Instituto Superior de Ciências do Trabalho e da Empresa
Departamento de Ciências e Tecnologias da Informação

Representação de Restrições Deônticas em OCL

Miguel Nuno Boavida das Neves Ramos

Tese submetida como requisito parcial para obtenção do grau de

Mestre em Ciências e Tecnologias da Informação
Especialidade em Gestão de Sistemas de Informação

Orientador:

Professor Doutor Pedro Nogueira Ramos

Setembro de 2008

Instituto Superior de Ciências do Trabalho e da Empresa
Departamento de Ciências e Tecnologias da Informação

Representação de Restrições Deônticas em OCL

Miguel Nuno Boavida das Neves Ramos

Tese submetida como requisito parcial para obtenção do grau de

Mestre em Ciências e Tecnologias da Informação
Especialidade em Gestão de Sistemas de Informação

Orientador:

Professor Doutor Pedro Nogueira Ramos

Setembro de 2008

Agradecimentos

Ao Professor Doutor Pedro Nogueira Ramos, orientador científico desta investigação, pelo apoio sistemático e orientação que sempre disponibilizou e pela compreensão e encorajamento, imprescindíveis para levar a bom termo este trabalho.

À minha mulher Sara pelo apoio e paciência constantes, e por acreditar, mais do que eu, que este trabalho chegaria ao fim.

Ao meu filho André, pelas brincadeiras adiadas.

Aos meus pais e ao meu irmão pelo apoio e estímulo constantes.

Aos meus colegas de trabalho, Carlos e Vera, pelo apoio e compreensão demonstrados.

Agradeço a todas as pessoas que contribuíram para a concretização desta dissertação, estimulando-me intelectual e emocionalmente.

Resumo

Por vezes com a utilização, real, dos sistemas de informação somos confrontados com situações cujo carácter excepcional não permite, ao modelador de sistemas de informação, representar todas as propriedades que revestem as relações entre as entidades do mundo real. Isto acontece porque as associações entre classes de objectos, que representam as entidades do mundo real, dependem de restrições rígidas, verificadas por um valor “sim” ou “não” das quais depende a formação dos objectos da associação. Nestes casos não basta retirar a restrição imposta e permitir a associação, porque a representação da realidade sairia empobrecida, em vez disso torna-se necessário o enriquecimento gráfico e semântico do diagrama de classes da UML, possibilitando a caracterização de um novo tipo de restrição: a restrição deôntica.

É proposta uma metodologia de levantamento e descrição de casos reais em que a utilização de sistemas de informação justifique a utilização de restrições deônticas. Através dos mecanismos de extensão da uml e do recurso à linguagem de modelação ocl pretende-se definir uma representação genérica, a aplicar a estes casos.

Em qualquer sistema de informação é fundamental preservar a persistência da informação, assim o modelo relacional será utilizado para o efeito. Como o diagrama de classes implementa o modelo orientado ao objecto é necessário proceder ao mapeamento do diagrama de classes para o modelo relacional, para o efeito é proposta uma metodologia que assenta na aplicação nas regras generalistas de mapeamento entre os dois modelos; seguidas da criação de tabelas adicionais para mapear e preservar a informação relativa às restrições deônticas; e da criação de uma estrutura de triggers que implementam a semântica, ou seja o comportamento esperado das restrições deônticas.

Pretende-se após representação, no diagrama de classes, do sistema, e das restrições deônticas, que seja criado automaticamente o diagrama relacional que lhes corresponde. Para o efeito é proposto o desenvolvimento de uma aplicação, em Prolog, que lê os dados do diagrama de classes, processa os mesmos, e finalmente, cria automaticamente, a estrutura relacional correspondente.

Palavras-chave: UML, diagrama de classes, modelo relacional, restrições deônticas.

Abstract

Sometimes we are faced with exceptional situations that cannot be represented, by the modeler, as they exist in the real world. This happens because the associations between object classes, that represent the real world entities, depend on strong constraints, verified by a value "yes" or "no", on which depends the creation of the association objects. In these cases it is not enough to remove the restriction imposed and allow the association, because this would impoverished reality demonstration, instead it is necessary to enrich the graphical and semantic representation of the UML class diagram, allowing the characterization of a new type of restriction called deontic, or soft, constraint.

A methodology for survey and description of actual cases in which the use of information systems justify the application use of soft constraints, is proposed. Through the extension mechanisms of UML and the use of OCL modeling language, it is intended to define a generic representation, to apply to such cases.

In any information system it is essential to preserve the persistence of information, so the relational model will be used for this purpose. As the diagram of classes implements the object oriented model it is necessary to map from this model to the relational one. To achieve this it is proposed a methodology based on the application of general mapping rules between the two models; followed by the creation of additional tables, to map and preserve the information of soft constraints; and the creation of trigger structures that implement the semantics, or in other words the expected behavior, of those kinds of constraints.

After representation in the class diagram of the system and soft constraints, it is intended to be automatically created the relational diagram that represents them. The aim is to develop an application in Prolog, which reads data from the diagram of classes, processes them, and finally, automatically generates the correspondent relational structure.

Keywords: UML, class diagram, relational model, soft constraints.

Indíce

Indíce de figuras

Indíce de tabelas

Capítulo 1 – Introdução

1.1 Motivação e enquadramento.

1.2 Apresentação do problema.

1.3 Metodologia.

1.4 Objectivos.

1.5 Limitações do estudo.

1.6 Estrutura da dissertação

1.7 Contributos para o desenvolvimento da dissertação

1.1 Motivação e enquadramento

Tentar representar a realidade é um trabalho, para sempre, inacabado. Contudo, cada vez mais necessário.

Capturar a realidade, mesmo sobre determinada perspectiva, e descrevê-la de forma que outros a apreendam é um trabalho àrduo e rigoroso. A realidade não é estática, transforma-se e adquire novas perspectivas, e por isso a tarefa de descrevê-la é contínua e inacabada. Por exemplo, a sobrevivência das organizações no mundo empresarial depende da capacidade que os sistemas de informação detêm, na representação fidedigna dos seus processos de negócio. Quanto mais e quanto melhor a realidade de uma organização for representada, maiores as vantagens competitivas adquiridas, relativamente aos seus concorrentes directos.

A realidade é, por vezes, tão vasta, que não conseguimos compreendê-la de uma só vez. Nesses casos é recomendável perceber um pouco de cada vez, possibilitando que a estrutura e a envolvente sejam plenamente apercebidas e compreendidas.

Booch, Rumbaugh e Jacobson (1998), referem que o modelo é a simplificação da realidade. Segundo os mesmos autores, a modelação permite descrever sistemas sobre diferentes perspectivas, recorrendo a variados modelos que podem ser detalhados, ou generalistas; estruturais, ou comportamentais. Considerando, por exemplo, a concepção de um automóvel, no que respeita ao capítulo motor é necessário garantir o encaixe perfeito dos cilindros, a combustão está sujeita a uma série de parâmetros que é necessário respeitar, a arquitectura tem que se ajustar perfeitamente à potência pretendida, etc., neste caso é plenamente justificada a utilização de modelos estruturais, detalhados, em que o rigor deve ser a preocupação central para caracterizar o sistema; na concepção da carroçaria estão outros valores em causa, entre eles o design apelativo, normalmente pretendido, que justifica a utilização de modelos generalistas que apelem à abstracção dos elementos não essenciais.

Segundo Edsger Dijkstra, (citado em Booch et al., 1998), a resolução de problemas difíceis está na divisão dos mesmos numa série de pequenos problemas, resolúveis. O Autor refere que através da modelação se pode ampliar o intelecto humano e que um modelo devidamente escolhido pode permitir trabalhos com altos níveis de abstracção.

Por um lado, o desenvolvimento de sistemas de informação depende em parte da divisão dos grandes problemas das organizações em pequenos problemas, até que a solução seja encontrada; por outro, tal como na concepção de um automóvel, são necessários diferentes tipos de modelos para o descrever: modelos generalistas que são utilizados no desenho e descrição dos processos de negócio, modelos estruturais para definir a arquitectura dos objectos e da base de dados e modelos comportamentais que definem o comportamento do sistema de informação e as suas excepções. Assim, contribuir para melhorar a representação da realidade, com recurso à modelação, é a motivação deste estudo.

A UML[1], é uma linguagem de modelação aceite como standard pela OMG[2], que permite modelar os sistemas de software através de uma variedade de modelos, que se integram de forma a estruturarem e documentarem os sistemas de informação de forma sistemática. A sua utilização é preconizada num contexto abrangente de modelação de sistemas de informação, o que justifica a sua escolha no decorrer desta investigação.

O modelo de classes, integrado na linguagem UML, especifica as classes de objectos que se vão utilizar no decorrer do desenvolvimento do sistema de software. (Booch et al., 1998) define uma classe como sendo uma descrição dos atributos, operações, relações, e semântica que um conjunto de objectos partilha.

No contexto de utilização dos sistemas de informação, coloca-se uma importante questão: Se o diagrama de classes cria a estrutura dos objectos que vão ser criados, como se podem guardar os objectos e a informação contida em cada um deles? Por outras palavras como é que se pode garantir a persistência dos objectos? Silva e Videira (2005) referem a existência de 4 hipóteses:

> - Linguagens Persistentes (e.g., Pijama (Java Persistente), Napier, Prothos), em que a própria linguagem de programação dá suporte à definição de objectos persistentes. . . .
> - Mecanismos simples de serialização dos dados dos objectos para/de ficheiros simples. Esses ficheiros podem ser de texto ou binário. . . .
> - SGBD-R: Sistemas de gestão de bases de dados relacionais (e.g., Oracle, MySQL, SQL Server, DB2) são sistemas versáteis que oferecem inúmeras capacidades de armazenamento e gestão de dados, sendo na prática, os sistemas mais populares e com maior sucesso comercial. . . .

[1] UML é o acrónimo para “Unified Modelling Language”, é uma linguagem de modelação de sistemas de Software.

[2] OMG é o acrónimo para “Object Management Group”, organização internacional que aprova padrões abertos para aplicações orientadas a objectos.

> - SGBD-OO: Sistemas de gestão de bases de dados OO (e.g., Ontos, Ardent, ObjectStore, GemStone) são sistemas mais limitados e com menor sucesso comercial comparativamente com os SGBD-R. Em vez de se basearem no modelo relacional suportam directamente o modelo OO. . . . (p. 296).

As linguagens persistentes e os mecanismos simples de serialização não têm as capacidades de armazenamento e gestão de dados, que caracterizam os sistemas de gestão de bases de dados, portanto a sua utilização está restrita a sistemas pessoais, em que a quantidade de dados envolvida não justifica processos de gestão dos dados ou eficácia melhorada no seu armazenamento. As duas hipóteses restantes são os sistemas de gestão de bases de dados, que implementam ou o modelo relacional ou o modelo OO[3]. No caso do modelo relacional, a estrutura dos objectos tem que ser traduzida para tabelas, que posteriormente armazenam os dados; no modelo OO, a representação dos objectos é directa, porque o modelo suporta a sua estrutura. Uma análise inicial determina a utilização do modelo OO, justificado pela sua capacidade imediata de guardar a estrutura dos objectos e os seus dados. No entanto, existem outros factores a considerar, o facto de estes sistemas apresentarem mais limitações, menos sucesso comercial, e menos proliferação comparados com o rival, assente no modelo relacional, são factores determinantes na escolha de uma tecnologia e justificam a escolha do modelo relacional em detrimento do modelo OO. Assim, no decorrer desta investigação será utilizado o sistema de gestão de bases de dados relacional.

Através de um conjunto de regras bem definidas, o modelo de classes pode ser mapeado para o modelo relacional, de forma automática. Às tabelas, do diagrama relacional gerado, correspondem classes com os mesmos atributos. As relações entre classes são mapeadas através da criação de chaves primárias e estrangeiras que garantem respectivamente a unicidade e a relação dos registos.

Entre classes, existem vários tipos de relações possíveis que, por vezes, definem uma semântica de relacionamento assente em restrições obrigatórias. Neste caso o objecto de uma classe só pode existir se a restrição obrigatória relativa ao objecto com que se relaciona, for cumprida. Um exemplo prático desta semântica é a relação entre factura e número. Uma factura sem número, no mundo real, não deve existir, assim, considerando que factura e número são dois objectos distintos, a factura tem que cumprir

[3] OO é o acrónimo para Object Oriented, diz respeito ao paradigma de programação OOP (Object Oriented Programming) que utiliza objectos e interacções entre os mesmos para desenhar software aplicacional.

obrigatoriamente o requisito de estar associada a um número, para garantir que a representação da realidade tenha consistência.

As restrições obrigatórias repercutem-se também no modelo relacional, através da geração de um campo correspondente a uma chave estrangeira, que tem que assumir um dos seus possíveis valores para que a transacção se efectue.

Segundo Carmo, Demolombe e Jones (2001) as restrições fortes (obrigatórias) são verdades analíticas, que em qualquer possível mundo, que seja real, são satisfeitas.

No caso da factura e do número, dois objectos distintos, a restrição obrigatória que é imposta significa que a factura tem que ter sempre um número em qualquer mundo que seja real. No entanto há situações em que isto pode não ser verdade: criar uma factura está dependente de bens ou serviços que foram prestados. No decorrer da sua criação, vão-se adicionando as linhas respeitantes aos diversos itens que a compõem. Supondo que após adicionar um conjunto de itens, o funcionário que está a redigir a factura não sabe quantas horas de serviço deve adicionar ao último item, então, ou o funcionário numera a factura, e a mesma é criada; ou desiste, e a factura não é criada. Se a opção for numerar, o sistema não tem maneira de saber que aquela factura não é a final, podendo originar dados incorrectos, por exemplo se o funcionário esquecer que tem de adicionar mais um item. A opção que resta, desistir do processo de criação da factura, é demasiado penosa, uma vez que todo o trabalho será perdido.

Assim, a situação descrita sugere a utilização de outro tipo de restrição que em (Carmo et al, 2001) é designado por restrição deôntica. Segundo os mesmos autores as restrições deônticas expressam as normas de como as coisas deviam ser, num mundo ideal, no entanto, em situações excepcionais que ocorrem no mundo real, a violação dessas normas é possível.

Conforme explicado em (Carmo & Jones, 2002), a lógica deôntica tem origem na filosofia, mas começou a atrair o interesse dos investigadores noutras áreas do conhecimento, tais como ciências da computação, gestão e organização. Entre as áreas de aplicação estão a segurança e políticas de controlo de acesso de sistemas de computadores, e restrições de integridade de bases de dados. Interessa a este estudo a aplicabilidade da lógica deôntica a restrições de integridade de bases de dados.

Na perspectiva de aplicação de restrições deônticas a uma base de dados, é essencial informar o sistema das violações que ocorrem. Como a lógica deôntica recorre à lógica

matemática para representar a informação e não considera representações diagramáticas de qualquer espécie, Ramos (2003) propõe a representação de restrições deônticas no diagrama de classes. O Autor exemplifica o caso de um aluno que não sabe o código postal, ao efectuar a matrícula num estabelecimento de ensino. Neste caso o sistema deve registar a violação da obrigação, de ter código postal, imposta pela restrição deôntica, recorrendo a uma classe, violação, para o seu registo. A figura 1.1 ilustra esta proposta. O operador (XOR) significa que se o aluno não tem código postal, então a violação é associada ao aluno. A ligação entre as classes Aluno/Violação e Aluno/Código Postal representam associações entre classes. No capítulo dois é explicado, em detalhe, o diagrama de classes.

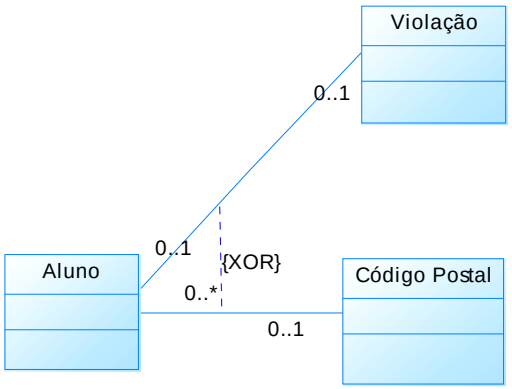


Figura 1.1. Representação de uma restrição deôntica, o estudante deve ter código postal caso contrário violação deve ser registada.

Nota. Figura transposta de Ramos (2003, p. 5).

Na óptica de utilização de uma base de dados relacional, é necessário traduzir a restrição deôntica representada na Figura 1.1 para o modelo relacional correspondente. Esta tradução justifica-se porque o modelo relacional define a arquitectura de armazenamento da informação e as regras para a sua manipulação.

Existem no mercado variadas ferramentas que fazem a transposição automática do modelo de classes para o modelo relacional, que se baseiam num conjunto de regras genéricas. Segundo Ramos (2006), esse conjunto de regras não deve ser entendido como um conjunto de leis rígidas, mas antes como um guia adaptativo ao problema específico em questão.

No caso exemplificado na figura 1.1, as classes Aluno, Violação, e Código Postal geram, segundo as regras genéricas referidas, tabelas com os mesmos nomes e atributos, e a cada tabela deve-se fazer corresponder uma chave primária. No caso da associação Aluno/Violação, que se caracteriza por ser uma associação do tipo "Um para um", a aplicação das regras determina que as tabelas Aluno e Violação herdem uma da outra as respectivas chaves primárias, caracterizando-as como chaves estrangeiras. Na associação Aluno/Código Postal, do tipo "Um para muitos" a regra difere, neste caso a tabela Aluno herda a chave primária de Código Postal e caracteriza-a como chave estrangeira. A figura 1.2 ilustra a transposição do diagrama de classes para o modelo relacional correspondente, o operador "XOR" não é traduzido porque não existem regras para a sua tradução. No capítulo dois é explicado o modelo relacional, e as regras de mapeamento entre o diagrama de classes e o modelo relacional.

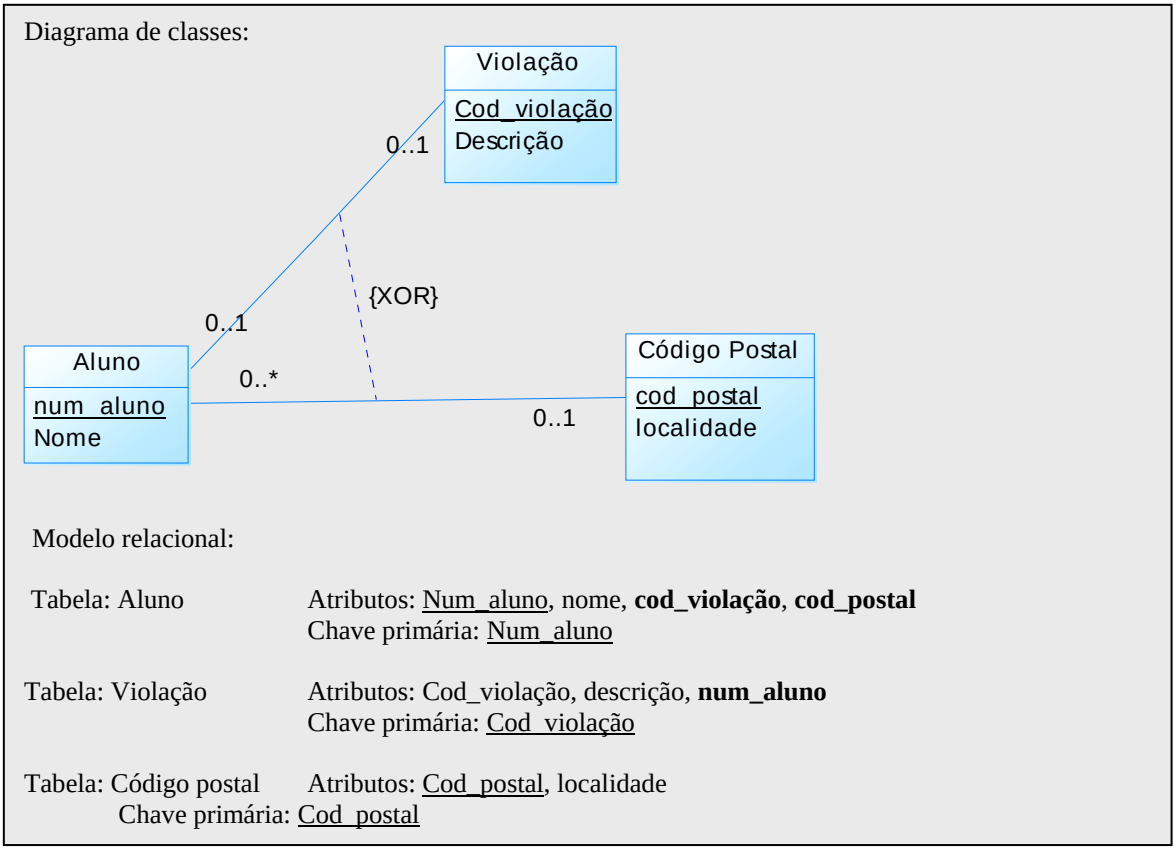


Figura 1.2. Representação de uma transição do diagrama de classes para o modelo relacional.

