-N2A-RoleComputer Departament"
classifier="U-NorOrg2">
<Extension extender="UModel">
<attributeExtension
instanceSpecification_LinkBegin="U-ObjNorN2A-Role"
instanceSpecification_LinkEnd="U-ObjOrgComputer Departament"/>
</Extension>
</packagedElement>

<packagedElement type="uml:InstanceSpecification"
id="U-StudentN2A-Role"
uuid="-U-StudentN2A-Role"
classifier="U-RolNor">
<Extension extender="UModel">
<attributeExtension
instanceSpecification_LinkBegin="U-ObjRolStudent"
instanceSpecification_LinkEnd="U-ObjNorN2A-Role"/>
</Extension>
</packagedElement>
</uml:package>
```

4.3.3 Uso da EOS na Ferramenta

Como detalhado anteriormente, a ferramenta EOS utilizada no nosso trabalho, é um componente Java eficiente para avaliação de OCL, linguagem em que descrevemos as operações da ferramenta para modelagem de normas em sistemas multiagentes e para verificação de conflitos entre as normas.

O mecanismo da EOS para a descrição de diagrama de classes foi utilizado para descrever o meta-modelo de NormML. E o aparato para descrição de diagrama de objetos foi utilizado para a instanciação das normas. O diagrama de classe definido segue o meta-modelo apresentado na seção 3.2. Neste somente algumas classes e relacionamentos foram descritos.

```
Public class Metamodel {

        public static void createMetaModel(final IEOS environment) {

        //Creating the Metamodel classes
        environment.createClassDiagram();
         environment.insertClass("Role");
         environment.insertClass("Norm");
         environment.insertClass("NormObligation");
         environment.insertClass("NormProhibition");
         environment.insertClass("NormPermission");
        environment.insertClass("Action");
         environment.insertClass("AgentAction");
        environment.insertClass("AtomicAction");
        environment.insertClass("CompositeAction");
        environment.insertClass("NormConstraint");
        environment.insertClass("Before");
        environment.insertClass("AtomicExecute");
        environment.insertClass("Date");

        …

         // Attribute
        environment.insertAttribute("Role", "Default", "Boolean");
        environment.insertAttribute("Date", "ConstraintDate", "Integer");
        environment.insertAttribute("Date", "ConstraintMonth", "Integer");
        environment.insertAttribute("Date", "ConstraintYear", "Integer");
        …

        //Associations
        environment.insertAssociation("Action", "action", "1..*", "1", "resource", "Resource");

        environment.insertAssociation("AgentAction", "actionToBeExecuted", "0..*",
        "0..*", "roleWillExecute", "Role");

        environment.insertAssociation("AgentAction", "actionExecuted", "0..*",
        "0..*", "roleExecutedAction", "Role");

        environment.insertAssociation("Norm", "isAssigned", "0..*", "1..*", "access", "Action");

        environment.insertAssociation("Role", "restrictRoleBehavior", "0..*", "0..*",
        "hasNorm",          "Norm");

        environment.insertAssociation("Before", "beforeConstraint3", "0..*", "0..*",
        "beforeDate3", "Date");

         environment.insertAssociation("NormConstraint", "constraint", "0..*", "0..*",
         "isConstraintBy", "Norm");
        …
```

```
        //Generalizations
        environment.insertGeneralization("NormPermission", "Norm");
        environment.insertGeneralization("NormObligation", "Norm");
        environment.insertGeneralization("NormProhibition", "Norm");

        environment.insertGeneralization("Before", "NormConstraint");


        environment.insertGeneralization("AtomicAction", "Action");
        environment.insertGeneralization("CompositeAction", "Action");


        environment.insertGeneralization("AtomicExecute", "AtomicAction");

        …

        }

}
```

Como contorno do problema de criação de “classe de associação” que foi introduzido na subseção 4.3.1, no momento em que o usuário finaliza a criação de uma norma, criamos cópias deste elemento quando este estiver restringindo ao mesmo tempo o comportamento de um agente, uma role, um agente exercendo uma role, uma organização, uma suborganização exercendo uma role ou um ambiente.

Em seguida, associamos as normas aos elementos, estabelecendo links entre eles. Por exemplo, para uma norma associada a um papel e a um agente exercendo um papel, criamos duas normas. Estas normas devem ser exatamente iguais, ou seja, com as mesmas relações de contexto, entidades, recursos envolvidos, restrições e sanções.

Logo depois, inserimos um link entre a primeira norma e o papel e dois outros links entre a segunda norma e o agente e ao papel que este exerce. Além destes, são associados o agente e o papel exercido por ele. Desta forma, quando forem gerados os objetos e os links na EOS, será possível identificar o estabelecimento desse tipo de relação.

Para exemplificar a criação do diagrama de objetos e instanciar o meta-modelo usaremos as normas descritas na subseção 3.3. A esse diagrama adicionamos um conjunto de operações que representam as regras de boa formação das normas e outro conjunto de operações utilizadas para a verificação de conflitos entre as mesmas. Para demonstrar a utilização destas operações, descrevemos no

quadro como as regras e as operações de conflito apresentadas nas seções 3.4 e 3.5 foram implementadas.

```
public static void createObjectDiagram(final IEOS environment){

                environment.insertObject("NormObligation","NA1");
               environment.insertObject("NormProhibition","NA2");
                environment.insertObject("Role","Student");
                environment.insertObject("AgentAction","returnBook");
                environment.insertObject("AgentAction","damageBook");
                environment.insertObject("AtomicExecute","executeReturnBook");
                environment.insertObject("AtomicExecute","executeDamageBook");
                environment.insertObject("Before","beforeDevolutionDate");
                environment.insertObject("Date","devolutionDate");
…
               environment.insertValue("Date", "ConstraintDate", "devolutionDate", "20");
                environment.insertAttribute("Date", "ConstraintMonth", "devolutionDate", "6");
                environment.insertAttribute("Date", "ConstraintYear", "devolutionDate", "2010");


…
//Links Figura 3.8

environment.insertLink("Norm","NA2","hasNorm","restrictRoleBehavior","Student","Role");

environment.insertLink("AgentAction","damageBook"," actionToBeExecuted "," roleWillExecute ",
"Student","Role");

environment.insertLink("AgentAction ","damageBook"," actionExecuted ","roleExecutedAction",
"Student","Role");

environment.insertLink("AgentAction ","damageBook","resource","action",
 " executeDamageBook ","AtomicExecute");

environment.insertLink("Norm ","NA2","isAssigned","access",
" executeDamageBook ","AtomicExecute");


//Links Figura 3.9

environment.insertLink("AgentAction","returnBook"," actionToBeExecuted "," roleWillExecute ",
"Student","Role");

environment.insertLink("AgentAction ","returnBook"," actionExecuted ","roleExecutedAction",
"Student","Role");

environment.insertLink("AgentAction ","returnBook","resource","action",
 " executeReturnBook ","AtomicExecute");

environment.insertLink("Norm ","NA1","isAssigned","access",
" executeReturnBook ","AtomicExecute");

environment.insertLink("Norm ","NA1"," isConstraintBy "," constraint ",
" beforeDevolutionDate ","Before");

environment.insertLink("Date,"devolutionDate"," beforeDate3 "," beforeConstraint3 ",
" beforeDevolutionDate ","Before");
```

```
…
}
```

Para descrever as operações, fazemos conforme detalhado anteriormente na subseção 4.3.1.