Nota. Figura adaptada de Ramos (2006, p. 61).

Apesar das restrições deônticas resolverem um número elevado de casos reais, e de ser possível a sua transposição para o modelo relacional, existem casos em que os contornos da realidade vão mais longe. Nestes casos, atípicos, a violação à regra é registada, mas não representa a realidade em toda a sua extensão. A razão é o facto da violação da regra dar origem a novas restrições. Para resolver esta questão, num contexto de lógica deôntica, Carmo e Jones (1996) citados por Ramos (2003) introduzem a noção de um novo tipo de restrições deônticas: "Contrary-to-duties", que se caracterizam por serem restrições que representam estados sub-ideais em que as obrigações são violadas e em consequência dessas violações surgem novas restrições, que lidam com a indesejada, mas no entanto tolerada, situação.

Como referido, a lógica deôntica é expressa por meio de lógica matemática, sem preocupações de representação diagramática, por isso Ramos (2003) propõe a representação das restrições deônticas no diagrama de classes, através de dois tipos destas restrições:

a) Proibição – Restrição que proíbe uma determinada associação de dados, até que uma obrigação seja cumprida.
b) Obrigação – Restrição que torna obrigatória determinada associação de dados, enquanto uma obrigação não for cumprida.

Um exemplo de aplicação da restrição "Proibição", apresentado por Ramos (2003), corresponde à necessidade que tem determinada organização de saber o código postal dos clientes para poder processar pedidos de encomenda. Nos casos em que o cliente não sabe o código postal é registada a violação no sistema, no entanto paralelamente deve emergir uma nova restrição: não podem ser feitos pedidos de encomenda sem que a obrigação de ter código postal seja satisfeita.

A representação desta restrição no diagrama de classes coloca um novo desafio na medida em que não existe, em UML, simbologia gráfica capaz de ilustrar a situação descrita. Para que seja possível representar este tipo restrições é necessário recorrer aos mecanismos de extensão da UML. Basicamente esses mecanismos permitem criar novos

elementos gráficos, de modo a estenderem as capacidades dos modelos existentes e assim conseguirem caracterizar diagramáticamente situações específicas.

Na figura 1.3 é exemplificada a representação deste caso recorrendo a dois novos elementos gráficos: o estereótipo "Proibido" e a correspondente seta a tracejado, que significam que a associação entre cliente e pedido só é possível se a associação entre cliente e código postal for cumprida. A UML e os mecanismos de extensão são explicados em detalhe no capítulo dois.

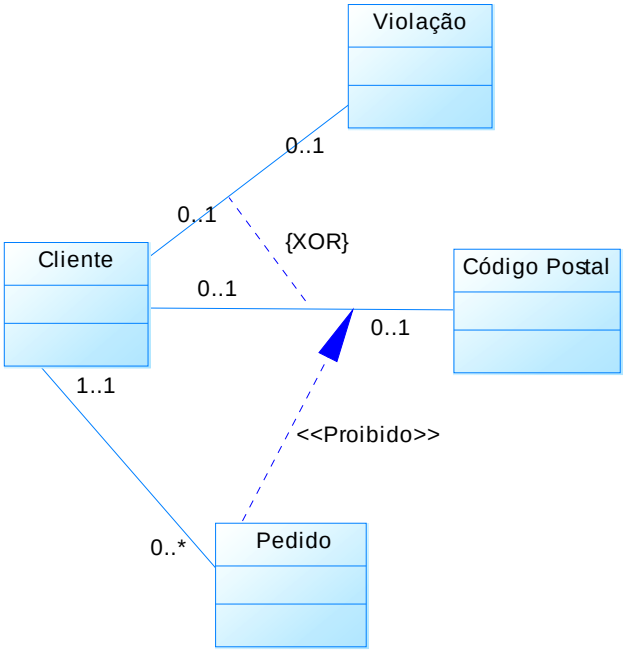


Figura 1.3. Representação da restrição "Proibição", é proibida a associação entre cliente e pedido até que a associação entre cliente e código postal seja cumprida. Nota. Figura transposta de Ramos (2003, p. 10).

A análise da figura 1.3 sugere uma questão: será que o novo símbolo detém o significado que pretendemos? (i.e. será que o sistema está informado que a associação entre Cliente/Pedido não é permitida, até ao momento em que a obrigação entre Cliente/Código postal seja cumprida?). A resposta é não, os novos elementos são apenas símbolos gráficos, cujo significado semântico se reduz à condição de elemento. É, portanto, necessário enriquecer esses elementos gráficos com o significado semântico das restrições que se pretendem implementar….

Segundo Rumbaugh, Jacobson e Booch (1999), uma restrição é um condicionamento semântico que se pode representar através de expressões textuais. Essas restrições podem ser escritas formal ou informalmente, respectivamente através de linguagens como a matemática, linguagens de programação, e linguagens de implementação de restrições baseadas em computador; ou, através de pseudo código, e linguagens informais. As linguagens informais são ambíguas e por vezes pouco claras, por isso é recomendável a utilização de linguagens formais para caracterizar restrições aos modelos de UML. A OCL[4] é definido por Warmer e Kleppe (2003) como sendo uma linguagem standard, faz parte da UML, e é capaz de descrever expressões que implementam restrições sobre os seus modelos de forma clara e não ambígua. Os autores consideram-no vital para adicionar informação aos modelos orientados a objecto. A sua especificidade e integração no motor da UML justificam a sua escolha para a implementação de restrições aos modelos, mais precisamente ao modelo de classes em UML.

No decorrer deste estudo, em resposta à necessidade de enriquecer os elementos gráficos do modelo de classes com o significado semântico das restrições a implementar, será utilizada a linguagem de restrições OCL. A linguagem OCL é explicada formalmente no capítulo dois.

Em harmonia com o exemplo da figura 1.1, é necessário, também, traduzir o diagrama de classes da figura 1.3 para o correspondente modelo relacional. A questão que se levanta é que o estereótipo “Proibido” não tem tradução, recorrendo às regras de mapeamento entre o diagrama de classes e o modelo relacional.

As razões para a não existência de tradução, no caso deste tipo de situações, são as limitações que o modelo relacional impõe. Segundo Ramos (2003), devido a essas limitações as restrições deônticas só podem ser expressas através de código. O Autor sugere, por isso, a implementação de triggers, para que as restrições representadas no diagrama de classes possam ser traduzidas em código e mantidas desta forma no modelo relacional.

O presente estudo pretende fazer o levantamento e descrição pormenorizados de duas situações de utilização real de sistemas de informação, que pela sua excepcionalidade se enquadrem no domínio das restrições deônticas. Recorrendo à linguagem OCL pretende-se

[4] OCL é o acrónimo para Object Constraint Language, é uma linguagem de modelação e construção de modelos de software. É parte integrante da UML.

enriquecer o diagrama de classes com as restrições necessárias para caracterizar de forma consistente as duas situações referidas. Finalmente, pretende-se que o modelo relacional correspondente a cada um dos diagramas de classes seja automaticamente gerado, com recurso à linguagem de programação Prolog e à implementação de triggers, na geração do modelo relacional. Os triggers são explicados no capítulo dois.

1.2 Apresentação do problema

O diagrama de classes em UML tem como objectivo representar a realidade através da modulação da estrutura e das relações entre classes de objectos.

Numa abordagem inicial a introdução de restrições fortes (obrigatórias) nas associações entre classes caracteriza parte dessa realidade, mas fá-lo de forma inflexível, não permitindo a caracterização de situações que não podem depender exclusivamente de um valor booleano (sim ou não). Nestes casos, excepcionais, a inflexibilidade das restrições obrigatórias impossibilita a caracterização de alguns contornos da realidade, que assumem particular relevância no contexto da situação respectiva, e que se repercutem na qualidade final do sistema de informação.

Poder-se-ia pensar que retirando a restrição inicialmente imposta não haveria problemas (como não havia restrição seria possível violar a regra), no entanto, a caracterização da realidade sairia empobrecida e o sistema ficaria sem saber que a regra foi violada.

Para que seja possível violar as regras inicialmente definidas e registar de forma inequívoca as violações correspondentes é, portanto, necessário devolver flexibilidade às restrições fortes (obrigatórias) entre classes.

1.3 Metodologia

É proposto o levantamento e descrição pormenorizados de duas situações de utilização real de sistemas de informação, que se caracterizem pela impossibilidade de serem totalmente modeladas através da utilização de restrições obrigatórias.

Com recurso à introdução, no diagrama de classes da UML, de restrições deônticas e da utilização da OCL pretende-se modelar todas as propriedades de excepção que decorrerem das situações atípicas descritas.

Propõe-se a utilização da linguagem de programação Prolog, na implementação do desenvolvimento necessário, que permita a geração automática do diagrama relacional correspondente, reflectindo todas as propriedades, já modeladas, pelo diagrama de classes.

1.4 Objectivos

O objectivo operacional do presente estudo corresponde ao levantamento e representação de duas situações reais de utilização de sistemas de informação em que se justifique a necessidade de utilização de restrições deônticas. O objectivo referido decompõe-se em três fases distintas:

a) representação, no diagrama de classes da UML, de dois casos reais em que se justifique a utilização de restrições “Contrary to duties”;
b) formação do diagrama de classes correspondente; e
c) elaboração do automatismo para gerar o diagrama relacional, a partir do diagrama de classes.

Os objectivos estratégicos do presente estudo fundem-se com o interesse do tema no âmbito do desenvolvimento e gestão de sistemas de informação e pretendem:

1) contribuir para melhorar as ferramentas de geração automática de código, mais concretamente acrescentar melhorias às ferramentas que geram bases de dados;

2) contribuir para o enriquecimento da documentação gráfica que descreve sistemas de informação, com mais e melhor informação; e
3) contribuir, com a adição de semântica e informação gráfica, facilmente apreendida pelas diversas áreas de conhecimento, para a melhoria da comunicação e da compreensão colectivas, dos sistemas de informação.

1.5 Limitações do estudo

O resultado desta investigação, de natureza qualitativa, operacionalizado com recurso ao levantamento de dois casos reais de utilização de sistemas de informação, é válido para todos os casos que se enquadrem na arquitectura genérica dos dois tipos de restrições "Contrary to duties", considerados para o presente estudo. A arquitectura genérica, das restrições referidas, será descrita em detalhe, no decorrer do capítulo três.

Os resultados obtidos por este estudo, podem servir para que numa investigação futura seja possível generalizar a representação das restrições "Contrary to duties", num contexto mais alargado de arquitecturas possíveis para este tipo de restrições.

1.6 Estrutura da dissertação

A modelação de sistemas e a necessidade de representar a realidade dos processos de negócio, nas organizações, constitui a essência deste estudo. Por vezes, fruto da utilização em real de sistemas de informação, deparamo-nos com situações extraordinárias cujos contornos da realidade não são passíveis de ser modelados. Com recurso à UML, e introduzindo alguns conceitos de lógica deôntica pretende-se adicionar capacidade semântica ao diagrama de classes da UML, de forma a representar este tipo de situações. Paralelamente, é pretendida a geração automática do modelo relacional correspondente. Neste enquadramento, a organização da estrutura do estudo é desenhada em função do processo de investigação desenvolvido.

No primeiro capítulo, introdutório, é feito o enquadramento temático do estudo, seguido da apresentação do problema que motiva esta investigação e da descrição dos seus objectivos e delimitações.

No segundo capítulo é feito o enquadramento teórico das diversas áreas que compõem o tema em estudo.

O terceiro capítulo enquadra o problema que serve de base a este estudo e descreve em pormenor a solução genérica que se propõe para a resolução do mesmo. Seguidamente descreve-se a metodologia a utilizar em cada uma das fases de tradução das restrições "Contrary to duties" consideradas neste estudo. A solução genérica será apresentada tendo em conta três fases: Caracterização genérica das restrições "Contrary to duties", no contexto do diagrama de classes; aplicação da metodologia para criação do modelo relacional correspondente; e criação do automatismo para geração automática do modelo relacional.

No quarto capítulo, são descritos e caracterizados dois casos reais de utilização de sistemas de informação que pelo carácter atípico justificam a utilização de restrições "Contrary do duties", em seguida é aplicada a metodologia genérica definida no capítulo três aos dois casos, no sentido de proceder à caracterização dos mesmos no diagrama de classe e relacional.

No quinto e último capítulo, são sintetizados os resultados obtidos com este estudo, e confrontados com os objectivos inicialmente estabelecidos. Por fim são feitas as considerações finais, acompanhadas de algumas sugestões para investigações futuras.

1.7 Contributos para o desenvolvimento da dissertação

Para a persecução dos objectivos do presente estudo, dada a diversidade das áreas temáticas abordadas, torna-se aconselhável distinguir objectivamente os contributos provenientes de fontes de informação e conhecimento já existentes, em contraste com o contributo do autor da dissertação, para o efeito a figura 1.4 sintetiza as duas vertentes referidas, necessárias ao desenvolvimento da dissertação.

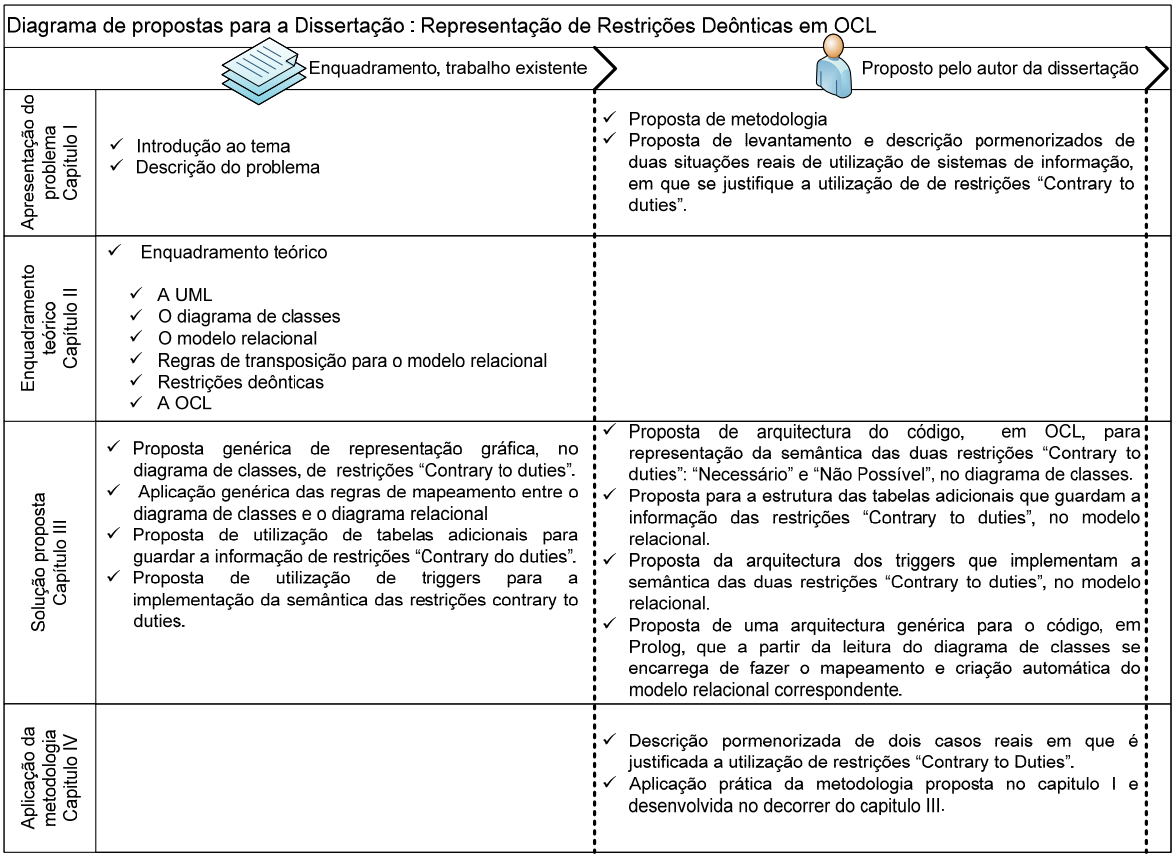


Figura 1.4. Síntese dos diferentes contributos relevantes ao desenvolvimento da dissertação.

Na figura 1.4 são perceptíveis os contributos para o desenvolvimento do presente estudo, o capítulo cinco não se encontra referenciado porque corresponde às conclusões, do autor da dissertação.

Capítulo 2 – Enquadramento teórico

2.1 A UML.

2.2 Diagrama de classes.

2.3 Modelo relacional.

2.4 Do diagrama de classes para o modelo relacional.

2.5 Restrições deônticas.

2.6 A OCL.

2.1 A UML

A UML representa um esforço no sentido de simplificar, consolidar e unificar os diferentes métodos de desenvolvimento de software orientados ao objecto que começaram a surgir nos finais da década de 80. A proposta inicial da UML visava desenvolver uma linguagem de modelação comum aos métodos referidos, que fosse bem aceite pelos fabricantes de ferramentas de programação e pelos programadores (utilizadores finais). Em finais de 1997 o esforço empreendido e a concordância da maior parte da comunidade de modeladores deram frutos: a OMG adopta a UML como standard para a indústria de ferramentas de software.

Desde 1997, até aos dias de hoje, a UML tem vindo a consolidar-se de forma consistente, abrangendo um conjunto vasto de métodos orientados ao objecto com capacidade de modelar e documentar todos os aspectos relevantes do processo de desenvolvimento de sistemas de informação. A UML é não proprietária, por isso de utilização livre, característica que, em conjunto com o facto de ser aceite como standard, potenciou a que a maior parte dos fabricantes de software da actualidade o tivessem incluído e integrado nas suas ferramentas de software.

A UML é uma linguagem de modelação que especifica e documenta, com recurso a modelos, sistemas de informação. Os modelos que constituem a UML permitem descrever a estrutura, o comportamento, e a interacção dos sistemas que se pretendem desenvolver de forma abrangente e sistemática.

Segundo Rumbaugh, Booch e Jacobson (1999) os modelos têm dois importantes aspectos a considerar: informação semântica e representação visual. A informação semântica diz respeito ao significado intrínseco do modelo e dos elementos que o constituem, a representação visual mostra a informação semântica de forma a torná-la imediatamente perceptível ao entendimento humano. A representação visual de um modelo corresponde ao diagrama desse modelo.

Os modelos não devem ser vistos como documentos isolados, mas antes como um conjunto integrado de informação que serve o propósito de descrever e documentar todos os aspectos relevantes de determinado sistema. O nível de abstracção dos modelos vai variando consoante o objectivo pretendido, segundo Rumbaugh, Booch, e Jacobson (1999) o modelo captura os aspectos essenciais de um sistema e ignora os outros. Os aspectos que

são essenciais para um modelo dependem do julgamento do modelador relativamente ao objectivo específico que se pretende atingir.

Segundo os mesmos autores o nível de detalhe dos modelos varia, também, consoante os objectivos e tem que ser adaptado ao propósito a que se propõe, por exemplo o nível de detalhe de um documento que serve para apresentar à gestão é necessariamente menor que o nível de detalhe necessário para especificar geração de código numa determinada linguagem de programação.

Sobre a égide da OMG, a UML sofreu, desde 1997 alguns desenvolvimentos adicionais, que deram origem a uma nova versão suportada por um conjunto de treze modelos específicos, que Silva e Videira (2005) agrupam em três visões específicas:

1) Visão estática ou estrutural: Diagrama de classes, Diagrama de objectos, Diagrama de componentes, Diagrama de estrutura composta, Diagrama de pacotes e Diagrama de instalação;

2) Visão Funcional: Diagrama de casos, Diagrama de actividade e Diagrama de estados;

3) Visão Dinâmica: Diagrama de Sequência, Diagrama de Comunicação, Diagrama Temporal e Diagrama de Interacção.

Uma das preocupações na filosofia do desenho inicial da UML foi mantê-la aberta e extensível, antecipando futuros desenvolvimentos. A utilização integrada dos modelos da UML permite descrever a quase totalidade dos aspectos relevantes a ter em conta no processo de desenvolvimento dos sistemas de informação, mas não todos. Nos casos em que não é possível descrever os contornos particulares de determinado sistema com os modelos disponíveis a UML oferece a possibilidade de recorrer a três tipos de mecanismos de extensão: estereótipos, marcas com valor e restrições.

Silva e Videira (2005) definem os mecanismos de extensão: o estereótipo é um mecanismo de classificação de elementos que pode ser utilizado para implementar as suas próprias restrições, pode criar novos ícones para representação visual e incluir marcas com valor, e pode, ainda, ser um subtipo de outros estereótipos e herdar deles restrições e marcas com valor; as marcas com valor são um par <marca,valor> que permite associar

informação aos elementos de qualquer modelo; finalmente, as restrições consistem em especificar a semântica de um ou mais elementos dos modelos, essa especificação pode ser feita com recurso a linguagens formais ou menos formais, como por exemplo linguagens de programação, OCL, notação matemática, ou linguagem natural.

2.2 O diagrama de classes

Um objecto corresponde em geral a uma entidade do mundo real com características e comportamento intrínsecos. Por exemplo, se considerarmos um automóvel como um objecto temos que definir uma série de características como a cor, o tamanho, ou o formato. Para além das características também importa saber como se comporta o automóvel quando o acelerador é impulsionado, ou quando se vira o volante.

Apesar das características e do comportamento diferirem de veículo para veículo: cores diferentes, comportamentos diferentes, etc., cada objecto veículo partilha com os outros a necessidade de ter uma cor, de ter uma forma, ou de ter um comportamento ao virar o volante. Essa partilha corresponde à estrutura do objecto, que pode ser representada recorrendo à utilização de uma classe. Silva e Videira (2005), definem uma classe como sendo a descrição de um conjunto de objectos que partilham atributos, semântica, operações e relações. Para os Autores a classe é uma fábrica de objectos e o objecto uma instância da classe que o cria.

As classes representam-se com recurso a um rectângulo, conforme descrito na Figura 2.1. Dentro do rectângulo a primeira divisão corresponde ao nome da classe; o rectângulo interior do meio corresponde à listagem dos atributos; e por último, o rectângulo que resta corresponde à listagem das operações.

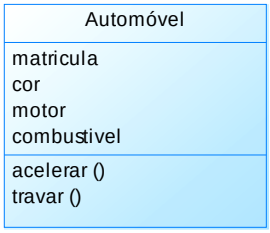


Figura 2.1. Representação gráfica de uma classe.

Num sistema, os objectos relacionam-se entre si para que a representação do mundo real faça sentido, por exemplo o automóvel relaciona-se com o condutor, uma empresa relaciona-se com os fornecedores; o fornecedor relaciona-se com os produtos que fornece, etc. O relacionamento entre objectos é representado pela estrutura que os cria, ou seja, pela relação entre as classes que criam esses mesmos objectos.

O diagrama de classes da UML corresponde à representação visual da estrutura e relacionamentos entre as classes que compõem determinado sistema, por isso é considerado um dos diagramas de estrutura mais importantes no processo de descrição de sistemas de informação.

Existem dois tipos de relações possíveis na UML: associações e generalizações, as associações consideram ainda dois tipos especiais de associação: agregação e composição.

As associações especificam relacionamentos entres objectos de diferentes classes, que impõem algumas restrições. No diagrama de classes, as associações são representados por intermédio de uma linha que une as suas extremidades às classes que pretende associar. A associação poderá ter um nome que distingue o seu propósito, ou não, conforme a necessidade do modelador. A cardinalidade define os limites das ligações entre os objectos e é representada por intervalos de valores colocados nas extremidades da associação. As cardinalidades de cada terminação podem ter um nome associado, o seu propósito é identificar o papel da cardinalidade. Ramos (2006) dá um exemplo prático de três possíveis associações, categorizando-as conforme descrito na Figura 2.2.

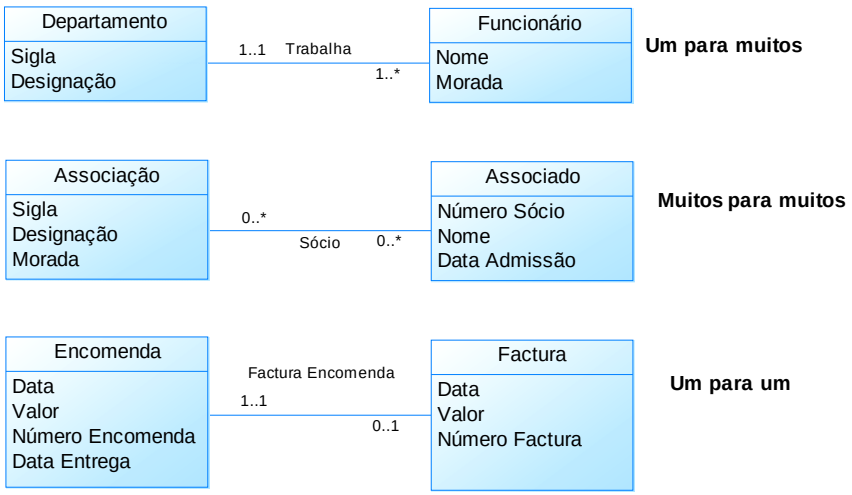


Figura 2.2. Representação e classificação de três associações classificadas segundo os limites superiores.

Nota. Figura adaptada de Ramos (2006, p. 26).

Na Figura 2.2, a associação Trabalha impõe um significado semântico à relação entre Departamento e Funcionário: o intervalo 1..* significa que o departamento tem que estar associado a um funcionário (porque o limite inferior do intervalo é 1) e pode estar associado a infinitos funcionários (o símbolo * representa infinito); o intervalo 1..1 significa que um funcionário tem que estar associado a um departamento (porque o limite inferior do intervalo é 1) e um funcionário só pode estar associado a um departamento (porque o limite superior é 1). Note-se que a semântica da associação impõe duas restrições: um funcionário não pode trabalhar em mais de um departamento e um departamento tem que ter pelo menos um funcionário. Nas associações subsequentes, Sócio e Factura Encomenda, o significado semântico lê-se pelo processo já descrito em "Trabalha", o valor que falta explicar é o 0, que nos limites inferiores dos intervalos significa que um dos objectos pode existir sem estar associado ao outro.

A agregação é um caso especial de associação, utiliza-se quando se pretende realçar que um objecto corresponde à agregação de outros objectos. A figura 2.3 dá um exemplo desta associação, no exemplo a semântica significa que o conjunto dos objectos do tipo "peça" fazem parte do objecto "automóvel". É importante realçar que os objectos "peça" podem existir independentemente do objecto "automóvel".

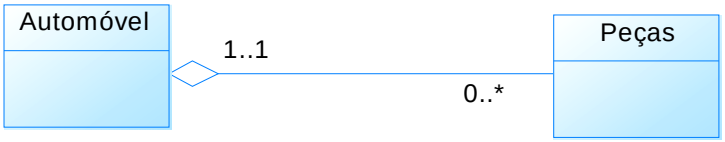


Figura 2.3. Representação de uma agregação.

A agregação é representada por um losango representado graficamente junto da classe que representa o objecto que agrega outros objectos. A finalidade da agregação é dar a ideia de um objecto único.

A composição é outro dos casos especiais de associação, neste caso representa uma agregação mais forte, ou seja algo que é composto e cuja composição é indissociável. A principal diferença entre a agregação e a composição verifica-se ao nível dos objectos que se agregam, no caso da agregação esses objectos têm existência própria; no caso da composição não faz sentido a existência sem estarem ligados ao objecto que as compõe.

Por exemplo, a associação entre um puzzle e as suas peças é um dos casos em que a existência das peças só faz sentido no contexto do puzzle, o que justifica que o exemplo da figura 1,7 seja caracterizado por uma composição.

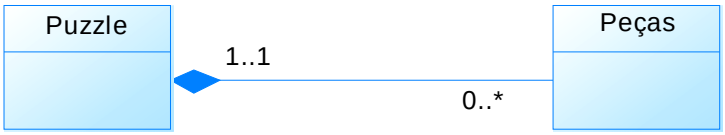


Figura 2.4. Representação de uma composição.

A composição representa-se da mesma forma que a agregação, mas com o losango preenchido com cor negra.

As generalizações são relações que se utilizam para realçar a existência de subconjuntos específicos dentro de um conjunto de objectos. A sua utilização serve para mostrar informação específica, só relevante para os subgrupos que se pretendem realçar.

A figura 2.5 exemplifica a utilização de uma generalização. A generalização corresponde a uma associação do tipo um para um e é representada graficamente por uma seta no sentido da supraclasse (classe que representa a totalidade dos objectos), isto significa que as cardinalidades 1..1 e 0..1 estão implícitas respectivamente do lado da supraclasse e das subclasses (classes que representam um determinado subconjunto de objectos).

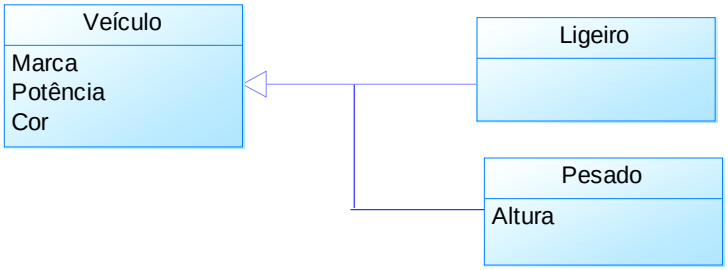


Figura 2.5. Representação gráfica de uma generalização.

O significado semântico da Figura 2.5 indica que o objecto veículo pode ser ligeiro ou pesado, mas não pode ser ligeiro e pesado. Note-se que no exemplo o veículo pode não ser nenhum dos dois: nem ligeiro nem pesado.

2.3 O modelo relacional

O modelo relacional surgiu na década de 70 pela mão de E. J. Codd, é utilizado ainda hoje pela generalidade das bases de dados para armazenar informação. Baseia-se num conjunto de conceitos básicos, simples e generalistas, que permitem organizar estruturadamente a informação através de relações, tem origem na álgebra relacional e na teoria matemática dos conjuntos.

Segundo Codd (1990), uma relação no modelo relacional é similar a uma relação no modelo matemático. Considerando que a relação corresponde a uma tabela o autor distingue a relação (tabela) do modelo relacional: cada coluna da tabela tem um nome único, identificativo, que representa um domínio próprio (conjunto de valores que pode assumir), os valores que pode assumir são atómicos, ou seja não podem ser uma composição de valores com significados diferentes. Na Figura 2.6 é dado um exemplo de uma tabela relacional.

Automóvel				
Matrícula	Marca	Modelo	Motor	Potência
23-CD-90	Audi	A3	1900	115
11-AV-31	Opel	Astra	1700	105
35-ER-36	Nissan	Qashqai	1500	105

Figura 2.6. Representação da tabela Automóvel.

Dentro da tabela é necessário assegurar que cada um dos registos pode ser identificado univocamente através da inclusão de uma chave primária.

A chave primária identifica o registo através dos valores de determinado atributo, ou conjugação de atributos. Os valores do atributo, ou atributos, que formam a chave primária não se podem repetir, caso contrário não é garantida a unicidade de cada registo; para além da repetição de valores, não é permitido, na coluna de atributo da chave primária, valores nulos, porque existiriam registos não identificados; por último e por razões de performance convém que a chave primária se forme a partir de valores numéricos, sempre que possível a partir de um só atributo.

Em situações pontuais, embora muito raras, uma tabela pode ser suficiente para guardar a informação de determinado sistema, mas normalmente necessitamos de várias para que a

informação seja correctamente representada, tenha consistência, e promova a rapidez e facilidade de utilização da base de dados.

Considerando que pretendemos guardar dados de automóveis e correspondentes peças, a matrícula seria um bom candidato a chave primária porque existe a garantia que é única para cada automóvel, no entanto não é numérica, o que em termos de performance não é aconselhado. Na tabela, a organização de dados poderia ser a apresentada na Figura 2.6. Para efeitos do exemplo considera-se a matrícula como chave primária, embora não seja a opção mais correcta.

Até agora a informação da tabela da Figura 2.6 é consistente, e não existe duplicação de informação, no entanto é necessário representar as peças que estão indicadas para cada automóvel. Uma hipótese seria adicionar mais atributos à tabela, para guardar todas as peças. Como cada automóvel tem mais que uma peça, isso significa que o automóvel teria que ser registado tantas vezes quantas as peças que o compõem. A Figura 2.7 representa a opção referida, incorrecta.

Automóvel						
Matrícula	Marca	Modelo	Motor	Potência	Número Série Peça	Descrição
23-CD-90	Audi	A3	1900	115	185543-TRT-4958	Junta
23-CD-90	Audi	A3	1900	115	25355-TGR-2345	Depósito
11-AV-31	Opel	Astra	1700	105	894764-EFG-3856	Rotor
11-AV-31	Opel	Astra	1700	105	239837-ADT-6908	Tuche
35-ER-36	Nissan	Qashqai	1500	105	222789-WER-7765	Alternador

Figura 2.7. Representação da tabela Automóvel estendida aos atributos das peças.

Não é conveniente duplicar desnecessariamente a informação por isso a opção correcta seria estruturar a informação relativa às peças numa nova tabela. Ao dividir a informação coloca-se uma importante questão: como relacionar a informação das duas tabelas? É necessário garantir que não existem peças sem estarem relacionadas com automóveis, ou saber para cada automóvel, que peças existem. O conceito de chave estrangeira resolve estas questões e garante a integridade parcial do sistema. Uma chave estrangeira corresponde a um atributo ou conjunto de atributos que correspondem ao domínio da chave primária de outra tabela. A figura 2.8 ilustra a nova representação de dados.

Automóvel				
Matrícula	Marca	Modelo	Motor	Potência
23-CD-90	Audi	A3	1900	115
11-AV-31	Opel	Astra	1700	105
35-ER-36	Nissan	Qashqai	1500	105

Chave primária - Matrícula

Peças		
Número Série	Modelo	***Matrícula***
185543-TRT-4958	Junta	23-CD-90
25355-TGR-2345	Depósito	23-CD-90
894764-EFG-3856	Rotor	11-AV-31
239837-ADT-6908	Tuche	11-AV-31
222789-WER-7765	Alternador	35-ER-36

Chave primária – Número série

Chave estrangeira **- Matrícula**

Figura 2.8. Representação da informação com recurso à partição em duas tabelas.

Numa base de dados não basta criar a estrutura de armazenamento, é necessário também introduzir, alterar, apagar e pesquisar informação, em suma é necessário manipular a informação, respeitando a integridade dos dados.

O SQL[5] é a solução para manipular dados, surgiu na década de 70 e foi considerado standard nos finais da década de 80, actualmente é utilizado pela maioria das bases de dados para manipulação e gestão da informação, contém uma linguagem própria de comandos que permite delimitar, inserir, actualizar, e apagar informação relacional.

Os comandos disponíveis no SQL são intuitivos, por exemplo o comando INSERT serve para inserir registos de dados, o comando SELECT delimita o domínio da informação, o comando UPDATE altera registos de dados, e o comando DELETE apaga os registos de dados.

Embora à primeira vista a sua utilização seja facilitada pela aproximação clara à linguagem natural, a verdade é que relacionar informação, para dar resposta aos sistemas do mundo real, é uma actividade mais ou menos complexa, que exige algum esforço e concentração. Para simplificar a tarefa de relacionar a informação, em sistemas mais complexos, o SQL disponibiliza alguns objectos especiais, esses objectos são normalmente guardados pela base de dados para utilização aplicacional.

[5] SQL é o acrónimo para Stuctured Query Language, linguagem de programação standard, capaz de pesquisar e modificar informação, e de gerir bases de dados.

Um desses objectos, o trigger, tem particular importância no decorrer deste estudo. O trigger corresponde a código procedimental que é automaticamente executado em resposta a determinado evento, ou conjunto de eventos. Existem algumas diferenças na implementação de triggers entre os principais fabricantes de bases de dados, mas basicamente existem três tipos de eventos que causam o disparo do trigger, ou seja que fazem com que o código procedimental seja executado: INSERT, UPDATE, e DELETE.

2.4 Do diagrama de classes para o modelo relacional

A utilização do modelo de classes para descrever a estrutura e relacionamento dos objectos que compõem determinado sistema de informação torna necessário, numa fase posterior, a sua transposição para uma base de dados relacional. Essa transposição pode ser feita automaticamente por variadas ferramentas disponíveis no mercado, segundo um conjunto de regras genéricas que garantem a coerência entre a modelação do sistema (diagrama de classes) e o modelo de dados que a suporta (diagrama relacional).

(Ramos, 2006) descreve esse conjunto de regras de transposição, que caracteriza como genéricas:

Regra 1 (classes)
Todas as classes e associações do tipo «muitos para muitos» dão origem a tabelas, e nada mais dá origem a tabelas. As tabelas não têm que de ter a mesma designação das classes ou associações que lhes deram origem;

Regra 2 (Atributos de classes)
Todos os atributos de uma classe são atributos da tabela que implementa a classe;

Regra 3 (Atributos de associações)
Todos os atributos de uma associação são atributos da tabela que ou (i) implementa a associação ou (ii) herda as chaves primárias das restantes tabelas que implementam as classes envolvidas na associação (ver restantes regras);

Regra 4 (Associações do tipo «um para um»)
As tabelas que implementam os argumentos da associação herdam a chave primária da(s) outra(s) tabela(s) como chave estrangeira;

Regra 5 (Associações do tipo «um para muitos»)
A tabela cujos registos são susceptíveis de serem relacionados com apenas um registo da outra tabela (lado muitos) herda como chave estrangeira a chave da(s) tabela(s) cuja correspondência é unitária (lado um);

Regra 6 (Associações do tipo «muitos para muitos»)

A associação dá origem a uma nova tabela representativa da associação, e a chave primária é composta pelos atributos chave das tabelas que implementam as classes associadas;

Regra 7 (composições)

A tabela que implementa a classe (componente) que representa os objectos, pelos quais os objectos de outra classe são compostos, tem como atributos da chave primária os atributos da chave primária da tabela que implementa a classe composta, e um outro atributo (ou mais) pertencente à classe componente (caso não exista é necessário criá-la)

Regra 8 (Generalizações)

Duas situações distintas podem ocorrer:

a) As subclasses têm existência própria independentemente das supraclasses: a chave primária das tabelas que implementam as subclasses é obtida através dos atributos da própria tabela. Terá de ser criada uma chave primária para a tabela que implementa a supraclasse, sendo que essa tabela deverá ter uma propriedade discriminante (um atributo) que indica qual das subclasses o registo diz respeito. Todos os atributos que compõem a chave primária da tabela que implementa a supraclasse terão de constar como chaves estrangeiras nas tabelas que implementam as subclasses.

b) As subclasses só têm identidade enquanto associadas à supraclasse: a chave da tabela que implementa a supraclasses é obtida através dos atributos da própria tabela. As tabelas correspondentes às subclasses herdam a mesma chave da tabela que implementa a supraclasse. (p..59)

O modelo relacional gerado, com a aplicação das regras de transposição, não é, a maior parte das vezes, um modelo eficiente no que respeita à performance e capacidade de resposta da base de dados. É necessário perceber que o propósito do modelo relacional difere do modelo de classes no que respeita ao fim a que se destina, no caso do modelo relacional a preocupação central é desenhar uma base de dados o mais eficiente possível, por isso alterações adicionais são por norma necessárias.

2.5 Restrições deônticas

Segundo Carmo e Jones (2002), a lógica deôntica preocupa-se com noções normativas. Uma norma representa e descreve um comportamento ideal, que é esperado mas que em determinadas situações pode ser desviado da sua idealidade, ou seja violado. Para Jones e Sergot (citado em Carmo & Jones, 2002) as potencialidades da lógica deôntica residem precisamente na possibilidade de violar normas.

Em (Carmo et al, 2001) é referido que as restrições deônticas expressam as normas de como as coisas deviam idealmente ser, mas, ao mesmo tempo, permitem a violação e registo dessas mesmas normas. A violação de normas, e o seu registo, torna perceptível

que o comportamento real das coisas se desvia, por vezes, do comportamento que seria o ideal.

Jones e Sergot (citado em Carmo & Jones, 2002) sugerem que a lógica deôntica se preocupa essencialmente em representar a distinção entre a realidade e a idealidade. Para que seja possível representar essa distinção, a lógica deôntica introduz quatro conceitos:

c) Forbidden - (P) Proibição – proibido fazer algo, mas se o fizer o sistema não rejeita.

d) Obligation - (O) Obrigação – obrigatoriedade de fazer algo, mas se não for feito, o sistema não rejeita.

e) Necessity – (N) Necessidade – tem sempre que ser feito algo, o sistema não admite excepções.

f) Possibility – (PO) Possibilidade – pode ser feito algo, o sistema não impede que seja feito.

Entre os conceitos referidos existem equivalências matemáticas, assim Obrigação(a) equivale a Proibição (not a); e Necessidade(a) equivale a not Possibilidade(not a).