```
...
// Regras de boa formação usadas no exemplo, seção 3.4

//Regra: Uma norma deve estar no contexto de uma organização ou de um ambiente.

environment.insertOperation("Set(Norm)","rule2","Boolean",
"self->forAll(norma:Norm|((((norma.context1)->isEmpty()) and ((norma.context2)->notEmpty())) or
(((norma.context1)->notEmpty()) and ((norma.context2)->isEmpty())))" ,new Object[0]);

//Regra: Uma norma deve restringir um agente, um papel, um agente exercendo um papel, um
ambiente, uma organização ou uma suborganização exercendo um papel.

environment.insertOperation("Set(Norm)","rule1","Boolean",
self->forAll(norma:Norm|if((norma.restrictAgentBehavior)->
isEmpty())then(((norma.restrictRoleBehavior)-> notEmpty())or((norma.restrictOrganization)->
notEmpty())or((norma.restrictEnvironmentBehavior)->
notEmpty()))else(if((norma.restrictRoleBehavior)-> isEmpty())then(((norma.restrictAgentBehavior)->
notEmpty())or((norma.restrictOrganization)-> notEmpty())or((norma.restrictEnvironmentBehavior)->
notEmpty()))else(if((norma.restrictOrganization)-> isEmpty())then(((norma.restrictAgentBehavior)->
notEmpty())or((norma.restrictRoleBehavior)-> notEmpty())or((norma.restrictEnvironmentBehavior)->
notEmpty()))else(if((norma.restrictEnvironmentBehavior)->
isEmpty())then(((norma.restrictAgentBehavior)-> notEmpty())or((norma.restrictRoleBehavior)-
>notEmpty())or((norma.restrictOrganization)-> notEmpty()))else(if(((norma.restrictAgentBehavior)->
notEmpty())and((norma.restrictRoleBehavior)-> notEmpty())and((norma.restrictOrganization)->
notEmpty())and((norma.restrictEnvironmentBehavior)->
notEmpty()))then(true)else(false)endif)endif)endif)endif)endif)" ,new Object[0]);
…

//Regra de conflito utilizada no exemplo, seção 3.5

//A fim de verificarmos se as normas estão em situação de potencial conflito, utilizaremos a regra de
Conceito Deôntico já descrita na seção 3.5.3:

environment.insertOperation("Norm", "deonticInconsistence", "Boolean",
"((((self.oclIsTypeOf(NormObligation))and(n2.oclIsTypeOf(NormPermission)))or((n2.oclIsTypeOf(Norm
Obligation))and(self.oclIsTypeOf(NormPermission))))or(((self.oclIsTypeOf(NormObligation))and(n2.oclI
sTypeOf(NormProhibition)))or((n2.oclIsTypeOf(NormObligation))and(self.oclIsTypeOf(NormProhibition
))))or(((self.oclIsTypeOf(NormPermission))and(n2.oclIsTypeOf(NormProhibition)))or((n2.oclIsTypeOf(N
ormPermission))and(self.oclIsTypeOf(NormProhibition)))))", PNorm2);
...
```

Aplicando aos dois objetos, NA1 e NA2, a primeira regra de boa formação (*rule 2*), temos o resultado falso já que nenhuma das duas possui um relacionamento de contexto estabelecido. Em um segundo momento, aplicando a segunda regra

(*rule 1*) sobre os objetos, o retorno também será falso, já que NA1 não restringe o comportamento de nenhuma entidade.

Para execução da operação supracitada que verifica conflitos, consideramos todas as normas criadas: NA1 e NA2 (na chamada da operação basta utilizar *Norm.allInstances*). Nesta circunstância, teremos como consequência *false*, pois as normas não se encaixam nas situações de conflito descritas na subseção 3.5.3, onde o conceito deôntico é esmiuçado. Se for aplicada apenas a operação de Conceito Deôntico (deonticInconsistence), o retorno será verdadeiro, demonstrando assim, que as normas estão em situação de potencial conflito (Proibição e Obrigação).

4.4 RESPOSTA AO USUÁRIO

Após a criação do modelo na ferramenta, o usuário pode realizar a verificação de conflitos no mesmo e obter como resposta um relatório, como representado nas Figuras 4.26 e 4.27, que contém com detalhe as regras de boa formação, informando se foram ou não cumpridas, bem como a informação se houve, ou não conflito entre as normas modeladas.

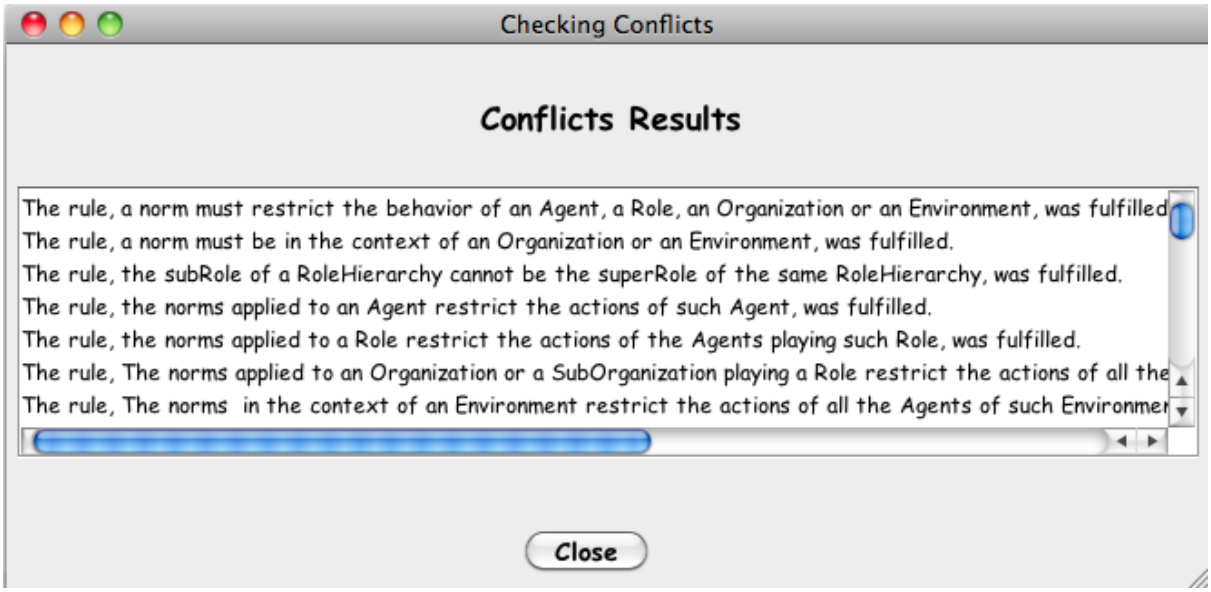

Figura 4.26 Feedback ao usuário - regra

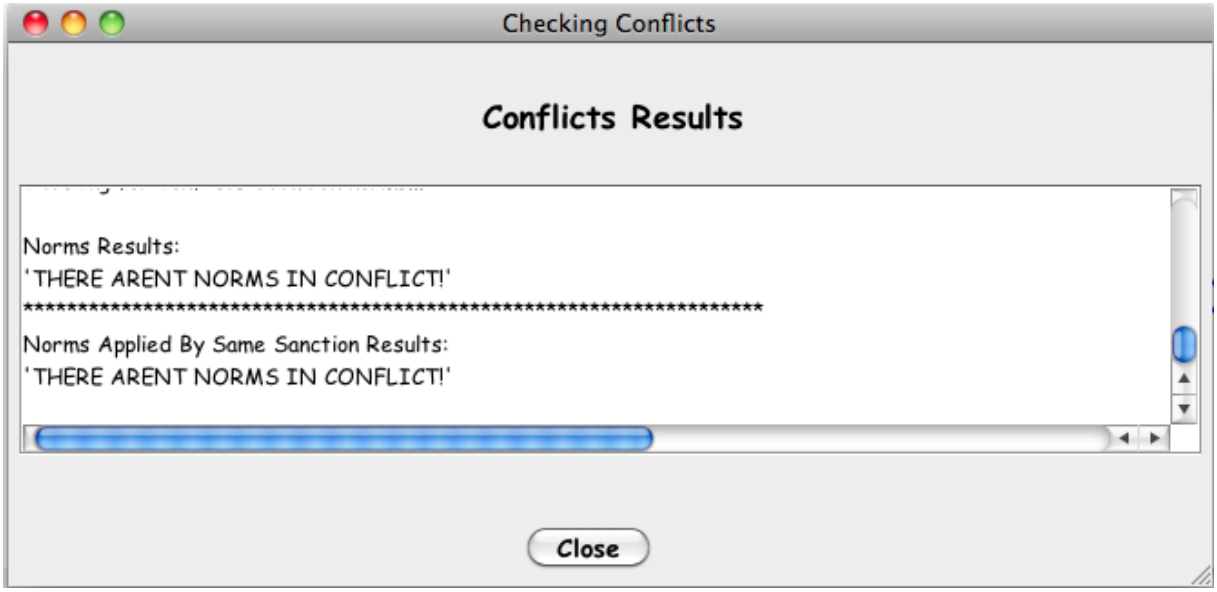

Figura 4.27 Feedback ao usuário - conflito

4.5 DETALHES DA IMPLEMENTAÇÃO

4.5.1 Casos de Uso da ferramenta

Antes de detalhar os casos de uso da ferramenta, inicialmente iremos listar os requisitos **funcionais** da mesma. Os requisitos não funcionais não serão listamos pois não foram implementados..

I. Lista de requisitos funcionais

RF1: Possibilidade de abrir um arquivo XMI (que já foi criado na ferramenta em outro instante).
RF2: Possibilidade de salvar, em qualquer momento, o XMI que está sendo gerado na ferramenta.
RF3: Na criação da norma, primeiramente, é preciso informar um nome e um tipo. Posteriormente, é necessário relacionar os contextos, as entidades, os recursos e as restrições com a norma que está sendo gerada.
RF4: Na criação da norma é possível excluir qualquer associação criada antes de finalizar a geração da norma.
RF5: Possibilitar à visualização a norma e seus relacionamentos, bem como as sanções desta norma, caso haja.
RF6: Durante a visualização da norma é possível excluir qualquer relação anteriormente criada, como também suas sanções.
RF7: Possibilidade de associar sanções para as normas criadas. Na definição das sanções utilizamos as normas já existentes e definimos estas como recompensas ou punições.
RF8: Deve ser possível verificar a existência de conflitos entre as normas criadas. Sendo analisado desta forma também, se o modelo está ou não bem formado.

RF9: Possibilidade de criar outros elementos (agentes, *roles*, organizações, ambientes e entidade), sem a necessidade de associação a uma determinada norma.

RF10: É possível visualizar o manual da ferramenta a qualquer instante.

RF11: É possível sair da ferramenta em qualquer instante.

II. Descrição dos casos de uso

UC1 – Abrir um arquivo XMI

Cenário Típico

1. O usuário acessa a ferramenta.
2. A tela inicial é exibida com as seguintes opções: File, Norm, Check for Conflicts, Elements, Help e Exit.
3. O usuário seleciona File e em seguida escolhe a opção Open XMI File.
4. Uma tela, com a opção de inserir o nome do arquivo, é exibida para que o arquivo desejado seja aberto.
5. O usuário preenche na tela o nome desejado.
6. Em seguida, o usuário tem duas opções: selecionar OK (para abrir o arquivo) ou Exit (para sair da tela).
7. Caso o usuário selecione a opção Exit, a tela será fechada.
8. Caso o usuário selecione a opção OK, a ferramenta verifica o nome do arquivo e carrega o XMI.

Cenários Alternativos

5.a: O usuário preenche um nome inválido.

1. Uma tela é exibida com a seguinte mensagem: “Arquivo não encontrado. Por favor digite um nome válido”.
2. A ferramenta volta para o passo 4.

UC2 – Salvar um arquivo XMI

Cenário Típico

1. O usuário acessa a ferramenta ou está realizando qualquer operação em determinado instante.

2. A tela inicial é exibida com as seguintes opções: File, Norm, Check for Conflicts, Elements, Help e Exit.
3. O usuário seleciona File e em seguida escolhe a opção Save.
   O arquivo será salvo no caminho: C:\XMIFile_Data.

UC3 – Criar uma Norma

Cenário Típico

1. O usuário acessa a ferramenta.