Para os autores, a especificação da distinção entre a realidade e idealidade, referida anteriormente, sugere que se acrescentem normas secundárias que indicam o que deve ser feito nas circunstâncias em que o comportamento real se desvia do real. O conceito de restrição “Contrary-to-duties” surge, na lógica deôntica, como consequência daquilo que deve ser feito sempre que o comportamento real se desvia do ideal. Mais concretamente sempre que existir uma violação do conceito de “Obrigação”, anteriormente definido, surge uma consequência, que pode ser implementada por qualquer dos conceitos apresentados anteriormente.

No contexto da formação de restrições “Contrary to duties”, os conceitos de “Proibição”, “Possibilidade”, e “Necessidade” representam relações de dependência, que surgem, exclusivamente, como consequência da violação de uma obrigação. O conceito de “Obrigação” pode surgir como uma restrição incondicional, que descreve um estado de idealidade; ou como uma relação de dependência, que resulta da violação de uma

obrigação anterior. Assim a caracterização das restrições “Contrary to duties”, no contexto de lógica deôntica, é a seguinte:

a) Forbidden (Proibição) – Restrição que, idealmente, proíbe determinada associação de dados, até que uma obrigação seja cumprida.

b) Obligation (Obrigação) – Restrição que torna, idealmente, obrigatória determinada associação de dados, enquanto uma obrigação não for cumprida.

c) Necessity (Necessidade) – Restrição que torna necessária, sem excepções, determinada associação de dados, enquanto uma obrigação não for cumprida.

d) Possibility (Possibilidade) – Restrição que torna possível determinada associação de dados, enquanto uma obrigação não for cumprida.

As restrições “Obrigação”, e “Proibição” devem ser cumpridas sempre que a obrigação inicial, da qual dependem, for violada, no entanto, o conceito intrínseco que está na origem das duas restrições indica que também elas podem ser violadas, originando novas restrições, que idealmente deviam ser cumpridas. Desta forma, o aparecimento de novas restrições pode não ter fim. A restrição “Necessidade” equivalente a “not Possibilidade (not a)”, difere de “Obrigação” e “Proibição” porque embora decorrente da violação de uma obrigação inicial não pode ser violada, é obrigatória sempre que a obrigação inicial não for cumprida.

São várias as áreas do conhecimento a que se aplica a lógica deôntica, derivado ao interesse crescente de investigadores, atraídos pela versatilidade e flexibilidade dos seus conceitos. A aplicabilidade dos conceitos de restrição deôntica e “Contrary-to-duties” à área de restrições de integridade de bases de dados traz benefícios, porque pode devolver flexibilidade a restrições obrigatórias no que se refere à descrição dos relacionamentos entre objectos de determinado sistema de informação.

A lógica deôntica deriva da filosofia e é expressa por meio de lógica matemática, sem preocupações de representação diagramática, por isso Ramos (2003) propõe a representação das restrições “Contrary-to-duties” no diagrama de classes, com recurso a

elementos gráficos. A figura 2.9 exemplifica a representação diagramática das quatro restrições deônticas, anteriormente referidas.

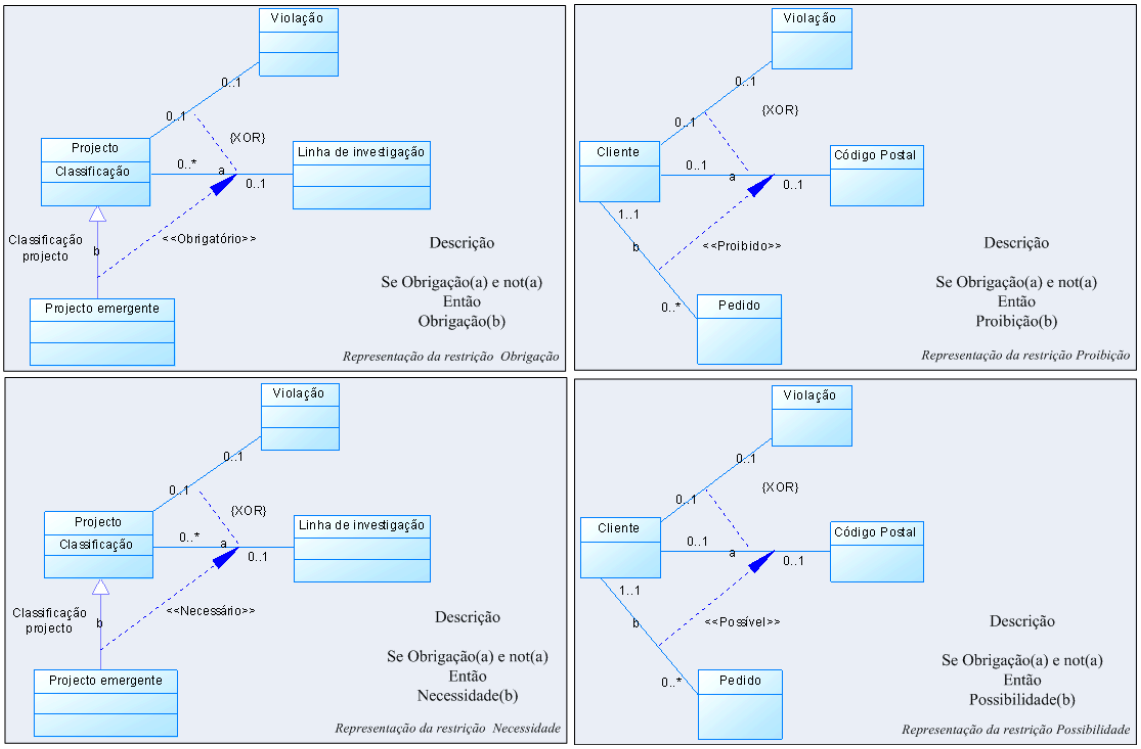


Figura 2.9. Representação das restrições deônticas Obrigação, Proibição, Necessidade e Possibilidade.

Nota. Figura adaptada de Ramos (2003, p. 10).

A abordagem descrita na figura 2.9 será desenvolvida durante os capítulos três, e quatro do presente estudo.

2.6 A OCL

O modelador, ao desenhar um sistema de informação, preocupa-se em descrever uma arquitectura que consiga dar resposta aos requisitos da organização. A UML disponibiliza um conjunto de elementos gráficos que se podem conjugar de forma a representar a estrutura dos sistemas de forma abrangente. Essa conjugação, dos elementos gráficos da UML, adiciona, também, uma percepção intuitiva e rápida do sistema de informação que

se pretende descrever; mas falha porque não é capaz de implementar restrições e regras, decorrentes da utilização em real desses mesmos sistemas.

Warmer e Kleppe (2003) referem que os diagramas da UML têm limitações que não lhes permitem descrever todas as propriedades relevantes de um sistema. Essas limitações acontecem porque a conjugação dos elementos gráficos, que cada diagrama disponibiliza, não é suficiente para descrever as restrições impostas pela realidade dos sistemas.

A OCL (Object Constraint Language) é uma linguagem de modelação que implementa restrições, faz parte da UML e é baseada na lógica matemática; corresponde por isso à necessidade de adicionar informação semântica, ou seja restrições aos diagramas de UML.

Por exemplo, a associação entre as classes voo e passageiro, descrita na figura 2.10 indica que determinado grupo de pessoas são passageiros de um voo. A cardinalidade 0..*, do lado da classe pessoa, indica que o número de passageiros para o voo não tem limite. O problema que se coloca é que o número de pessoas para determinado voo não pode ser ilimitado, mas sim limitado aos lugares disponíveis do avião. Como representar essa informação? Colocando no limite superior da cardinalidade, da classe pessoa, o número de lugares disponíveis no avião?.... A resposta é não, porque os lugares disponíveis variam com o avião, não são fixos de voo para voo. Warmer e Kleppe (2003) dão a solução, conforme expressão escrita em OCL (rectângulo dobrado) na Figura 2.10.

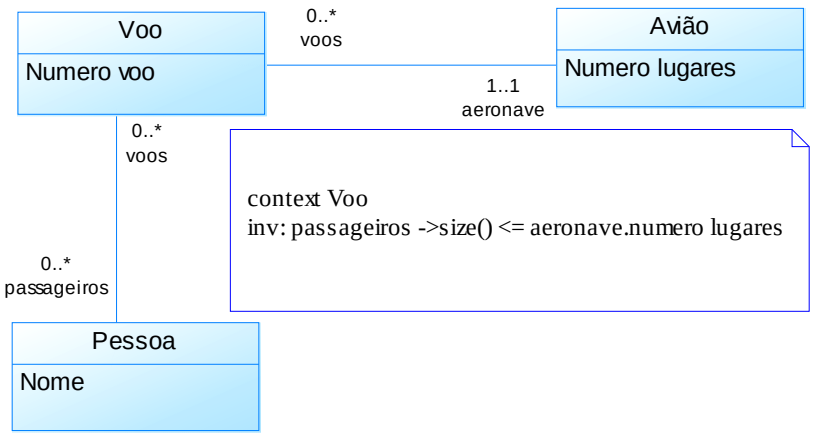


Figura 2.10. Representação das associações entre voos, aviões e passageiros. Com recurso a uma expressão escrita em OCL.

Nota. Figura adaptada de Warmer e Kleppe (2003).

As expressões escritas em OCL dependem dos elementos de modelação disponíveis em cada um dos modelos da UML, por isso é necessário definir o seu contexto, ou através de uma ligação directa ao elemento que pretendemos referenciar (no diagrama a que o elemento pertence); ou através de uma referência explicita ao elemento, escrita num ficheiro de texto. A figura 2.10 exemplifica a utilização de uma referência em ficheiro, através da linha de código "context Voo". No exemplo dado, a linha de código referida define a classe Voo como o alvo a que se destina a expressão escrita em OCL, ou seja a classe Voo é definida como contexto. A palavra "context" é uma das palavras reservadas da OCL e define, precisamente, o contexto das expressões em OCL.

Para além do contexto é importante perceber o tipo de elemento que é contextualizado, ou seja importa saber para que tipo de objecto se dirige a expressão em OCL. A importância de saber qual o tipo de objecto definido como contexto reside no facto das operações diferirem para tipos diferentes, por exemplo as operações disponíveis para uma classe diferem das operações disponíveis para uma interface, um tipo de dados ou um componente. Na figura 2.10, os atributos, associações e operações da classe Voo podem ser directamente referenciados por código em OCL. Isto é possível porque a classe Voo foi inicialmente definida como sendo o contexto das expressões em OCL. Assim a linha de código "inv:passageiros-->size()<=aeronave.numerolugares" referencia as extremidades das associações entre as classes Voo/Avião e Voo/Pessoa, e possibilita que sobre as mesmas classes sejam feitas operações. A palavra reservada "inv" especifica que para todas as instâncias do tipo de objecto contextualizado, a expressão de OCL é verdade. No caso concreto do exemplo da figura 2.10 significa que para todas as instâncias da classe Voo a expressão a aplicar é verdadeira. Assim a expressão "inv:passageiros-->size()<=aeronave.numerolugares" indica que o tamanho da extremidade "passageiros" tem sempre que ser menor ou igual que o número de lugares definido no atributo "numero lugares" da classe Avião. Assim, a expressão em OCL adiciona informação relevante para o comportamento deste modelo: limita o número de passageiros ao número disponível de lugares por aeronave. Esta é sem dúvida uma representação mais consistente da realidade.

No diagrama de classes da UML, a cada extremidade de uma associação corresponde uma multiplicidade e, opcionalmente, um nome, que a distingue univocamente, no diagrama de classes. Ao atribuir um nome, único, a uma extremidade de uma associação, garantimos que a mesma não possa ser confundida com qualquer outra, no contexto do diagrama de classes a que pertence. Mas para quê garantir que não existem ambiguidades

entre as extremidades das associações? A resposta é porque assim se pode navegar pelo diagrama de classes. Efectivamente, desde que seja definido um elemento de partida, ou seja um contexto inicial, é possível navegar facilmente por todo o diagrama de classes, existe também uma outra razão, é que ao referenciar uma extremidade de uma associação é também referenciada a classe que lhe está mais perto, o que possibilita a exposição de todos os atributos e operações disponíveis para essa classe à OCL.

Da figura 2.11, pode ser retirado um exemplo de navegabilidade entre as extremidades "cliente1" e "cliente2". Supondo que é definido como contexto a classe "Pedido", podemos navegar para a extremidade "pedidos", e em seguida navegar na outra associação da classe "Cliente" até à extremidade "cliente2". Claro que navegar entre as associações de um diagrama de classes deve ter um objectivo, ou servir determinado propósito, caso contrário seria uma perca de tempo. Assim, o propósito de navegar entre as associações do diagrama representado pela figura 2.11 serve um objectivo bem definido, impor uma restrição ao relacionamento entre classes. Mais concretamente, pretende-se implementar uma restrição que analise cada instância da associação entre as classes "Cliente" e "Código Postal", se essa associação não for cumprida, então a associação entre as classes "Cliente" e "Pedido" deve ser proibida. O código, em OCL, que serve o objectivo referido está representado na figura 2.11.

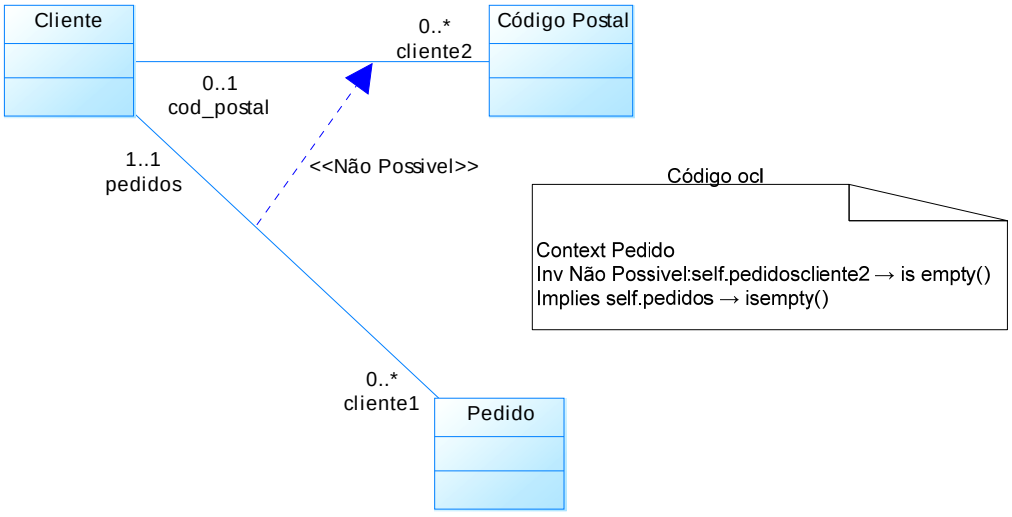


Figura 2.11. Representação de associações entre classes.

Nota. Figura adaptada de Ramos (2003).

A linha de código “Context pedido” referencia um elemento no diagrama de classes, a referência a esse elemento, específico, serve dois propósitos fundamentais: por um lado serve como referência inicial, a partir da qual se pode navegar pelo diagrama de classes e referenciar outros objectos; por outro define o tipo de objecto referenciado. A capacidade de referenciar tipos de objectos permite expor as operações disponíveis para cada um desses tipos, portanto transfere para o modelador a capacidade de questionar e manipular objectos. No exemplo da figura 2.11, pretende-se saber se a associação entre as classes “Cliente” e “Código Postal” é cumprida. Como a extremidade direita da associação entre as mesmas classes, “cliente2”, corresponde ao objecto específico que contém informação acerca do cumprimento ou não cumprimento dessa associação, duas acções têm que ser tomadas: navegar até esse elemento; e questionar o valor do objecto. O fragmento de código “Inv proibido:self.pedidos.cliente2 - > is empty()” implementa as duas acções referidas, ou seja navega da classe “Pedido” à extremidade “cliente2, e questiona se a associação foi cumprida. Se a associação não tiver sido cumprida, é implementada uma restrição que determina que outra associação, entre as classes “Cliente” e “Pedido”, não pode ser cumprida. Para o conseguir recorre-se novamente a um pequeno fragmento de código, em OCL: “Implies self.pedidos - > is empty()”, que implementa na prática a restrição pretendida.

Para os autores Warmer e Kleppe (2003) a combinação entre UML e OCL representa o melhor de dois mundos, os diferentes diagramas combinados com expressões escritas em OCL complementam-se na especificação dos modelos de software.

A OCL é uma linguagem precisa, não ambígua e fácil de compreender, embora assente na lógica e rigor matemáticos, está definida como standard pela OMG, na especificação de expressões que adicionam informação complementar aos objectos de modulação. Inicialmente as características da OCL apenas permitiam a implementação de restrições a valores e/ou a elementos de modulação, mas com a evolução da UML (actualmente 2.0) as suas capacidades foram acrescidas, permitindo escrever expressões que:

- Possibilitam pesquisas.
- Referenciam valores.
- Definem regras de negócio.
- Implementam condições.

Capítulo 3 – Solução Proposta

3.1 Introdução

Na perspectiva do modelador de sistemas de informação a realidade das organizações representa a essência do seu trabalho. Ao observar a organização o modelador traduz, sob forma de um sistema, a sua percepção da realidade.

Na UML, o diagrama de classes corresponde a um modelo de organização de dados que permite, recorrendo a elementos gráficos e intuitivos, representar sistemas. Assim, o modelador associa os elementos do diagrama de classes de forma a representar a realidade. O diagrama de classes permite, também, descrever objectos e suas relações, esses objectos correspondem, no mundo real, à relação entre entidades, por exemplo a entidade “factura” relaciona-se com um “número único” dentro da organização; no diagrama de classes essa representação corresponde a dois objectos: objecto “factura” e objecto “número único”, que, da mesma forma, se relacionam entre si.

A associação entre classes descreve a arquitectura dos sistemas de informação, define o comportamento esperado dos objectos, e possibilita impor restrições ao seu relacionamento. As restrições dependem de uma condição que pode ou não verificar-se, se a condição se verificar são criados os objectos que dependem da associação, se não se verificar então os objectos das classes, que dependem da associação, não podem existir. Esta restrição, forte, depende de um valor, “sim” ou “não”, do qual a criação dos objectos que fazem parte do relacionamento, depende.

Existem, no entanto, casos excepcionais em que as restrições fortes não são suficientes para representar a realidade, nesses casos, não chega impor uma restrição dependente de um valor booleano, “sim” ou “não”, em vez disso, é necessário implementar uma restrição de idealidade, que verifique a condição imposta e registe eventuais violações, mas que ao mesmo tempo permita a criação dos objectos que dependem do relacionamento. Nos casos referidos é proposta a utilização do conceito deôntico “Necessidade” em detrimento da utilização de restrições fortes, na realidade este conceito já tinha sido utilizado por Ramos (2003), embora os nomes atribuídos pelo autor às restrições que implementam este conceito não fossem os correctos: “Proibição” e “Obrigação” correspondem, conforme explicado no capítulo dois, a conceitos diferentes, uma vez que significam uma restrição, que surge após violação da obrigação inicial, mas que pode, também ela, ser violada, dando origem a mais restrições. Como a restrição deôntica “Necessidade” não permite a

violação das restrições impostas, após violação da obrigação inicial, o presente estudo propõe a sua utilização. A lógica deôntica e os conceitos referidos encontram-se descritos em detalhe no capítulo dois.

É necessário, no entanto adequar a utilização do conceito deôntico "Necessidade" a duas situações distintas, a primeira situação corresponde à obrigatoriedade, sem excepções, de cumprir uma associação de dados, decorrente da violação de uma obrigação; a segunda corresponde à necessidade de proibir, sem excepções, determinada associação de dados, decorrente, da mesma forma, da violação de uma obrigação. O presente estudo propõe a utilização do conceito de "Necessidade" nos dois casos, embora no segundo caso se pretenda proibir sem excepções determinada associação de dados. Assim considerando "a" uma qualquer associação de dados, proibir sem excepções "a" corresponde a utilizar "Necessidade (not a)", como "Necessidade (not a)" é equivalente a "not Possibilidade (a)" o presente estudo propõe a sua utilização para proibir sem excepções determinada associação de dados. A figura 3.1 ilustra as duas situações.