2. A tela inicial é exibida com as seguintes opções: *File*, *Norm*, *Check for Conflicts*, *Elements*, *Help* e *Exit*.
3. O usuário seleciona *Norm* e em seguida escolhe a opção *Create*.
4. Uma tela com detalhes para criação da norma é exibida com o campo *Name* e um combo para escolher o tipo.
5. O usuário deve digitar um nome e escolher um tipo, dentre os seguintes: Obrigação, Permissão ou Proibição.
6. Após detalhar o nome e o tipo da norma, o usuário deve clicar em *Create* para criar as associações.
7. Depois de selecionar a opção Create, o usuário tem seis opções: associar um contexto à norma (caso de uso UC4), associar uma entidade à norma (caso de uso UC5), associar um recurso à norma (caso de uso UC6), associar uma restrição à norma (caso de uso UC7), determinar sanções para a norma (UC11), excluir alguma associação já criada (caso de uso UC8), finalizar a criação da norma (caso de uso UC9) ou sair.
8. O usuário clicou em *Exit*.
9. A tela é fechada.

Cenários Alternativos

5.a: O usuário digitou um nome já existente.

1. Uma mensagem de erro é exibida para o usuário, informando que este nome já existe.
2. A tela é finalizada e a ferramenta volta para o passo 5.

UC4 – Associar um contexto à norma

Cenário Típico

1. Na tela de criação da norma o usuário seleciona a opção Contexts.
2. Uma nova tela é exibida com os combos: Type e Name. E as opções: *Create New*, *Back* e *Apply*.
3. O usuário seleciona alguma opção nos combos *Type* (*Environment* ou *Organization*) e Name. Desta forma, o usuário está utilizando um contexto já existente.
4. A ferramenta carrega nos combos as informações dos contextos já criados.
5. O usuário seleciona a opção *Apply* para finalizar a associação.
6. Na tela de criação da norma, o contexto associado será exibido.

Cenários Alternativos

4.a: O usuário deseja associar um novo contexto.

1. O usuário seleciona a opção *Create New*.
2. Uma nova tela é exibida, para a criação do novo contexto. Conforme casos de uso UC19 ou UC20.
3. O usuário clica em *Apply* e realiza a criação e associação do novo contexto.
4. A ferramenta volta para o passo 6.
5. O usuário seleciona a opção *Back*.
6. A tela é fechada e a ferramenta volta para o passo 4.

5.a: O usuário desistiu de associar um contexto à norma.

1. O usuário seleciona a opção *Back*.
2. A tela é fechada.

UC5 – Associar uma entidade à norma

Cenário Típico

1. Na tela de criação da norma o usuário seleciona a opção Entities.
2. Uma nova tela é exibida com os combos: *Type* e *Name*. E as opções: *Create New, Back* e *Apply*.
3. O usuário seleciona alguma opção nos combos *Type* (*Agent, Role, Organization, Environment, Agent Playing Role* ou *Sub Organization Playing Role*) e *Name*. Desta forma, o usuário está utilizando uma entidade já existente.

4. A ferramenta carrega as informações de contextos já criados nos combos.
5. O usuário seleciona a opção *Apply* para finalizar a associação.
6. Na tela de criação da norma, a entidade associada será exibida.

Cenários Alternativos

4.a: O usuário deseja associar uma nova entidade.

1. O usuário seleciona a opção *Create New*.
2. Uma nova tela é exibida para a criação da nova entidade como descrito nos caso de uso UC12, UC17, UC18, UC19, UC20 ou UC22.
3. O usuário clica em *Apply* e realiza a criação e associação da nova entidade.
4. A ferramenta volta para o passo 6.
5. O usuário seleciona a opção *Back*.
6. A tela é fechada e a ferramenta volta para o passo 4.

5.a: O usuário desistiu de associar uma entidade à norma.

1. O usuário seleciona a opção *Back*.
2. A tela é fechada.

UC6 – Associar um recurso à norma

Cenário Típico

1. Na tela de criação da norma o usuário seleciona a opção *Resources*.
2. Uma nova tela é exibida com os combos: *Resource* e *Action*. E as opções: *Back* e *Apply*.
3. A ferramenta carrega as informações dos recursos nos combos.
4. O usuário seleciona alguma opção nos combos *Resource* (*Agent, Role, Organization, environment, Attribute, Entity, Agent Action, Association End, Method, Message, Protocol, Belief, Goal* ou *Plan*) e *Name* (de acordo com o recurso escolhido).
5. O usuário seleciona a opção *Apply* para finalizar a associação.
6. Na tela de criação da norma, o recurso associado será exibido.

Cenários Alternativos

4.a: O usuário escolheu a opção *Attribute, Belief, Goal, Method* ou *Plan*.

1. Um combo é exibido para que o usuário selecione de quem é aquela entidade.
2. A ferramenta volta para o passo 5.

5.a: O usuário desistiu de associar uma entidade à norma.

1. O usuário seleciona a opção *Back*.
2. A tela é fechada.

UC7 – Associar uma restrição à norma

Cenário Típico

1. Na tela de criação da norma o usuário seleciona a opção *Constraints*.
2. Uma nova tela é exibida com os combos: *Type* e *Condition*. E as opções: *Back*, *Apply* e *Choose*.
3. A ferramenta carrega as informações das restrições nos combos.
4. O usuário seleciona alguma opção nos combos Type (*If, After, Before* ou *Between*) e *Condition* (de acordo com a restrição escolhida).
5. O usuário clica em *Choose*.
6. O usuário seleciona a opção *Apply* para finalizar a associação.
7. Na tela de criação da norma, a restrição associada será exibida.

Cenários Alternativos

5.a: Caso o tipo seja *Between, After* ou *Before* e a condição *Action*.

1. Uma nova tela é exibida para associar a restrição, à uma ação de um determinado recurso.
2. A ferramenta volta para o passo 6.

5.b: Caso o tipo seja *After* ou *Before* e a condição *Date*.

1. Um campo *Date* é exibido para determinar a data que irá restringir.
2. A ferramenta volta para o passo 6.

5.c: Caso o tipo seja *Between* e a condição *Date*.

1. Dois campos *Date* são exibidos para determinar as datas que irão restringir.
2. A ferramenta volta para o passo 6.

5.d: Caso o tipo seja *If* e a condição *Date*.

1. Um campo *Date* é exibido para determinar a data que irá restringir.
2. A ferramenta volta para o passo 6.

5.e: Caso o tipo seja *If* e a condição *Attribute*.

1. Uma nova janela é exibida para a associação de um atributo que vai restringir a norma.
2. O usuário deve informar de que entidade é aquele atributo.
3. A ferramenta volta para o passo 6.

5.f: Caso o tipo seja *If* e a condição *Belief*.

1. Uma nova janela é exibida para a associação de uma crença que vai restringir a norma.
2. O usuário deve informar de que entidade é aquele atributo.
3. A ferramenta volta para o passo 6.

6.a: O usuário desistiu de associar uma entidade à norma.

1. O usuário seleciona a opção *Back*.
2. A tela é fechada.

UC8 – Excluir uma associação da norma

Cenário Típico

1. Na tela de criação da norma o usuário seleciona a associação que deseja excluir e seleciona a opção *Delete*.
2. A ferramenta exclui esta associação da base de dados (listas).
3. A ferramenta carrega na tela de criação da norma todas as informações das associações da norma.

UC9 – Finalizar a criação da norma

Cenário Típico

1. Depois de realizar todas as associações mínimas necessárias (contexto, entidade e recurso), o usuário seleciona a opção *Complete*.
2. A ferramenta cria efetivamente a norma na base de dados (listas).
3. A tela para criação da norma é fechada.

UC10 – Visualizar a norma e suas associações

Cenário Típico

1. O usuário acessa a ferramenta.

2. A tela inicial é exibida com as seguintes opções: *File, Norm, Check for Conflicts, Elements, Help* e *Exit*.
3. O usuário seleciona *Norm* e em seguida escolhe a opção *View*.
4. Uma nova tela será exibida com os combos: *Norm, Contexts, Entities, Resources, Constraints* e *Sanctions*.
5. A ferramenta carrega as normas existentes no combo *Norm*.
6. O usuário seleciona a norma que ele deseja visualizar.
7. O usuário seleciona a opção *View*.
8. A ferramenta carrega nos demais combos as associações (contexto, entidade, recurso, restrição e sanção) da norma.
9. O usuário pode adicionar ou deletar associações desta norma.
10. O usuário seleciona a opção *Apply*.
11. A tela para visualização da norma é fechada.

Cenários Alternativos

5.a: Caso nenhuma norma tenha sido criada.

1. A opção *View* da tela fica desabilitada.
2. O usuário seleciona a opção *Exit*.
3. A tela é fechada.

9.a: O usuário deseja excluir alguma associação.

1. Na tela de visualização da norma o usuário seleciona a associação que deseja excluir e seleciona a opção *Delete*.
2. A ferramenta exclui esta associação da base de dados (listas).
3. A ferramenta carrega na tela de visualização da norma todas as informações das associações da norma.

9.b: O usuário deseja adicionar alguma associação.

1. Na tela de visualização da norma o usuário seleciona qual elemento deseja associar e seleciona o botão *Add*.
2. A ferramenta inclui esta associação da base de dados (listas).
3. A ferramenta carrega na tela de visualização da norma todas as informações das associações da norma.

10.a: O usuário seleciona a opção *Exit*.

1. A tela para visualização da norma é fechada e nenhuma informação é salva.

UC11 – Definir sanções para uma norma

Cenário Típico

1. O usuário acessa a ferramenta.
2. O usuário está criando uma norma e seleciona a opção Sanction.
3. Uma nova tela será exibida com os combos: *Norm* e *Sanction*. E as opções: *Punishment, Reward, Back* e *Apply*.
4. A ferramenta carrega as normas existentes no combo *Norm*.
5. O usuário define se a sanção será uma punição ou uma recompensa.
6. A ferramenta carrega todas as normas, exceto a escolhida no combo *Norm*, no combo *sanction*.
7. O usuário clica em *Apply*, finalizando a criação de uma sanção pra uma determinada norma e salvando esta informação na base de dados (listas).
8. Esta informação é mostrada na tela de criação da norma.
9. A tela é fechada.

Cenários Alternativos

6.a: O usuário deseja criar uma nova norma.

1. O usuário seleciona a opção *Create*.
2. A tela de criação de norma é exibida, conforme detalhado no caso de uso UC3.
3. O usuário finaliza a criação da norma.
4. A tela para relacionar sanção com norma é exibida contendo as informações anteriores, e carregando no combo *Sanction* a nova norma criada.