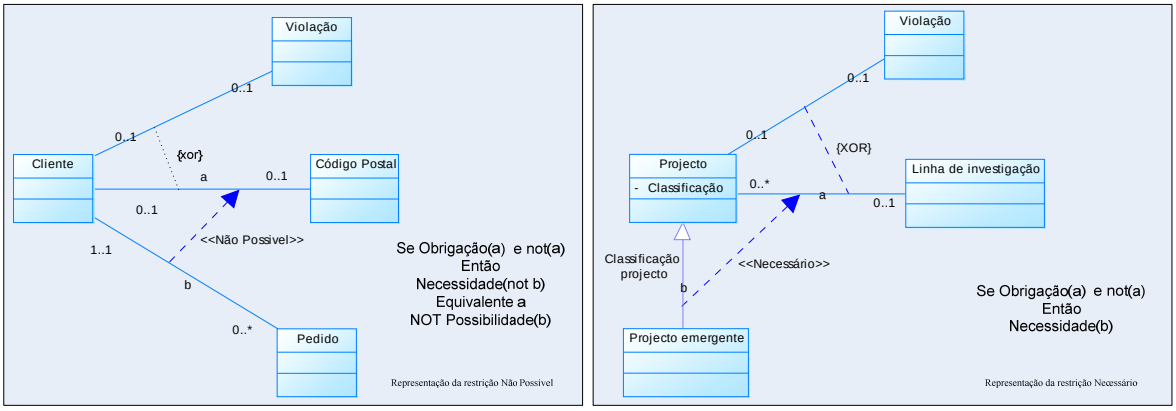


Figura 3.1. Representação gráfica das restrições Necessário e Não possível.

Nota. Figura adaptada de Ramos (2003).

Na figura 3.1 estão representadas as restrições "Contrary to duties" propostas para este estudo, considerando as associações de dados "a" e "b, o estereótipo "Necessário" implementa o conceito de Necessidade (b), o que significa que "b" é obrigatório, sem excepções; por sua vez o estereótipo "Não Possível" implementa o conceito de Necessidade (not b), cujo significado é a proibição, sem excepções de "b". Note-se que Necessidade (not b) equivale a not Possibilidade (b). A formação das restrições "Contrary to duties e o conceito deôntico Necessidade são explicados em detalhe no capítulo dois.

O problema que se coloca à utilização de restrições "Contrary to duties" é que a os novos elementos gráficos, que designam o tipo de restrições deônticas utilizadas, não implementam a semântica que deles é pretendida, ou seja o comportamento dos novos elementos gráficos não é conhecido pelo diagrama de classes da UML. Para adicionar significado semântico aos novos elementos gráficos não basta desenhá-los, é necessária uma interacção, essa interacção permite definir o comportamento esperado dos elementos no contexto das suas relações. Para o efeito, este estudo propõe a utilização de uma linguagem de modelação, a OCL, que permite, justamente, interagir com os elementos de qualquer um dos modelos da UML, nesse processo é possível adicionar significado semântico aos novos elementos gráficos.

Em síntese, nos casos de utilização real de sistemas de informação em que se justifique a utilização de restrições "Contrary to duties" o presente estudo propõe três etapas para a sua representação no diagrama de classes: definir as classes de objectos e associações entre as mesmas; criar novos elementos gráficos, com recurso aos mecanismos de extensão da UML, conforme proposto por Ramos (2003); e adicionar código, em OCL, capaz de implementar o comportamento esperado de cada uma das restrições impostas. A figura 3.2 ilustra o processo.

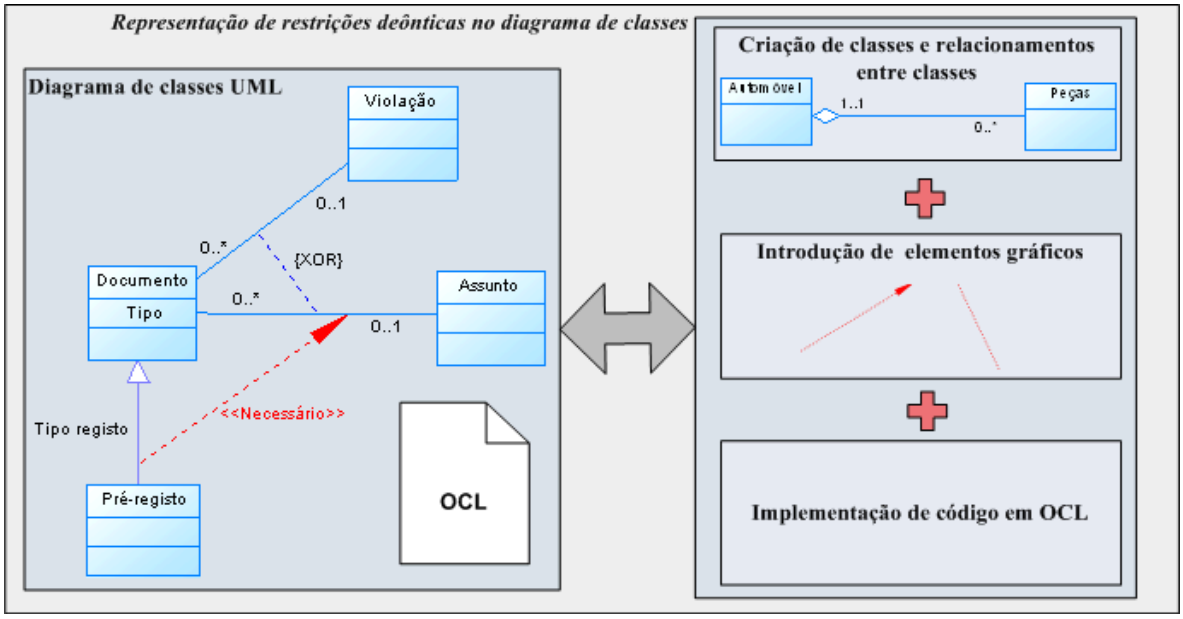


Figura 3.2. Etapas da formação de restrições "Contrary to duties", no diagrama de classes.

No exemplo da figura 3.2, as etapas de criação do diagrama de classes são ilustradas no lado direito da figura, do lado esquerdo encontra-se um exemplo de representação gráfica de uma restrição “Contrary to duties”.

Resolvida a fase de criação do diagrama de classes, é necessário guardar os objectos que resultem da utilização do sistema de informação, por outras palavras é necessário definir um método que garanta a persistência da informação. Para o efeito existem duas hipóteses, ou o modelo orientado ao objecto, que garante a salvaguarda directa de dados; ou o modelo relacional, cuja salvaguarda é garantida por tabelas relacionais. Como o modelo relacional apresenta maior proliferação, maior sucesso comercial, e maiores capacidades na gestão da informação, será utilizado no decorrer deste estudo, em detrimento do modelo orientado ao objecto.

A utilização do modelo relacional levanta, no entanto, uma nova questão: Como fazer a passagem do modelo de classes para o modelo relacional? Existem regras de mapeamento entre os dois modelos, mas essa regras não se aplicam a novos elementos gráficos, nem a código escrito em OCL. Para resolver esta questão, o presente estudo propõe a geração automática do modelo relacional, em três fases distintas:

1) Aplicação das regras de mapeamento entre o modelo de classes e o modelo relacional, conforme descrito em Ramos (2006), e explicado no decorrer do capítulo dois.

2) Criação de tabelas adicionais, com dados, relativos aos elementos gráficos que caracterizam os dois tipos de restrição “Contrary to duties”.

3) Criação de triggers específicos, para garantir o comportamento esperado de cada uma das restrições.

Para que seja possível a geração do modelo relacional é necessário recolher dados do modelo de classes, aplicar regras de mapeamento, e criar, através de código, o modelo relacional correspondente, tudo isto de forma automática e transparente. A geração automática do modelo relacional e cada uma das fases referidas serão explicadas em detalhe no decorrer do presente capítulo.

Para implementar as três fases que conduzem à geração do modelo relacional, o presente estudo propõe a utilização da linguagem de programação Prolog, a linguagem Prolog será utilizada para receber todos os dados relevantes do diagrama de classes e transformá-los, através de regras bem definidas, em informação que permita a criação do modelo relacional. O modelo relacional deve implementar o comportamento esperado das restrições "Contrary to duties", para isso o código em Prolog recebe dados do diagrama de classes e processa a informação, conjugando-a com as regras de mapeamento, com a estrutura das tabelas adicionais, e com a definição dos triggers.

3.2.1 Limitações do diagrama de classes

O diagrama de classes tem como objectivo representar as entidades do mundo real, num esquema lógico, perceptível, e intuitivo. Em correspondência, com as relações entre entidades, do mundo real, o diagrama de classes relaciona as classes de objectos a que correspondem cada uma delas Desta forma pretende-se que a realidade seja retratada de forma fiel pelos sistemas de informação.

No entanto, por vezes a utilização em real de sistemas de informação determina algumas excepções, em que o tipo de restrições fortes, ou seja dependentes de um valor booleano "sim" ou "não", não chega para representar toda a extensão da realidade. Nesses casos, a situação determina que a realidade seja expressa com recurso a uma norma, ou seja que a realidade seja substituída por uma idealidade, que pode ser violada. Ao ser violada, essa idealidade dá origem a novas restrições, que implementam na prática o conceito de restrição "Contrary to duties". A questão que se levanta é que o diagrama de classes não implementa esse tipo de restrições, por isso não existem símbolos gráficos para as suportar.

Através de mecanismos de extensão é possível adicionar novos elementos gráficos aos modelos da UML. Assim, uma das soluções possíveis passa pelo recurso aos mecanismos de extensão da UML, e à criação de novos símbolos que permitam representar graficamente as restrições "Contrary to duties". Resta resolver o problema da semântica, ou seja do comportamento esperado de cada tipo de restrição. Para isso não bastam os

mecanismos de extensão da UML. É necessário interagir com os elementos que formam as restrições “Contrary to duties” e nesse processo adicionar-lhes semântica.

Como a OCL é uma linguagem de modelação, faz parte da UML, e permite condicionar o comportamento dos elementos, o presente estudo propõe a sua utilização para enriquecer o diagrama de classes com o significado semântico das restrições “Contrary to duties” propostas.

3.2.2 Proposta para representação genérica

A OCL é uma linguagem de modelação, faz parte da UML, e é baseada no rigor e na lógica matemática. A função da OCL é interagir com os elementos gráficos de cada um dos modelos da UML, e nesse processo, condicionar o seu comportamento de forma a enriquecer a representação da realidade.

No âmbito do presente estudo, à OCL é reservado o papel de representar, no diagrama de classes, o comportamento genérico de dois tipos de restrições “contrary to duties”. No capítulo quatro, a arquitectura que suporta o comportamento genérico dos dois tipos de restrições, incluindo o código genérico que irá ser proposto no decorrer do presente capítulo, será aplicado a dois casos reais de utilização de sistemas de informação.

Na realidade, as restrições referidas têm em comum a dependência de uma obrigação, que idealmente devia ser cumprida. Essa obrigação corresponde a uma restrição deôntica, que implementa um conceito de idealidade. O conceito de idealidade permite representar situações excepcionais, em que determinada associação de dados, que devia ser obrigatória, por vezes é violada. Assim, a representação gráfica e a semântica desse conceito de idealidade deve incluir, não só a associação que idealmente devia ser cumprida, mas também uma outra associação que permita o registo das suas violações. A figura 3.3 ilustra o conceito de obrigação.

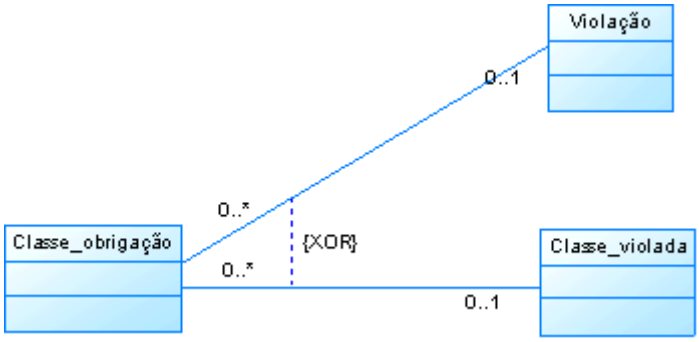


Figura 3.3. Representação gráfica da restrição deôntica "Obrigação".

Analisando a figura 3.3 verifica-se que, entre as duas associações que unem as três classes: "Classe_obrigação", "Classe_violada", e "Violação," existe uma linha a tracejado e um estereótipo: "XOR". O estereótipo "XOR" faz parte da UML, e serve, precisamente, para impor relações de exclusividade. No exemplo da figura 3.3, o operador "XOR" significa que, se uma das associações se verificar, então, a outra não se verifica. Não é, também, possível que as duas se verifiquem.

Então, nesse caso, a representação gráfica e o comportamento esperado da obrigação inicial estão representados na figura 3.3, sem recurso a OCL. Ou seja, se a associação entre as classes "Classe_obrigação" e "Classe_violada", que representa a obrigação, não ocorrer, isso significa que ocorre a associação entre "Classe_obrigação" e "Violação", que representa a violação dessa mesma obrigação. Falta analisar as restrições que emergem, decorrentes da violação da obrigação inicial.

Assim, no caso da restrição "Não Possível", representada na figura 3.4, o comportamento esperado da associação entre "Classe_obrigação" e "Classe_possibilidade" resulta na sua proibição, sempre que existir violação da obrigação entre "Classe_obrigação" e "Classe_violada".

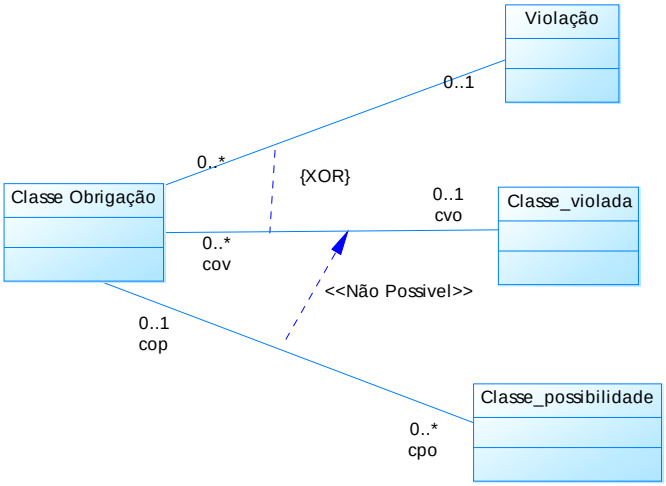


Figura 3.4. Representação gráfica da restrição "Não Possível".

Graficamente, a pretensão de proibir a associação entre "Classe_obrigação" e "Classe_possibilidade" até que a obrigação entre "Classe_obrigação" e "Classe_violada" se verifique está cumprida, com recurso a dois novos elementos: a seta a tracejado e um novo estereótipo, com o nome da restrição "Não possível". A questão que emerge é que os dois elementos são novos para a UML, e portanto desprovidos de qualquer significado semântico. Neste caso torna-se imprescindível a utilização da linguagem OCL, para implementar o comportamento pretendido. O ficheiro de texto da figura 3.5 contém o código genérico, em OCL, proposto para implementar a semântica da restrição referida.

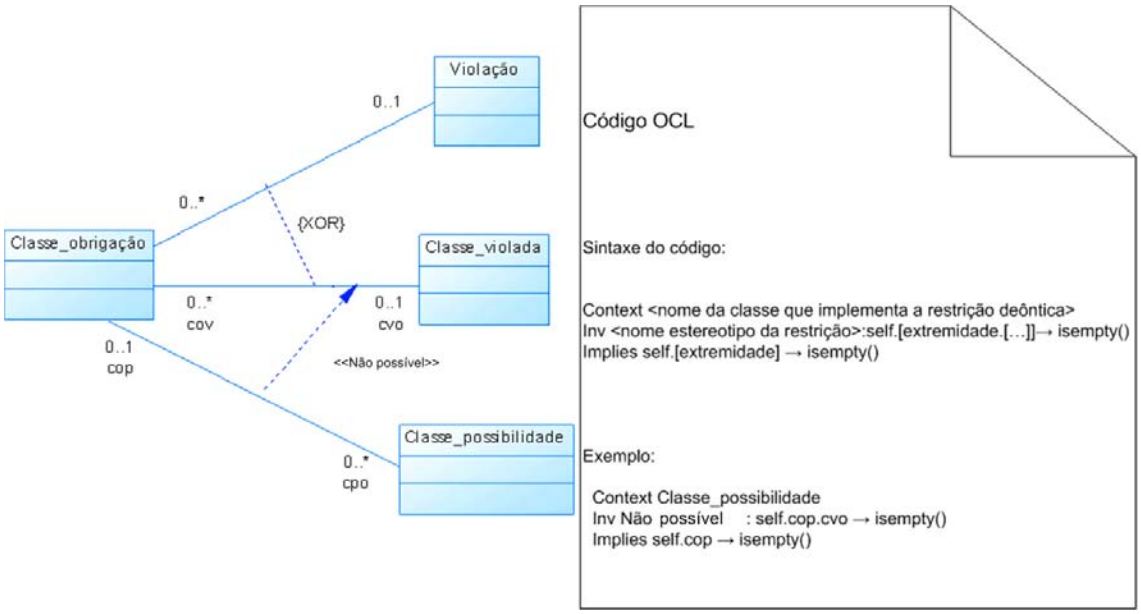


Figura 3.5. Proposta genérica da estrutura de código, em OCL que adiciona a semântica ao tipo de restrição "Não Possível".

Na figura 3.5 a primeira linha de código do ficheiro de texto define o elemento inicial de referência, ou seja o contexto das expressões de OCL, as linhas seguintes implementam a semântica da restrição "Contrary to duties".

Para implementar a semântica pretendida as expressões, em OCL, navegam do elemento de referência "Classe_possibilidade", para a associação de obrigação. Note-se que para proceder à navegação até à associação de obrigação são utilizados os nomes das extremidades de cada associação, neste caso "cop" e "cvo". O posicionamento no elemento pretendido, "cvo", e a expressão "isempty()" permitem questionar, para cada instância do diagrama, se a associação entre as mesmas classes se verifica. Se a associação se verificar, nada mais é feito. No entanto, se não se verificar, as expressões navegam para a associação que se pretende proibir. Por fim, a expressão "Implies", condiciona a associação referenciada, obrigando-a a não se verificar até que outra associação, a de obrigação, se verifique. Note-se que as expressões da OCL não avaliam as classes de objectos, mas sim, cada instância dos objectos dessas classes.

Falta, agora, caracterizar a restrição "Necessário"; as duas restrições, "Necessário" e "Não possível", não diferem na estrutura que caracteriza a restrição deôntica inicial, a representação diagramática e a semântica associada a essa restrição é precisamente a mesma, o que difere é a nova restrição, que surge decorrente da violação da anterior.

Assim, a nova restrição “Necessário” difere de “Não possível” no sentido em que torna obrigatória uma determinada associação de dados, decorrente da violação da obrigação inicial.

A figura 3.6 representa a proposta genérica de código, em OCL, que adiciona o comportamento esperado do tipo de restrição “Necessário”.

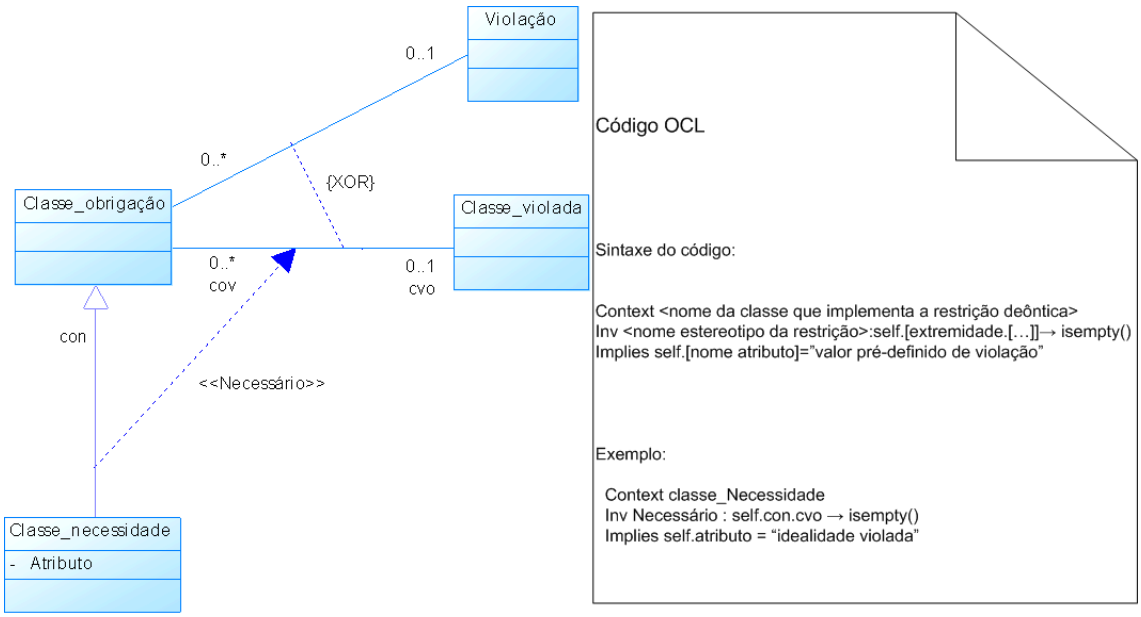


Figura 3.6. Proposta genérica da estrutura de código, em OCL, que adiciona a semântica ao tipo de restrição “Necessário”.