5. A ferramenta volta para o passo 8.

8.a: O usuário deseja criar uma nova norma.

1. O usuário seleciona a opção *Back*.
2. A tela é fechada.

UC12 – Criar um agente exercendo um papel

Cenário Típico

1. O usuário está criando as entidades da norma.
2. O usuário escolhe a opção *Agente Playing a Role*.

3. Uma tela é exibida com a opção de escolher um agente e um papel.
4. O usuário seleciona *Apply*.
5. A tela é fechada e a entidade é mostrada no combo.

Cenários Alternativos

4.a: O usuário desistiu.

1. O usuário clica em *Back*.
2. A tela de selecionar entidade para a norma é exibida.

UC13 – Checar conflito

Cenário Típico

1. O usuário acessa a ferramenta.
2. A tela inicial é exibida com as seguintes opções: *File, Norm, Check for Conflicts, Elements, Help* e *Exit*.
3. O usuário seleciona a opção *Check for Conflicts*.
4. A ferramenta verifica se existe conflito entre as normas criadas.
5. Uma tela é exibida contendo uma área de texto onde os resultados serão apresentados (Se existe, ou não conflito e se as regras de boa formação estão formadas da maneira correta).
6. O usuário seleciona a opção *Close*.
7. A janela fecha.

UC14 – Sair da ferramenta

Cenário Típico

1. O usuário acessa a ferramenta.
2. A tela inicial é exibida com as seguintes opções: *File, Norm, Check for Conflicts, Elements, Help* e *Exit*.
3. O usuário seleciona a opção *Exit*.
4. A ferramenta salva o XMI gerado pela ferramenta.
5. A ferramenta é finalizada.

UC15 – Acessar o manual da ferramenta

Cenário Típico

6. O usuário acessa a ferramenta.

7. A tela inicial é exibida com as seguintes opções: *File, Norm, Check for Conflicts, Elements, Help* e *Exit*.
8. O usuário seleciona *Help* e em seguida escolhe a opção Manual.
9. A ferramenta busca o documento (PDF) no local indicado e exibe o mesmo para o usuário.

UC16 – Criar outros elementos

Cenário Típico

1. O usuário acessa a ferramenta.
2. A tela inicial é exibida com as seguintes opções: *File, Norm, Check for Conflicts, Elements, Help* e *Exit*.
3. O usuário seleciona Elements e em seguida tem a opção de gerar cinco tipos de elementos: *Agent, Role, Organization, Environment, Association End* ou *Entity*.

UC17 – Criar um agente

Cenário Típico

1. O usuário escolhe a opção *Agent*.
2. Uma nova tela é exibida com as opções de editar um agente existente ou gerar um novo agente. Para gerar as associações deste agente temos os seguintes campos: Nome, *Environmente, Belief, Goal, Plan, Agent Action, Role* e *Organization*.
3. O usuário preenche os campos da maneira desejada.
4. O usuário seleciona a opção *Apply*.
5. A ferramenta armazena todas as associações geradas na base de dados (listas).
6. A página é fechada.

Cenários Alternativos

5.a: O usuário desistiu de criar um agente.

1. O usuário seleciona a opção *Back*.
2. A ferramenta vai para o passo 6.

UC18 – Criar uma *role*

Cenário Típico

1. O usuário escolhe a opção *Role*.
2. Uma nova tela é exibida com as opções de editar uma *role* existente ou gerar uma nova *role*, e definir se é uma *subrole* ou não. Para gerar as associações desta role temos os seguintes campos: Nome, *Goal, Protocol, SuperRole,* e *Organization.*
3. O usuário preenche os campos da maneira desejada.
4. O usuário seleciona a opção *Apply*.
5. A ferramenta armazena todas as associações geradas na base de dados (listas).
6. A página é fechada.

Cenários Alternativos

5.a: O usuário desistiu de criar uma role.

1. O usuário seleciona a opção *Back*.
2. A ferramenta vai para o passo 6.

UC19 – Criar um ambiente

Cenário Típico

1. O usuário escolhe a opção *Environment*.
2. Uma nova tela é exibida com as opções de editar um ambiente existente ou gerar um novo ambiente. Para gerar as associações deste ambiente temos o seguinte campo: Nome.
3. O usuário preenche os campos da maneira desejada.
4. O usuário seleciona a opção *Apply*.
5. A ferramenta armazena todas as associações geradas na base de dados (listas).
6. A página é fechada.

Cenários Alternativos

5.a: O usuário desistiu de criar um ambiente.

1. O usuário seleciona a opção *Back*.
2. A ferramenta vai para o passo 6.

UC20 – Criar uma organização

Cenário Típico

1. O usuário escolhe a opção *Organization*.
2. Uma nova tela é exibida com as opções de editar uma organização existente ou gerar uma nova organização. Para gerar as associações desta organização temos os seguintes campos: Nome, *SuperOrganization e Environment..*
3. O usuário preenche os campos da maneira desejada.
4. O usuário seleciona a opção *Apply*.
5. A ferramenta armazena todas as associações geradas na base de dados (listas).
6. A página é fechada.

Cenários Alternativos

5.a: O usuário desistiu de criar uma organização.

1. O usuário seleciona a opção *Back*.
2. A ferramenta vai para o passo 6.

UC21 – Criar uma entidade

Cenário Típico

1. O usuário escolhe a opção *Entiity*.