Na figura 3.6, o elemento de referência “Classe_necessidade” é o ponto de partida das expressões de OCL. A navegação para a associação que implementa a obrigação inicial permite questionar se a instância actual da associação se verificou. Note-se que até aqui o procedimento e as expressões de OCL que foram implementadas na restrição “Não possível” são iguais para “Necessário”. No entanto, as acções subsequentes diferem: na restrição “Não possível” uma associação de dados é proibida, em consequência da violação de uma obrigação inicialmente imposta; em “Necessário” é imposta a obrigatoriedade de uma associação de dados, em consequência da violação de uma obrigação.

A associação de dados imposta em “Necessidade” é suportada por atribuição de um valor fixo ao atributo de referência da classe “Classe_necessidade”, que se ilustra na figura 3.6. No caso de não haver violação da obrigação inicial a segunda associação deixa de ser obrigatória, e o atributo assume o valor pré-definido da classe.

Assim, com recurso à implementação de OCL, no diagrama de classes, é possível representar, na íntegra, as restrições “Não possível” e “Necessário”. O presente estudo pretende adaptar o código genérico proposto para os dois tipos de restrição aos casos reais descritos no capítulo quatro, de forma a representar, na prática, a estrutura e o comportamento esperados.

3.3 Geração do modelo relacional a partir do diagrama de classes

Neste estudo, o modelo utilizado para garantir a persistência da informação corresponde ao modelo relacional. Por esse motivo, torna-se necessário traduzir a informação fornecida pelo diagrama de classes, e fazer o mapeamento para o seu correspondente, no modelo relacional.

Com a aplicação de um conjunto genérico de regras é possível mapear a informação entre os modelos de classes e relacional, e ao mesmo tempo garantir a coerência entre a modelação do sistema e o modelo de dados que lhe dá suporte. A representação de restrições “Contrary to duties” no diagrama de classes representa um caso especial de modelação, isto porque depende da criação de um novo símbolo gráfico, representativo do tipo de restrição deôntica utilizada; e da implementação de código específico, em OCL, que implementa o comportamento esperado da mesma. A questão que se coloca é que as regras de mapeamento entre os dois modelos não se aplicam a novos elementos gráficos, nem a código, escrito em OCL. Para traduzir e mapear restrições deônticas é necessário traduzir o elemento que as representa e adicionar-lhe o comportamento que delas se espera, assim o presente estudo propõe a utilização de tabelas adicionais, conforme sugerido por Ramos (2003), e a criação de uma estrutura genérica de triggers, cujo código se encarrega de mapear o comportamento esperado das restrições referidas para o contexto do diagrama relacional que lhes pertence.

3.3.1 Proposta para tabelas adicionais

A aplicação de regras de mapeamento entre os diagramas que implementam os modelos de classes e relacional, permite traduzir a maior parte dos elementos de modelação. No entanto, o mapeamento de restrições "Contrary to duties" não é possível, recorrendo, em exclusivo, às mesmas regras. A figura 3.7 demonstra o resultado da aplicação das regras de mapeamento, para os dois tipos de restrição, anteriormente descritos.

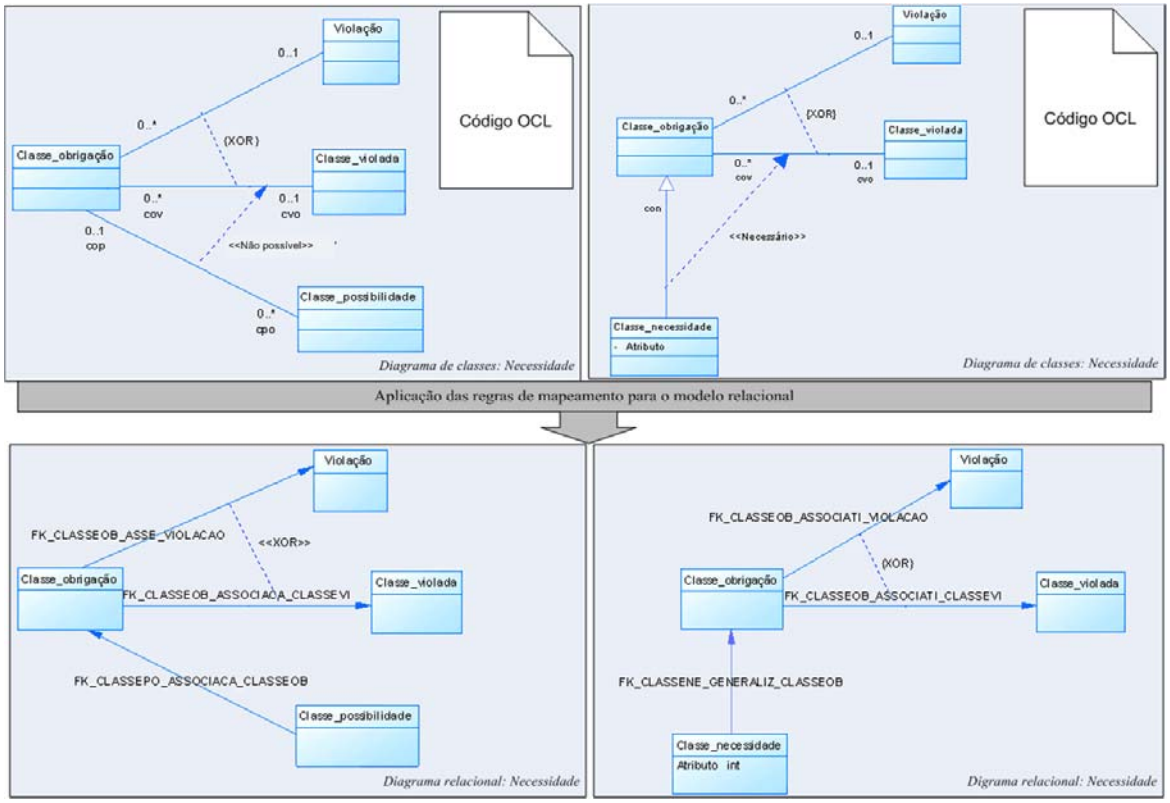


Figura 3.7. Aplicação genérica de regras de mapeamento para o modelo relacional.

Na figura 3.7, os estereótipos "Não possível" e "Necessário", não são traduzidos com a aplicação das regras de mapeamento. Efectivamente, os diagramas relacionais da figura 3.7 não têm informação da existência de restrições "Contrary to duties". Por esse motivo torna-se necessário enviar essa informação para o modelo relacional.

As restrições "Contrary to duties" são formadas por duas restrições deônticas. A primeira corresponde a uma obrigação, a segunda surge em consequência da violação da primeira, e prevalece, enquanto a obrigação inicial não for satisfeita.

No diagrama de classes, as restrições deônticas são referenciadas por um novo elemento gráfico, representativo da restrição “Contrary to duties”: corresponde a uma seta com um estereótipo, que designa o nome da restrição. O posicionamento desse elemento não é desprovido de intenção, efectivamente a direcção da seta indica a associação que implementa o conceito inicial de idealidade, ou seja a obrigação inicial. Na outra extremidade da seta é referenciada a associação que implementa a segunda restrição, consequência da violação da primeira. Como as associações referidas associam classes de objectos, torna-se simples determinar o nome das classes envolvidas na formação de restrições “Contrary to duties”.

As associações e as classes correspondem, no diagrama relacional, a tabelas e ao relacionamento entre tabelas, portanto se o diagrama relacional for informado das associações e das classes que formam as restrições “Contrary to duties” pode fazer corresponder essa informação aos elementos de modelação. Como o modelo relacional também garante a persistência da informação torna-se possível estruturar e guardar essa informação em tabelas de dados.

No processo de estruturação da informação, é necessário perceber a função dos modelos de classes, e relacional, o primeiro visa a modelação e descrição do sistema; o segundo corresponde ao modelo de dados que suporta esse mesmo sistema. O modelo de dados que suporta o sistema tem como objectivo ser ágil e consistente, de forma a responder rápida e eficientemente às solicitações, assim, tendo em conta os objectivos do modelo relacional verifica-se que existem melhorias a fazer.

Efectivamente, a caracterização, no modelo relacional, das restrições “Contrary to duties” depende da restrição deôntica inicial. Na origem da formação dessa restrição está incluída uma classe com o nome “Violação”, a que vai corresponder, com a aplicação das regras de mapeamento, uma tabela com o mesmo nome. A questão que se levanta é que por cada restrição “Contrary to duties”, traduzida para o modelo relacional, é criada uma tabela “violação”. Essa situação é desnecessária e pode gerar alguma inconsistência uma vez que podem ser criadas várias tabelas com o mesmo nome, que servem o mesmo propósito: registar violações da restrição deôntica inicial. Pelo exposto basta que seja criada uma tabela “violação” para guardar toda a informação.

O presente estudo propõe a criação de duas tabelas adicionais: “Idealidade” e “Violação”, conforme sugerido por Ramos (2003), com o objectivo de guardar toda a informação respeitante à caracterização e manutenção das restrições “Contrary to duties”.

A classe “Violação” não é traduzida segundo as regras genéricas de mapeamento, mas sim com o recurso à criação de uma tabela adicional com o nome “Violação”.

Resta estruturar a informação das tabelas adicionais, de forma a registar toda a informação relevante. A tabela “Idealidade” vai concentrar a informação que caracteriza as restrições “Contrary to duties”, por seu lado a tabela “Violação” será utilizada quando existir necessidade de registar as violações à restrição deôntica inicial.

Começando pela tabela “Idealidade”, para cada restrição “Contrary to duties” é necessário guardar a seguinte informação:

1) Nome da associação que implementa o conceito de idealidade, correspondente à obrigação inicial; nome das classes que fazem parte dessa associação; e indicação da classe que implementa a chave estrangeira da associação, no modelo relacional.

2) Nome da associação que implementa a restrição de necessidade ou possibilidade que surge quando a obrigação inicial não é cumprida; nome das classes que fazem parte dessa associação; e indicação da classe que implementa a chave estrangeira da associação, no modelo relacional.

Para além da informação referida, é necessário identificar o tipo de restrição que cada associação implementa, e relacionar as duas associações que dão origem à restrição “Contrary to duties”. A tabela 3.1 ilustra a estrutura proposta para a tabela idealidade.

Campo	***Descrição***	***Correspondência***
Deontica_id	Identificador único de registo.	Não tem correspondente no diagrama de classes
Tabela_um	Nome da classe esquerda da associação.	Classe
Tabela_dois	Nome da classe direita da associação.	Classe
Tabela_p	Nome da tabela que implementa a chave estrangeira.	Classe
Relação_e	Nome da associação.	Associação
Idealidade	Tipo de restrição implementada pela associação. “i” para idealidade;”f” para “Não possível; e “o” para “Necessário”.	Estereótipo
Relação_d	Relaciona duas associações, para que seja formada a restrição Contrary to duties.	Corresponde ao relacionamento das duas associações que dão origem a uma restrição “Contrary to duties”

Tabela 3.1. Estrutura genérica, proposta para a tabela Idealidade.

Na Tabela 3.1 a estrutura da tabela “Idealidade” define o repositório de toda a informação a utilizar pelo modelo relacional. Essa informação permite mapear, para o modelo relacional, as restrições “contrary to duties” existentes no diagrama de classes. O campo “Relação_d” desempenha um papel importante na tabela “Idealidade”. A sua função é relacionar os dois registos da tabela “idealidade”, que em conjunto definem a restrição “contrary to duties”. A Figura 3.8 exemplifica a utilização da tabela “Idealidade”, e ilustra a correspondência da informação que é extraída do modelo de classes e guardada na tabela “Idealidade”.

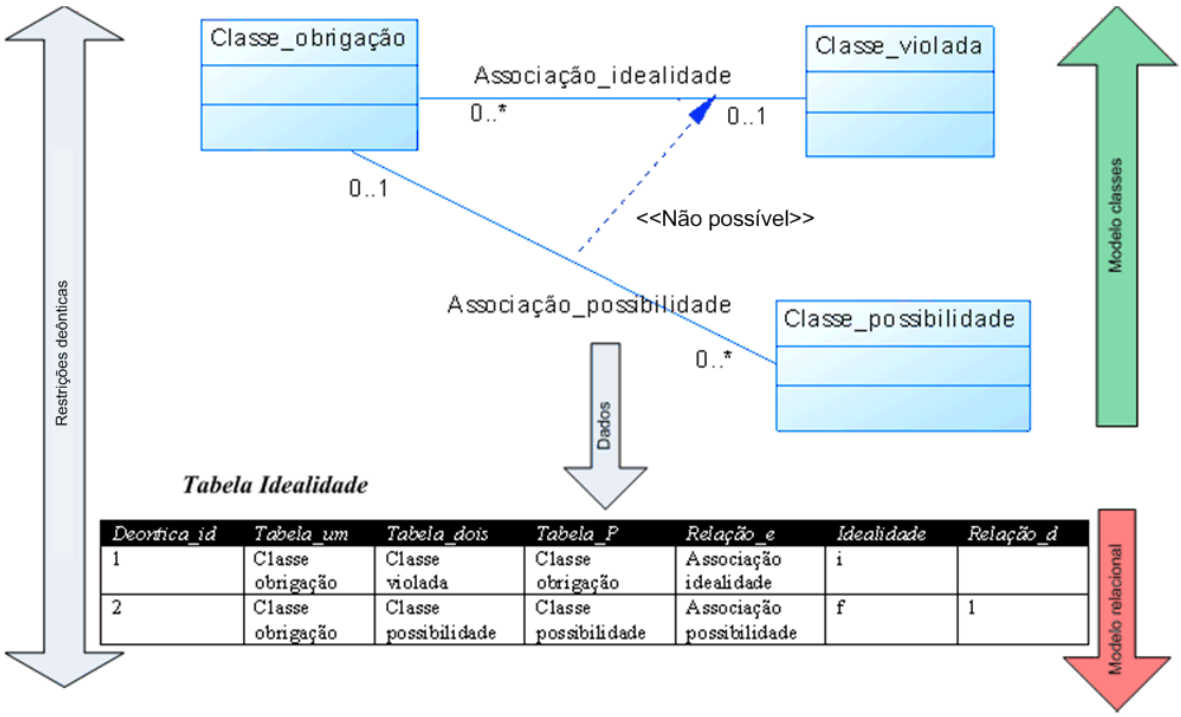


Tabela Idealidade

Deontica_id	*Tabela_um*	*Tabela_dois*	*Tabela_P*	*Relação_e*	*Idealidade*	*Relação_d*
1	Classe obrigação	Classe violada	Classe obrigação	Associação idealidade	i	
2	Classe obrigação	Classe possibilidade	Classe possibilidade	Associação possibilidade	f	1

Figura 3.8. Representação genérica de restrições deônticas, no modelo relacional.

Analisando a figura 3.8 verifica-se que os dados da tabela “Idealidade” caracterizam, no modelo relacional, a restrição “Não possível”; para a restrição “Necessário” o processo seria o mesmo. Note-se que o campo Relação_d relaciona de forma inequívoca as duas associações de que as restrições “contrary to duties“ dependem. Falta definir a estrutura da tabela “Violação”. Efectivamente, foi tomada opção de não fazer a sua tradução com recurso às regras de mapeamento entre os dois modelos. Essa opção visa melhorar a consistência do modelo relacional, uma vez que é desnecessária a criação de uma tabela “Violação” por cada restrição “contrary to duties”. Assim, a tabela violação deve registar as violações da associação de idealidade. Para o efeito torna-se necessário o

relacionamento com a tabela “Idealidade”, no sentido de identificar quais as associações violadas. A figura 3.9 ilustra a estrutura e o relacionamento propostos.

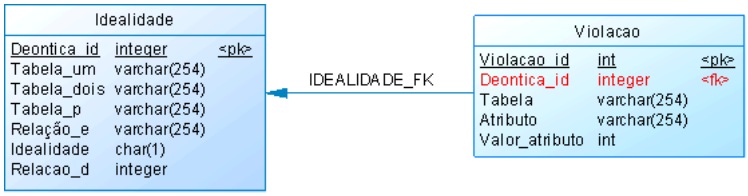


Figura 3.9. Representação das tabelas complementares.

Na figura 3.9, as tabelas “Idealidade” e “Violação” relacionam-se através da chave estrangeira “Deontica_id”. Desta forma é possível referenciar inequivocamente a restrição que foi violada. Os campos “Tabela”, “Atributo”, e “Valor_atributo” da tabela “Violação” identificam a tabela e a chave primária que implementa a associação de idealidade que foi quebrada.

O presente estudo propõe a estrutura descrita na figura 3.9 para a representação das restrições “contrary to duties” no diagrama relacional.

3.3.2 Proposta para arquitectura de triggers

A aplicação de regras de mapeamento entre os diagramas que implementam os modelos de classes e relacional, permite traduzir a maior parte dos elementos de modelação. A criação de tabelas adicionais, complementa as regras de mapeamento, e torna possível a caracterização das restrições ”contrary to duties” no modelo relacional. Resta analisar o comportamento, ou seja a semântica destas restrições no contexto do modelo relacional. No modelo de classes a semântica das restrições “contrary to duties” é implementada com recurso a código, escrito em OCL. A questão que emerge é que a OCL não tem tradução para o modelo relacional.

Para manipular e gerir informação, o modelo relacional oferece, como recurso, a escrita de código procedimental, esse código pode ser executado de diversas formas, uma delas corresponde a um objecto chamado “trigger”, que mantém no seu interior um conjunto pré-definido de código, com a ocorrência de determinado evento ou conjunto de eventos, o código do trigger é automaticamente executado.

Como a OCL, descreve, no modelo de classes, a semântica de cada uma das restrições "Contrary to duties", o presente estudo propõe o mapeamento da semântica implementada pelo código OCL, para código procedimental, suportado pelo modelo relacional. O mapeamento tem como objectivo garantir a implementação da semântica de cada uma das restrições, no modelo relacional. O código procedimental a implementar no modelo relacional será descrito em triggers.

3.3.2.1 Arquitectura da restrição "Não Possível"

Para descrever código procedimental, é necessário proceder à representação das restrições "Contrary to duties" no diagrama relacional. Para o efeito torna-se necessário proceder à aplicação das regras de mapeamento entre os dois modelos, e à criação da estrutura de tabelas adicionais. A figura 3.11 ilustra o processo para a restrição "Não possível".

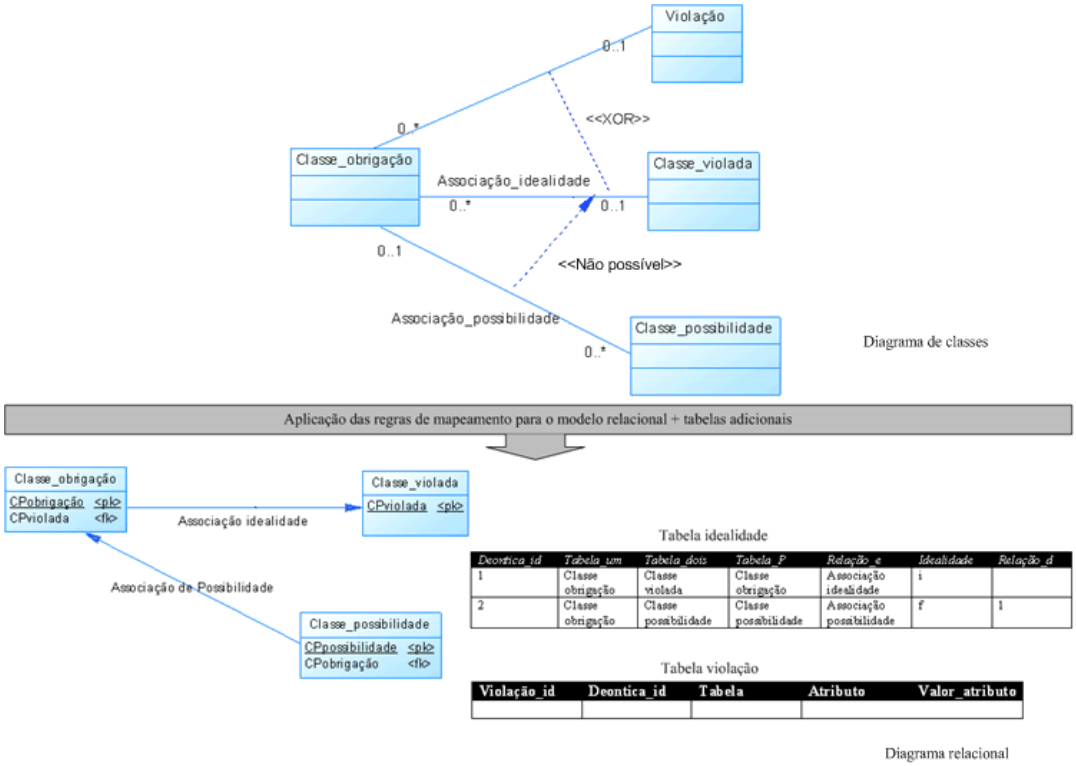


Figura 3.10. Aplicação genérica das regras de mapeamento e criação de tabelas complementares para a restrição "Não possível".

Com a aplicação das regras de mapeamento para o modelo relacional, são criadas tabelas e relações, correspondentes aos objectos do modelo de classes.

As tabelas do modelo relacional herdam os atributos das classes que as implementam, um desses atributos corresponde à chave primária, representada na figura 3.10 pela sigla "pk". A existência de uma chave primária justifica-se por ser o identificador unívoco dos registos de uma tabela. Outro atributo que resulta da aplicação das regras de mapeamento diz respeito ao tipo de associação, conforme o tipo de associação são criadas chaves estrangeiras, correspondentes a chaves primárias de outras tabelas. A correspondência entre os dois tipos de chave relaciona a informação.

No caso descrito pela figura 3.10, as chaves estrangeiras, definidas pela sigla "fk" resultam de duas associações do tipo "um para muitos", que com a aplicação da regra herdam a chave primária do lado "um" como chave estrangeira do lado "muitos". Note-se que na figura 3.10 o diagrama de classes não especifica atributos, isto acontece por se tratar de uma caracterização genérica da restrição "Não possível", no entanto o diagrama relacional reflecte as chaves primárias e estrangeiras a que correspondem, no modelo de classes, os objectos e seus relacionamentos.

No contexto do modelo relacional, após aplicação das regras de mapeamento e da geração de tabelas complementares, torna-se necessário implementar e descrever os triggers que vão suportar o comportamento semântico das restrições "Contrary do duties", para o efeito a linguagem relacional SQL desempenha um papel preponderante. A linguagem SQL é utilizada, pela grande maioria das bases de dados relacionais, para manipulação e gestão de informação, contém uma linguagem própria de comandos para delimitar, inserir, actualizar e apagar informação. Três desses comandos têm particular interesse no decorrer deste estudo, uma vez que têm a capacidade de alterar a informação. Efectivamente, os comandos "Insert", "Update", e "Delete" implementam a capacidade de inserir, alterar, e apagar dados. Os triggers correspondem a objectos do SQL, são implementados em tabelas, e são despoletados por eventos. Esses eventos correspondem, precisamente, à ocorrência de qualquer dos comandos anteriormente referidos, o que significa que sempre que a informação sofre alterações, dentro de uma tabela, o trigger a que corresponde qualquer um destes eventos é despoletado.

O interior do trigger contém código procedimental, que é executado sempre que o evento que o despoleta ocorre, ou seja sempre que é evocado qualquer um dos comandos "Insert", "Update" ou "Delete" sobre a tabela. Normalmente, o código do trigger é

executado antes da transacção do comando que o evoca, no entanto alguns fabricantes de bases de dados relacionais permitem, em opção, que o código seja executado depois da transacção. Para efeitos deste estudo é considerada a primeira opção, ou seja o código é executado antes da transacção do comando.

Genericamente, no caso da restrição “Não possível”, sempre que a obrigação inicial correspondente à restrição de idealidade é violada, torna-se necessário que a relação de dependência, que surge, deixe de ser permitida. No modelo relacional, a violação da obrigação inicial corresponde à existência de um valor nulo, no atributo que corresponde à chave estrangeira da relação de idealidade. Sempre que a informação da tabela que implementam a restrição de idealidade for alterada, é necessário questionar o modelo de dados. A questão a colocar é saber se o valor da chave estrangeira que implementa a relação de idealidade é nulo. Se for, então isso significa que a obrigação inicial foi violada. Como o código dentro dos triggers é executado antes da transacção de qualquer um dos comandos que altera a informação, então, o mesmo código pode questionar o valor da chave estrangeira. No caso de esse valor ser nulo, o trigger pode proceder ao registo da violação, na tabela “Violação”. Desta forma o modelo relacional fica com o conhecimento dos registos violados.

Ao ter conhecimento de uma violação, o sistema deve proibir a relação a que corresponde a restrição “Não possível”. Isto é conseguido com a implementação de mais triggers, desta vez na tabela que implementa a restrição “Não possível”. Os triggers da tabela que implementa a restrição “Não possível” questionam a tabela “Violação” e verificam se existe violação da relação de idealidade para o registo corrente. Se essa violação existir, o código dos triggers impossibilita a transacção do comando, tornando a relação proibida. Desta forma é garantida a semântica integral da restrição “Não possível” no contexto do diagrama relacional. Na figura 3.11 é ilustrada a solução proposta.

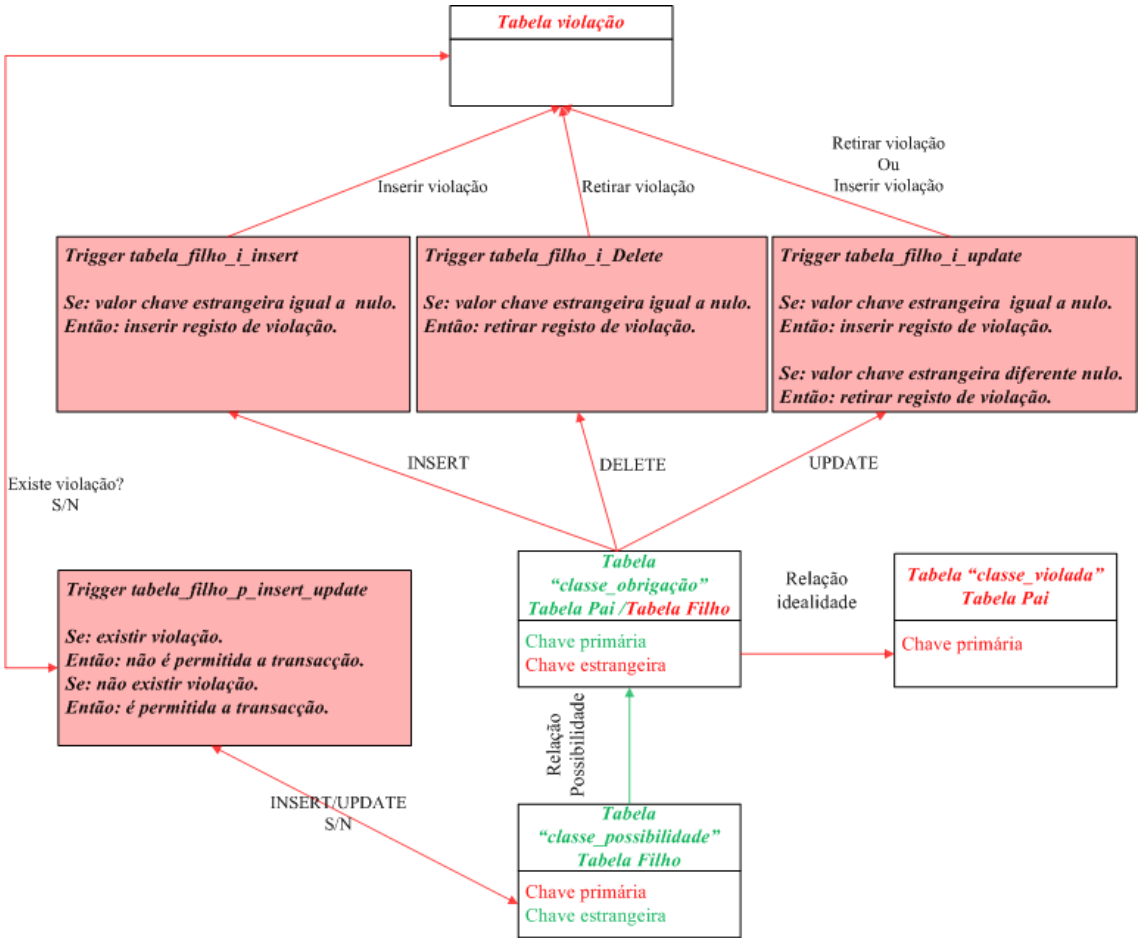


Figura 3.11. Proposta da arquitectura genérica dos triggers que implementam a semântica da restrição "Não possível".

A figura 3.11 descreve a arquitectura proposta, que implementa o comportamento genérico para o tipo de restrição "Não possível".

A relação entre as tabelas "classe_obrigação" e "classe_violada" corresponde à obrigação inicial. A tabela "classe_obrigação" implementa a chave estrangeira da relação de idealidade, e a tabela "classe_possibilidade" implementa a chave estrangeira da relação de possibilidade. Os triggers actualizam a tabela "violação" com informação dos registos que têm chave estrangeira nula. O trigger da tabela "classe_possibilidade" questiona a tabela "violação". Se existir violação para o registo corrente, a relação de possibilidade não se verifica, caso não exista então o trigger viabiliza a transacção.

3.3.2.2 Arquitectura da restrição "Necessário"

Analogamente à restrição "Não possível", a aplicação das regras de mapeamento, e a criação da estrutura das tabelas adicionais é essencial para descrever a restrição "Necessário". A figura 3.12 ilustra o processo de caracterização da restrição referida.

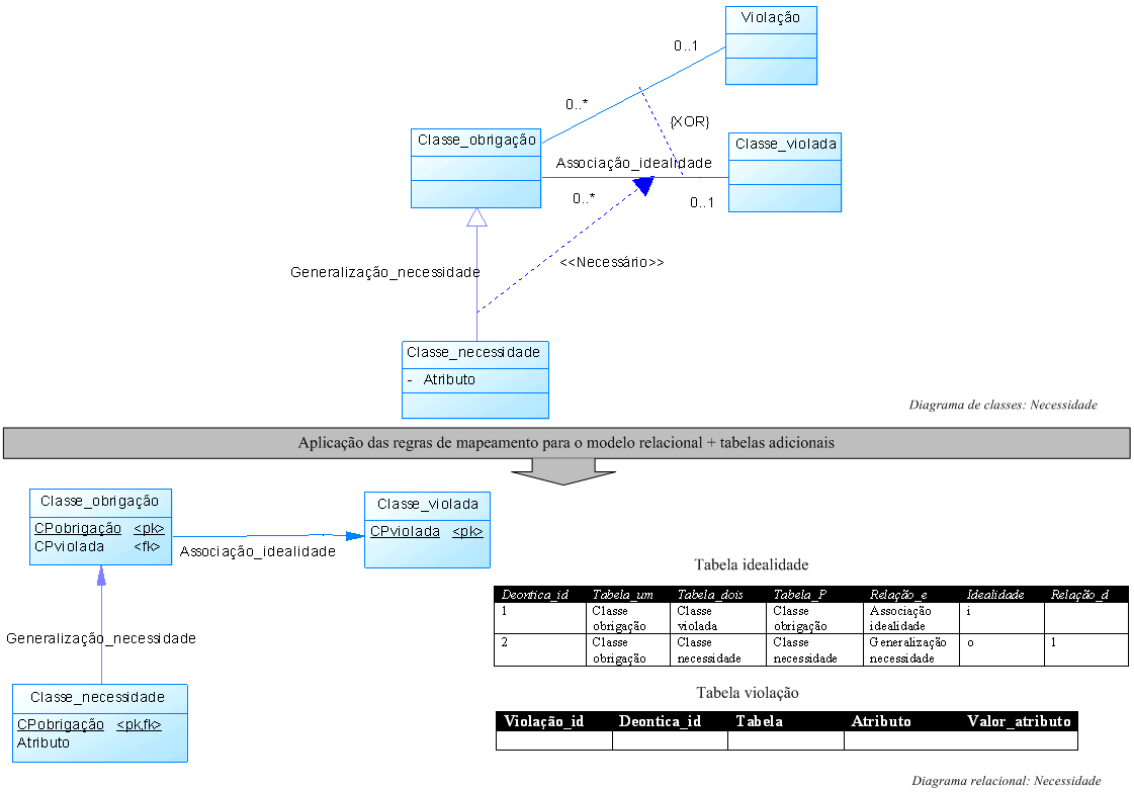


Figura 3.12. Aplicação genérica das regras de mapeamento e criação de tabelas complementares para a restrição "Necessário".

No caso da restrição "Necessário", a semelhança com a restrição "Não possível" diz respeito à forma como a mesma é despoletada, ou seja, a obrigação inicial entre duas das tabelas é violada, e em consequência dessa violação, surge uma outra restrição, desta vez "Necessário", que se traduz na obrigatoriedade de uma nova associação de dados. Como a violação da obrigação inicial corresponde, também, a um valor nulo no atributo que guarda a chave estrangeira, a forma de representação obrigação inicial é semelhante; no entanto, como a nova restrição implica o registo de um valor pré-definido, na tabela que

implementa a restrição “Necessário”, os triggers diferem, de forma a corresponderem ao código necessário para garantir a implementação da nova restrição.

A relação entre as tabelas que implementam a relação “Necessário” corresponde, no modelo de classes, a uma generalização. Essa generalização origina que os objectos da subclasse, que dá origem à tabela “filho”, não existam sem uma correspondência directa com os objectos da supraclasse, tabela “Pai”. O modelo relacional desta relação determina a correspondência entre as suas tabelas, tanto em número, como na relação entre os seus registos. Por esse motivo o trigger correspondente ao comando “Delete” é implementado na tabela “filho”. A sua função é apagar em primeiro lugar o registo da tabela “Pai”, depois verificar se existiu violação para esse registo.

A figura 3.13 descreve a arquitectura proposta, que implementa o comportamento genérico para o tipo de restrição “Necessário”.

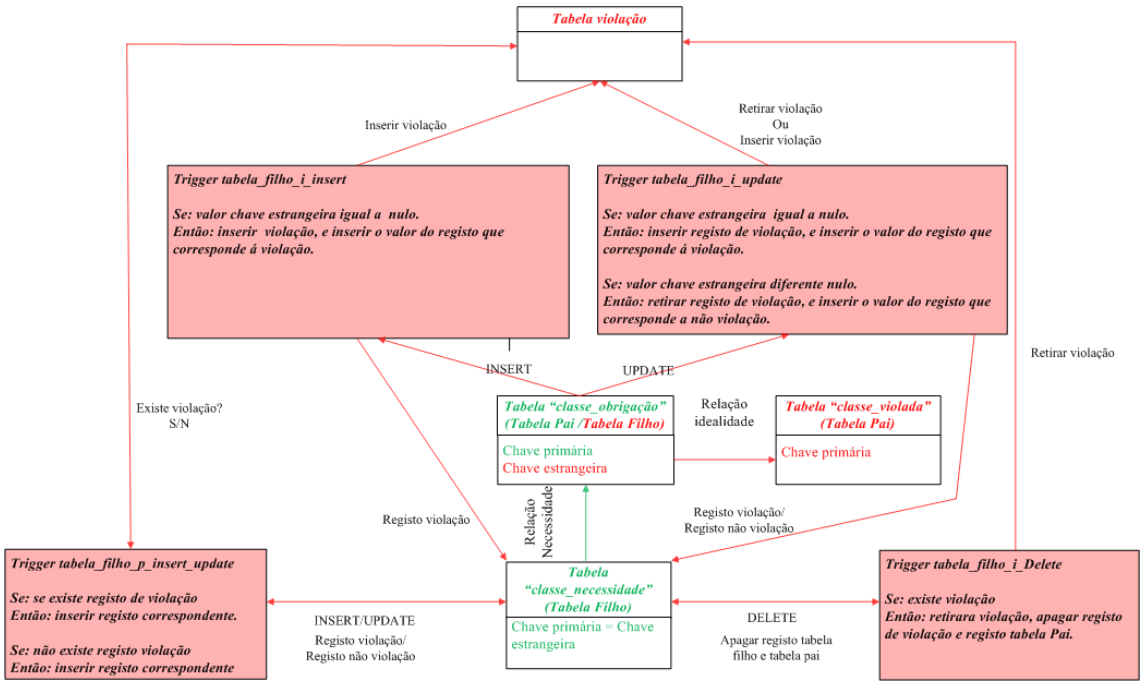


Figura 3.13. Proposta da arquitectura genérica dos triggers que implementam a semântica da restrição “Necessário”.

A relação entre as tabelas, “classe_obrigação” e “classe_violada” corresponde à obrigação inicial, a tabela “classe_obrigação” implementa a chave estrangeira dessa obrigação.

A tabela “classe_necessidade” implementa a relação de necessidade, que corresponde, ao mesmo tempo, à chave primária dessa mesma tabela. As setas com comandos SQL ligam as tabelas “filho” das relações de idealidade e necessidade aos respectivos triggers. Os triggers actualizam a tabela “violação” com informação dos registos com chave estrangeira nula. Ao mesmo tempo inserem os valores pré-definidos, na tabela “classe_necessidade”, consoante se trate de uma violação ou não.

3.4 Proposta de automatismo para geração do modelo relacional

Ao longo do presente capítulo foram propostas as soluções genéricas para a representação e mapeamento de restrições “contrary to duties”. A solução proposta inclui a representação, no diagrama de classes, das restrições “Necessário” e “Não possível”, e o mapeamento dessas restrições para o contexto do modelo relacional. Resta definir o processo que gera automaticamente o modelo relacional, a partir do modelo de classes. Para isso são propostas três etapas:

1) Aplicação automática das regras de mapeamento entre o modelo de classes e o modelo relacional.

2) Criação automática das tabelas adicionais, e caracterização de cada uma das restrições “contrary to duties”, através da inserção automática de dados.

3) Criação automática de triggers específicos, para garantir o comportamento esperado de cada uma das restrições.

O presente estudo propõe a utilização da linguagem de programação Prolog, para desenvolver uma aplicação preparada para ler a informação do diagrama de classes. Após leitura e processamento dos dados relativos a cada uma das fases referidas, o programa deve gerar o diagrama relacional correspondente. A Figura 3.14 ilustra o processo.

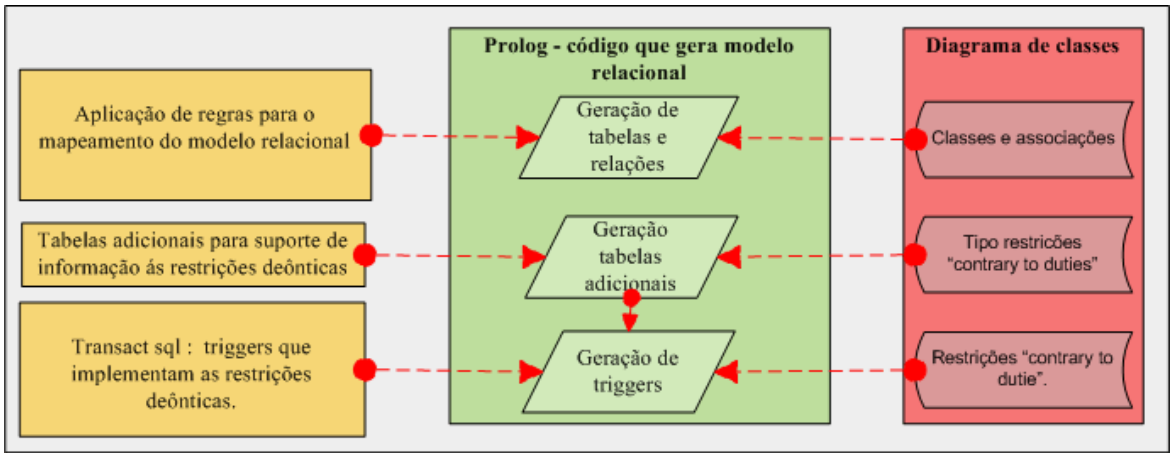


Figura 3.14. Proposta das etapas de formação do modelo relacional, com recurso à linguagem de programação Prolog.

Na figura 3.14, o código em Prolog implementa, no modelo relacional, as três etapas de geração do modelo relacional.

Na primeira etapa, geração de tabelas e relações, o Prolog lê toda a informação relevante, a partir do diagrama de classes:

1) Classes.
2) Atributos das classes.
3) Características de cada atributo.
4) Atributos identificadores únicos (vão corresponder à chave primária).
5) Associações entre classes.
6) Características de cada associação.

Seguidamente os dados são processados pelas regras de mapeamento, previamente introduzidas no programa de Prolog. O resultado é a criação das tabelas e relacionamentos correspondentes.