2. Uma nova tela é exibida com as opções de editar uma entidade existente ou gerar uma nova entidade. Para gerar as associações desta entidade temos os seguintes campos: Nome, *Attributes e Methods.*
3. O usuário preenche os campos da maneira desejada.
4. O usuário seleciona a opção *Apply*.
5. A ferramenta armazena todas as associações geradas na base de dados (listas).
6. A página é fechada.

Cenários Alternativos

5.a: O usuário desistiu de criar uma entidade.

1. O usuário seleciona a opção *Back*.
2. A ferramenta vai para o passo 6.

UC22 – Criar um suborganização exercendo um papel

Cenário Típico

1. O usuário está criando as entidades da norma.
2. O usuário escolhe a opção *SubOrganization Playing a Role*.
3. Uma tela é exibida com a opção de escolher uma suborganização e um papel.
4. O usuário seleciona *Apply*.
5. A tela é fechada e a entidade é mostrada no combo.

Cenários Alternativos

4.a: O usuário desistiu.

1. O usuário clica em *Back*.
2. A tela de selecionar entidade para a norma é exibida.

UC23 – Criar uma *Association End*

Cenário Típico

1. O usuário escolhe a opção *Association End*.
2. Uma nova tela é exibida com as opções de criar uma *Association End*. Para gerar as associações desta entidade temos os seguintes campos: As entidades que devem ter a ponta de associação.
3. O usuário preenche os campos da maneira desejada.
4. O usuário seleciona a opção *Apply*.
5. A ferramenta armazena todas as associações geradas na base de dados (listas).
6. A página é fechada.

Cenários Alternativos

5.a: O usuário desistiu de criar uma entidade.

1. O usuário seleciona a opção *Back*.
2. A ferramenta vai para o passo 2.

III. Diagrama de Casos de Uso

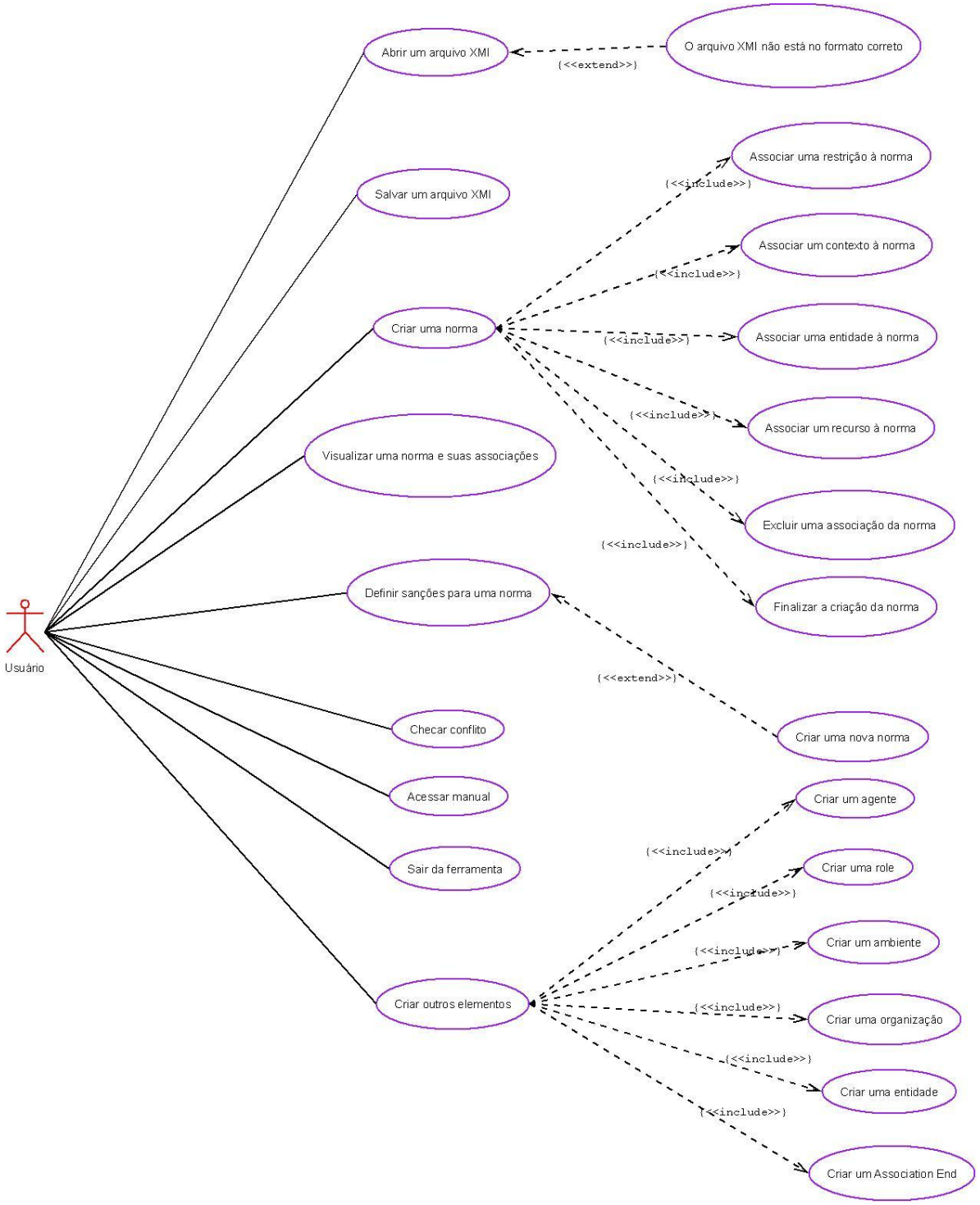

Figura 4.28 Diagrama de Casos de Uso

4.5.2 Diagrama de Classes

Para representar os tipos (classes) de objetos da ferramenta construímos um diagrama de classes, representado pela figura 4.3. E neste detalhamos tanto os atributos como também os métodos utilizados em cada classe.

Definimos Ferramenta como a principal classe do sistema. Esta utiliza três pacotes principais: Interface Pck, composto pelas classes que formam a interface (acesso direto do usuário); Leitura XMI, formado pelas classes necessárias para realizar a leitura do XMI; e Escrita XMI, que contém as classes para efetuar a escrita do XMI.

Temos neste diagrama a possibilidade de relacionar com a Ferramenta nenhum ou um Usuário ao longo do tempo. Podemos também associar à Ferramenta um Meta-modelo, um Diagrama de Objetos, um Regra de Boa Formação, um Regra de Verificação de Conflito.

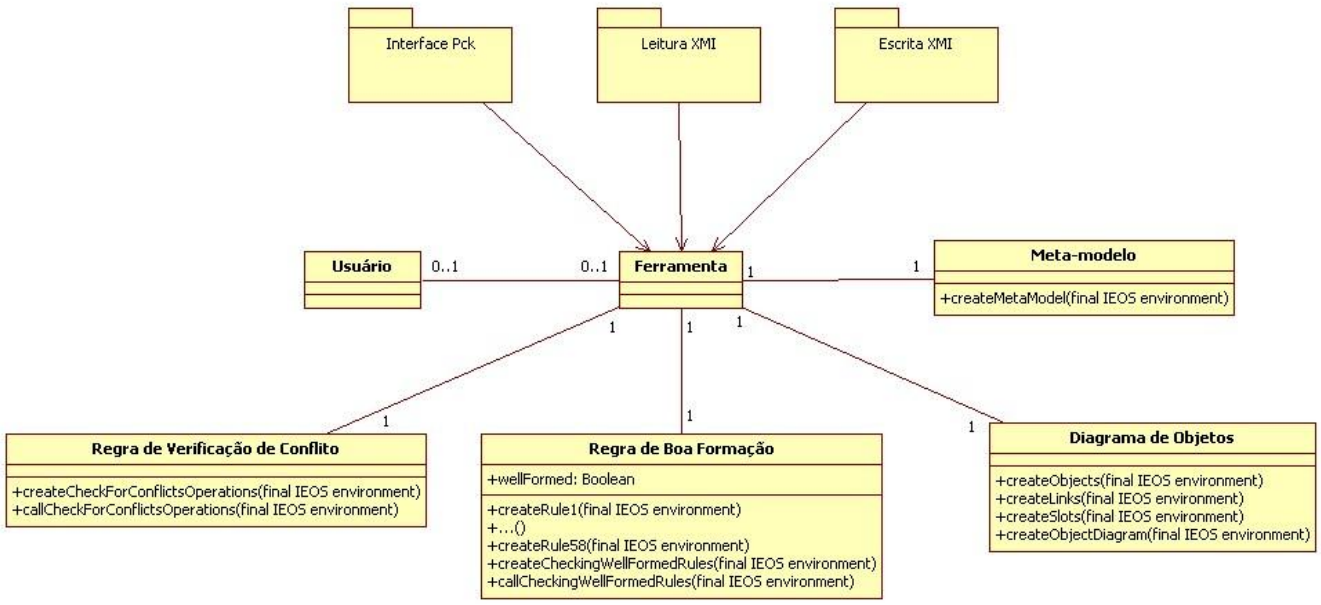

Figura 4.29 Diagrama de Classes

CAPÍTULO 5: TRABALHOS RELACIONADOS

Nas pesquisas realizadas durante este trabalho não encontramos nenhuma ferramenta que possui como objetivo principal a modelagem de normas em sistemas multiagentes e a verificação de conflitos entre estas.

Porém, a fim de compararmos certos aspectos da nossa abordagem, podemos considerar a ferramenta OperettA [Okouya, D. and Dignum,V. (2008)]. Esta é uma ferramenta gráfica que suporta o *design*, a verificação e a simulação de modelos OperA [Okouya, D. and Dignum,V. (2008)]. Possui como um ponto importante dar suporte ao desenvolver e documentar os vários aspectos da especificação e do *design* de uma organização multiagente. Porém, diferentemente da ferramenta apresentada neste trabalho, possui foco principal na modelagem de organizações, determinadas pelo seus *stakeholders* em termos dos seus objetivos, normas, interações e ontologias.

CAPÍTULO 6: CONCLUSÃO E TRABALHOS FUTUROS

Com esse trabalho pretendeu-se abordar algumas disciplinas de Ciência da Computação com a principal intenção de colocar em prática o que foi estudado ao longo do curso de graduação.

Embora o foco do projeto tenha sido a área de agentes e normas para sistemas multiagentes, outros aspectos importantes foram a fundamentação sobre o meta-modelo utilizado para implementação da ferramenta, a linguagem OCL, a API EOS utilizada e o estudo de caso que foi proposto.

Com a base fornecida pelas disciplinas e o estudo de artigos específicos da área de agentes e normas, bem como das áreas listadas no parágrafo anterior, foi possível exercitar o senso crítico e propor novas idéias, o que originou a Ferramenta para Modelagem de Normas.

A modelagem de normas é um aspecto de grande importância para um SMA no qual agentes são entidades autônomas e com objetivos diversos e neste estudo podemos modelar algumas normas dentro do escopo escolhido fazendo assim o uso dos conceitos definidos na NormML. Além disso, foram implementadas operações em OCL com objetivo de concretizar a modelagem de normas e de levantar questões que foram sistematizadas, colocando assim em prática os conhecimentos adquiridos. Essa linguagem foi fundamental para que pudéssemos descrever as regras de boa formação e de verificação de conflitos, criadas de acordo com o meta-modelo existente.

Como resultado deste projeto, tem-se uma primeira versão da Ferramenta para Modelagem de Normas. Essa versão consiste na interface com algumas funcionalidades implementadas, tais como a criação de um modelo completo de modelagem de normas, bem como a possibilidade de editar o modelo, adicionando ou excluindo associações, e por fim, verificar se existe, ou não, conflito entre as normas contidas no modelo.

Como um item interessante para trabalhos futuros, é possível efetuar melhorias nesta ferramenta, onde as restrições entre as classes e associações existentes no meta-modelo estivessem presentes na implementação e

modelagem.Outra expansão que pode ser realizada é fazer desta ferramenta uma ferramenta gráfica, ou seja, que o usuário possa modelar normas de forma gráfica e por fim verificar a existência de conflitos, através de uma modelagem gráfica, visualmente mais simples e clara.

REFERÊNCIAS BIBLIOGRÁFICAS

Agent - Object Management Group (2000), http://www.objs.com/agent, “Agent Technology”, agent/00-09-01.

Basin, D., Clavel, M., Doser, J. and Egea, M. (2009), “Automated analysis of security-design models”, Inf. Software Technology, 51(5), pp: 815-831.

Clavel, M., Egea, M. and Dios, M., (2008) “Building an Efficient Component for OCL Evaluation”, Proceedings of the 8th International Workshop on OCL Concepts and Tools (OCL 2008), at MoDELS 2008.

Clavel, M., Egea, M. and Silva, V. T. (2007) “Model Metrication in MOVA: A Metamodel-Based Approach using OCL”, Disponível em: <http://maude.sip.ucm.es/~marina/pubs/pubs.html>. Acesso em 20 de Setembro de 2010.

Danc, J. (2008). “Formal Specification of AML”, Formal Specification of AML, ser. Master’s Thesis.

Dictionary, Disponível em: <http://dictionary.reference.com/>. Acesso em 5 de Outubro de 2010.

Figueiredo, K. S., Silva, V. (2010) “NormML: A Modeling Language to Model Norms”. AutoSoft - Autonomous Software Systems (Workshop) at Brazilian Conference on Software: Theory and Practice (CBSoft 2010), Salvador, ISSN 2178-6097, p.11-20.

Jennings, N. R. (1999) “On agent-based software engineering”, In Artificial Intelligence, 117 (2) pp. 277-296.

Martins, E. (2003) “OCL: Object Constraint Language”, Disponível em: <http://www.omg.org/spec/OCL/>. Acesso em 5 de Outubro de 2010.

Meta-modelo Concreto. Disponível em: <http://www.ic.uff.br/~kfigueiredo/normML/concretemetamodel.pdf>. Acesso em 14 de Novembro de 2010.

Meta-modelo NormML. Disponível em: <http://www.ic.uff.br/~kfigueiredo/normML/metamodel.pdf>. Acesso em 14 de Novembro de 2010.

Object Management Group - OMG. Disponível em: <http://www.omg.org>. Acesso em 10 de Outubro de 2010.

OCL - Object Management Group - OMG. Disponível em: <http://www.omg.org/spec/OCL/>. Acesso em 10 de Outubro de 2010.

Odell, J., Parunak, H., and Bauer, B. (2000). “Extending UML for Agents”, In Proc. Agent-Oriented Information Systems Workshop at National Conf. of AI, pp. 3-17.

Okouya, D. and Dignum,V. (2008) “OperettA: A prototype tool for the design, analysis and development of multi-agent organizations”, In Proc. AAMAS 2008. IFAAMAS.

OMG – Object Management Group (2003) “UML 2.0 OCL Specification”, ptc/03-10-14.

Regras de Boa Formação. Disponível em: <http://www.ic.uff.br/~kfigueiredo/normML/rules.pdf >. Acesso em 14 de Novembro de 2010.

Regras de Transformação - Transformador. Disponível em: <http://www.ic.uff.br/~kfigueiredo/normML/concrete2abstract.pdf>. Acesso em 14 de Novembro de 2010.

Regras de Verificação de Conflitos. Disponível em: <http://www.ic.uff.br/~kfigueiredo/normML/conflicts.pdf >. Acesso em 14 de Novembro de 2010.

Silva, V. T., Braga, C. and Figueiredo, K. (2010) “A modeling language to model norms”, Proc. of COIN@AAMAS, Toronto, Canada, pp. 25-32.

Silva, V. T., Choren R. and Lucena, C. (2008), “MAS-ML: A Multi-Agent System Modeling Language”, In Int. Journal of Agent-Oriented Soft. Eng., Special Issue on Modeling Languages for Agent Systems, vol.2, no.4, pp. 382-421.

Silva, V. T. (2008) “From the Specification to the Implementation of Norms: An Automatic Approach to Generate Rules from Norms to Govern the Behaviour of Agents”, In International Journal of Autonomous Agents and Multi-Agent Systems, Special Issue on Norms in Muli-Agent Systems, 17(1) Springer-Verlag, pp. 113-155.

Transformador. Disponível em: <http://www.ic.uff.br/~kfigueiredo/normML/concrete2abstract.zip>. Acesso em 14 de Novembro de 2010.

UML – Object Management Group (2005) “Unified Modeling Language: Superstructure version 2.0”, formal/05-07-04.

UML - Object Management Group - OMG. Disponível em: <www.omg.org/uml/>. Acesso em 10 de Outubro de 2010.

Vazquez-Salceda, J., Aldewereld, H. and Dignum, F. (2005) “Implementing Norms in Multiagent Systems”, In LNAI 3187, pp. 313-327.

Wooldridhe, M. (1997) “Agent-based software engineering”, IEE Proc. Software Engineering 144 (1) 26–37.

Wooldridge, M. and Ciancarini, P. (2001) “Agent-Oriented Software Engineering: The State of the Art”, In Agent-Oriented Software Engineering, First International Workshop, AOSE 2000, Ciancarini, P., Wooldridge, M., Eds., LNCS 1957 Springer, Limerick, Ireland, pp. 1-28.