A segunda etapa corresponde à geração de tabelas adicionais, nesta fase o Prolog identifica os símbolos correspondentes a cada tipo de restrição “contrary to duties” e lê a informação que a elas diz respeito:

1) Tipo de restrição contrary to duties.
2) Caracterização das associações de idealidade, ou seja as obrigações iniciais

3) Caracterização das restrições, que surgem pela violação da obrigação inicial.

Após leitura dos dados, os mesmos são processados como registos da tabela adicional "Idealidade". Como resultado o programa cria duas tabelas adicionais, "Idealidade" e "Violação". Na tabela "Idealidade" são criados os registos que caracterizam os tipos de restrição "Contrary to duties".

Por fim, o programa de Prolog efectua o mapeamento do código em OCL, para o código em cada um dos triggers.

A figura 3.15 sintetiza o processo de implementação, proposto para representar as restrições "Contrary to duties", no diagrama relacional.

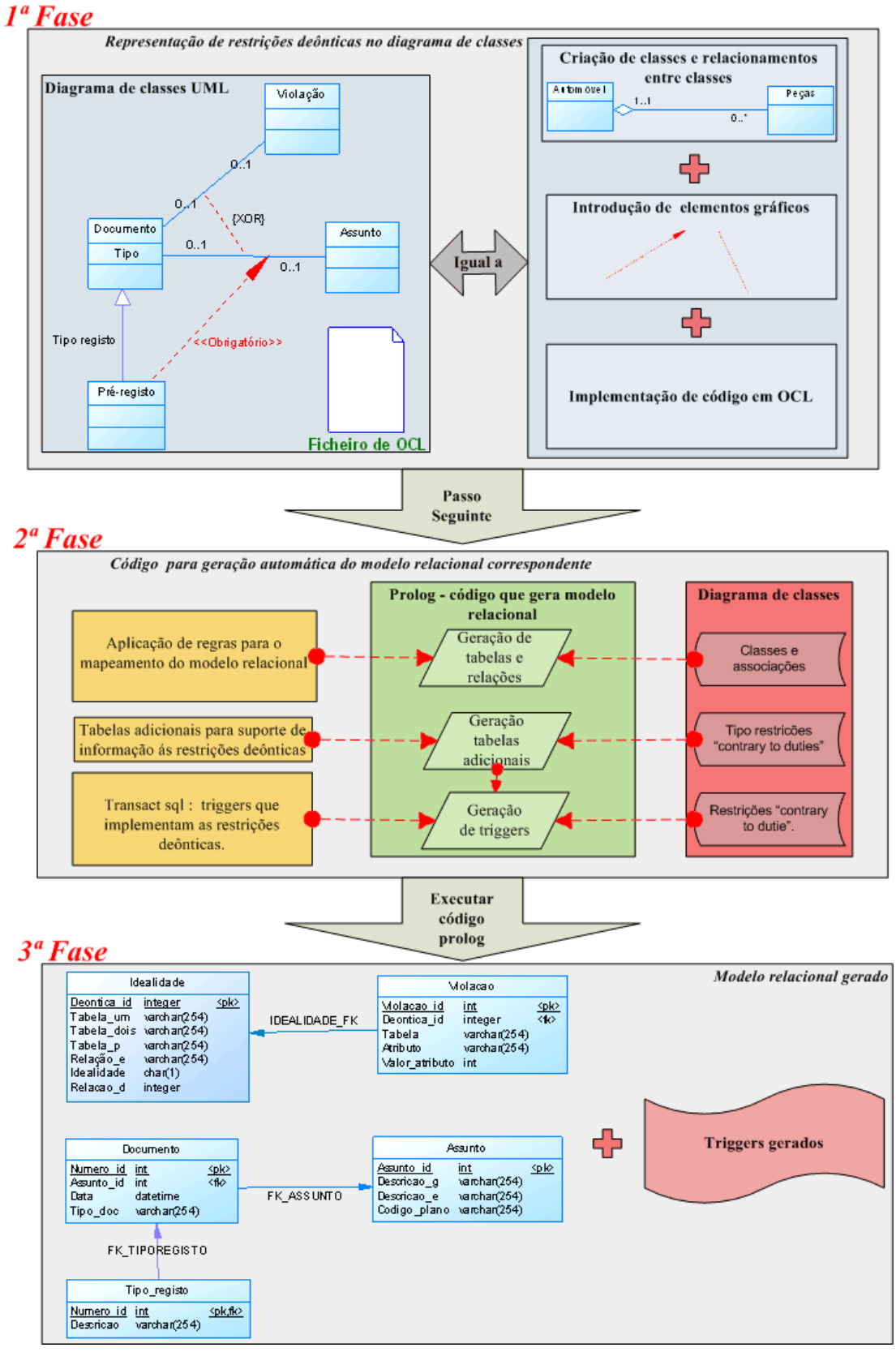


Figura 3.15. Processo de implementação das restrições "contrary to duties", no diagrama relacional.

Capítulo 4 – Aplicação da metodologia proposta

4.1 Introdução

No decorrer do capítulo três foi definida uma metodologia que serve dois propósitos, por um lado representar, no diagrama de classes, a caracterização e a semântica de dois tipos de restrição “contrary to duties”: “Necessário” e “Não possível”; por outro, proceder à geração do modelo relacional que lhes corresponde.

Essa metodologia, genérica, propõe a geração automática do modelo relacional com recurso à implementação de um processo implementado em três etapas: a aplicação das regras de mapeamento entre o modelo de classes e relacional; a introdução de tabelas adicionais para suporte à caracterização das referidas restrições; e o mapeamento da semântica, no diagrama relacional.

Pretende-se que essa metodologia seja aplicada a dois casos reais de utilização de sistemas de informação. Para o efeito, são apresentados dois casos reais, em funcionamento na Câmara Municipal de Lisboa, que justificam pela sua especificidade, o recurso à utilização de restrições “Contrary to duties”.

Aos dois casos referidos, correspondentes ao sistema de gestão orçamental e sistema de gestão documental, aplicam-se as restrições “Necessário” e “Não possível”, assim o presente estudo pretende aplicar a metodologia genérica, anteriormente definida, no capítulo três, para representar as restrições no diagrama de classes, e proceder ao mapeamento automático das mesmas, no contexto do diagrama relacional correspondente.

4.2.1 Descrição do sistema de gestão orçamental

Os sistemas de gestão orçamental garantem o controlo do planeamento orçamental, por norma o orçamento é definido anualmente e considera duas vertentes essenciais: o lado da receita, e o lado da despesa.

A parte da despesa define os valores globais, atribuídos às diversas despesas. Parte do valor global, dessas despesas, é estimado pela informação que as organizações têm relativamente aos gastos de anos anteriores; a outra parte é definida em função dos novos projectos que se pretendem implementar, e da estimativa de custos que resulta da execução dos mesmos.

Através de um código numérico a que corresponde uma rubrica orçamental, as despesas previstas no orçamento, são classificadas tendo em conta a sua natureza económica, e o serviço que sobre elas tem responsabilidade. A cada rubrica orçamental é atribuída uma dotação orçamental, ou seja um valor estimado do que se pretende gastar no decorrer do ano económico. Para que uma despesa seja autorizada é necessário que seja cabimentada, isto é o valor dessa despesa tem que ser subtraída à correspondente dotação orçamental, da rubrica a que pertence. Após autorização e concretização da despesa, é então efectuado o pagamento, total ou parcialmente, do valor anteriormente cabimentado. Em concreto, o sistema de gestão orçamental que serve de exemplo a este caso está em funcionamento na Câmara Municipal de Lisboa e amplamente disseminado por um conjunto significativo de serviços.

No início do ano, após aprovação do orçamento municipal, o sistema é carregado com os valores atribuídos às diferentes rubricas orçamentais. Posteriormente, as despesas a realizar são classificadas pela rubrica orçamental a que correspondem, o que permite distinguir não só o tipo de despesa, como o serviço responsável. Depois da classificação orçamental é feito o cabimento, ou seja é subtraído à dotação da rubrica orçamental o valor da despesa. A partir desse momento fica cativo um valor para aquela despesa, o que possibilita que mais tarde, e após autorização e concretização da despesa, os valores sejam pagos aos respectivos fornecedores.

A informação financeira, que se retira de um sistema de gestão orçamental é sempre crítica para qualquer organização, o caso específico da Câmara de Lisboa não foge à regra, por isso é importante garantir a fiabilidade e rigor do sistema de informação. Efectivamente, no caso do sistema em uso na Câmara de Lisboa existem requisitos obrigatórios, a cumprir pelo sistema de informação, que garantem o rigor e fiabilidade esperados. Por exemplo, no processo de registo de uma despesa, o sistema tem que garantir que existe saldo nessas rubricas, tem que designar o documento que dá origem à despesa, introduzir a data do documento, afectar o fornecedor, etc. No caso concreto do fornecedor, é obrigatório que o mesmo já se encontre registado no sistema, com um código numérico identificativo, para que possa ser seleccionado no processo de registo do cabimento. Isto acontece porque quem fornece à Câmara de Lisboa tem que apresentar um conjunto de documentos que o habilitam a ser fornecedor. Só após entrega e verificação dessa

documentação é possível criar o registo do fornecedor e fazer a sua afectação ao sistema de gestão orçamental.

O cenário descrito implica o cumprimento de um requisito obrigatório, (afectação do cabimento ao fornecedor), sem o qual não é possível registar o cabimento:

1) *O requisito obrigatório é cumprido* (é estabelecida a relação directa entre o cabimento e o fornecedor) – O cabimento é registado.

2) *O requisito obrigatório não é cumprido* (não é estabelecida a relação directa entre o cabimento e o fornecedor) – O cabimento não é registado.

Conforme ilustrado na figura 4.1, o cumprimento do requisito obrigatório garante a integridade do sistema.

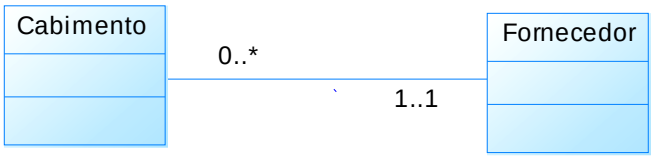


Figura 4.1. Restrição obrigatória entre a classe Cabimento e a classe Fornecedor.

No entanto, surgem por vezes situações que pela excepcionalidade, demonstram que nem sempre o sistema reflecte na íntegra realidade.

Para que uma despesa seja autorizada, é necessário que o seu valor seja cativo, ou seja tem que ser cabimentado. No caso de a despesa ser destinada a um novo fornecedor, a sua autorização fica dependente da criação do código de fornecedor. A criação do código, embora seja responsabilidade do fornecedor, depende de documentos específicos emitidos por organismos exteriores à Câmara de Lisboa, o que significa uma demora considerável. Isto significa que, por exemplo, que uma despesa urgente tem que aguardar pelo código do fornecedor para ser autorizada, situação que não é sustentável.

Para contornar este problema poder-se-ia pensar em retirar a restrição inicialmente imposta na figura 4.1 e permitir o registo de cabimentos sem fornecedor, mas esta seria uma solução muito fraca porque para além de empobrecer a representação da realidade, seria um factor potenciador de inconsistência.

Para garantir a consistência e ao mesmo tempo representar a realidade é necessário manter a restrição obrigatória da figura 4.1 e ter capacidade para registar as violações decorrentes de situações excepcionais. É necessário portanto enriquecer o diagrama de classes com elementos capazes de representar não só a restrição que idealmente devia ser obrigatória, mas também a sua violação. Para isso a utilização do conceito de restrição deôntica é o mais adequado, conforme ilustrado na figura 4.2. A inclusão do operador "XOR" garante exclusividade entre as duas associações i.e. ou a associação Cabimento/Fornecedor se verifica ou a associação Cabimento /Violação tem que ser cumprida.

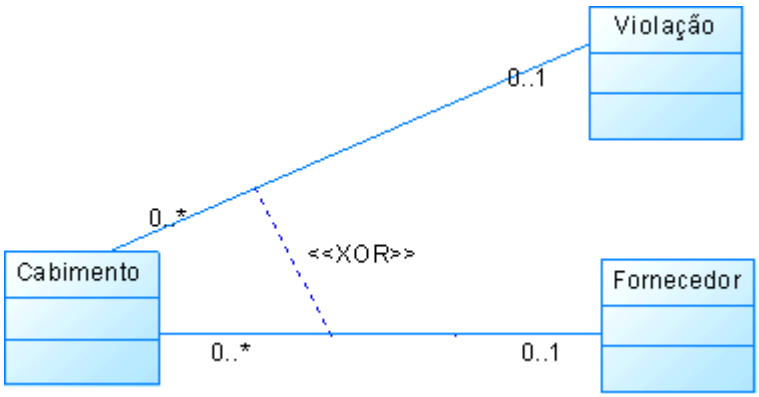


Figura 4.2. Representação da restrição deôntica Obrigação aplicada no caso concreto em estudo.

Retornando à situação do cabimento, se a violação for registada no sistema, mais tarde poderá ser correctamente afectado ao correspondente código de fornecedor.

Embora o problema pareça resolvido existe outra questão a considerar: se a criação do fornecedor demorar e entretanto a despesa se concretizar, deverá ser feito o pagamento ao fornecedor. O problema é que ao fazer esse pagamento, o sistema não sabe a quem pagou, porque a despesa (neste caso o cabimento) não está afecta ao fornecedor.

A solução para este problema parece complicada, mas se a associação entre o cabimento e o pagamento for travada até que o cabimento cumpra a obrigação de estar associado a um fornecedor teremos a questão resolvida. Para que isso seja possível é necessário enriquecer o diagrama de classes com a restrição definida como "Não possível", conforme Figura 4.3

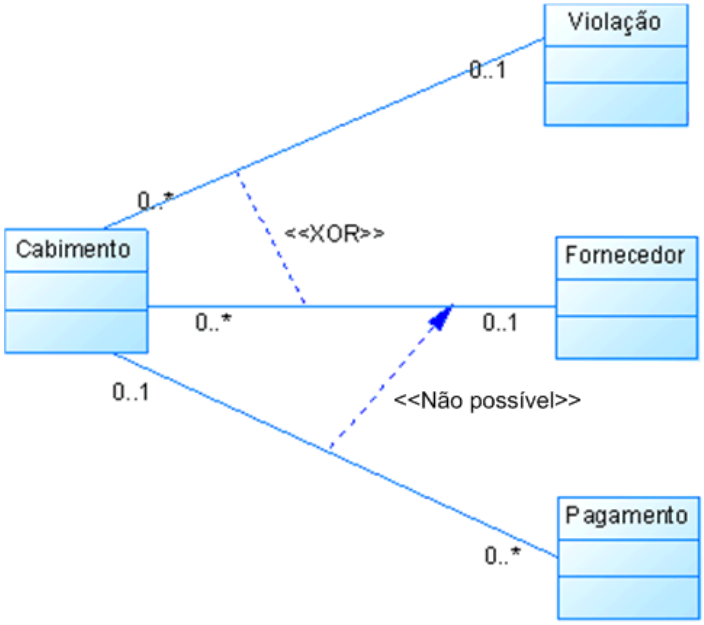


Figura 4.3. Representação da restrição "Não possível" que indica que a associação entre Cabimento/Pagamento é proibida até que o cabimento seja afecto ao fornecedor.

A aplicação em UML do estereótipo "Não possível" indica que a associação entre as classes Cabimento/Pagamento é proibida até que a associação Cabimento/Fornecedor se efective.

4.2.2 Representação do caso real no diagrama de classes

A representação de restrições "Contrary to duties", no diagrama de classes, pode ser repartida em três etapas. A primeira corresponde à criação das classes e definição das associações entre as mesmas; de seguida são criados os elementos gráficos que representam a restrição "Contrary to duties"; por fim o código em OCL adiciona a semântica, intrínseca a cada restrição.

Na figura 4.4 as três etapas propostas para a caracterização de restrições "Contrary to duties" são aplicadas ao caso real do sistema de gestão orçamental, assim foram criadas as classes, atributos, e associações; foram adicionados novos símbolos gráficos, que designam o tipo de restrição utilizada; e foi adicionado o código, em OCL, que define a semântica da

restrição utilizada. As etapas para a representação de restrições "Contrary to duties" e a sintaxe do código em OCL foram anteriormente propostas, no decorrer do capítulo três.

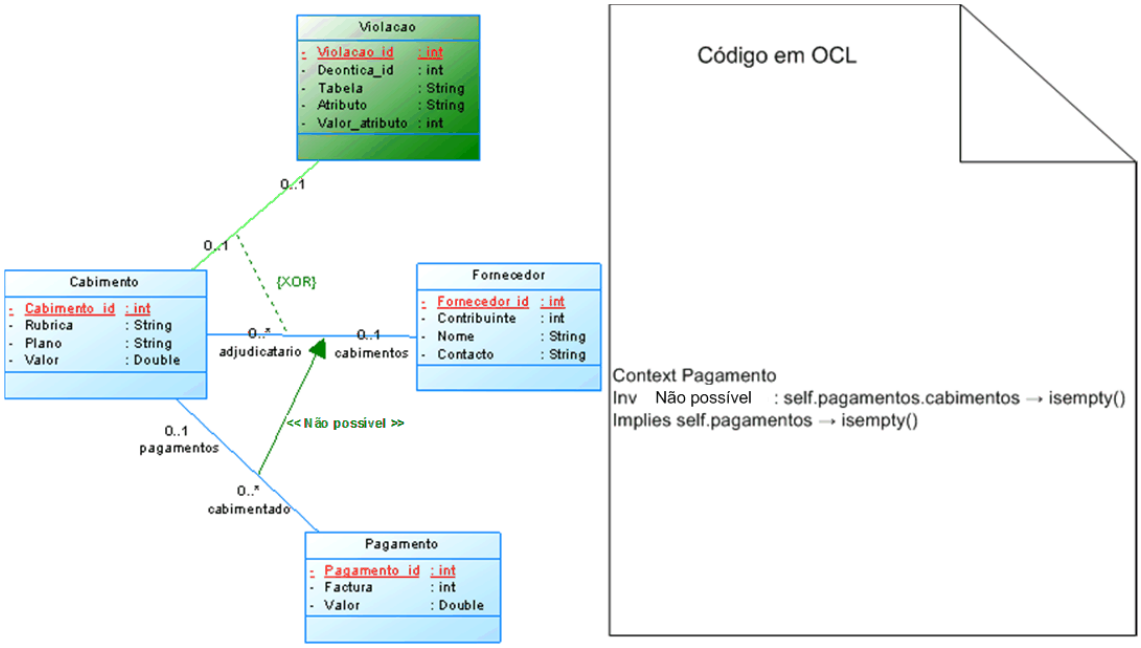


Figura 4.4. Representação da restrição "Não possível" aplicada ao sistema de gestão orçamental.

No contexto do sistema de gestão orçamental a adição de código, em OCL, complementa a caracterização do comportamento esperado do sistema nas situações em que a obrigação inicial é violada. No caso concreto do sistema de gestão orçamental a obrigação inicial corresponde à associação entre as classes "Cabimento" e "Fornecedor". Sempre que essa obrigação é violada, a associação entre "Cabimento" e "Pagamento" deixa de ser possível, até que a obrigação inicial seja cumprida.

4.2.3 Mapeamento para o modelo relacional

Com a representação gráfica e semântica da restrição "Não possível", no diagrama de classes, torna-se necessário proceder à implementação da fase seguinte, ou seja proceder ao

mapeamento para o modelo relacional. O mapeamento para o modelo relacional obedece a uma metodologia proposta durante o capítulo três deste estudo. A primeira etapa dessa metodologia corresponde à aplicação das regras generalistas de mapeamento do modelo de classes para o modelo relacional; a segunda etapa procede à criação de tabelas adicionais e ao registo de restrições “Contrary to duties”; por fim a geração de triggers adiciona o comportamento esperado de cada restrição. As regras generalistas de mapeamento entre os modelos de classes e relacional são explicadas em detalhe no capítulo dois, as tabelas adicionais e a criação de triggers são propostos no decorrer do capítulo três.

Para aplicar as regras de mapeamento, entre os modelos de classes e relacional, é necessário distinguir os elementos que têm tradução, no caso em estudo os elementos gráficos que representam a restrição “Contrary to duties” e o código em OCL que lhes adiciona semântica não têm correspondência directa para o modelo relacional, portanto não é possível o mapeamento, segundo as mesmas regras. Existe também outro pormenor a ter em conta, o caso da classe “Violação”, em que a aplicação das regras de mapeamento significaria a criação de duas tabelas com estrutura e nomes iguais. Isto acontece porque são considerados dois casos reais que utilizam a mesma classe, “Violação”, para proceder ao registo de obrigações que não foram cumpridas. Na verdade a criação de duas classes com o mesmo nome é desnecessária, por isso a solução proposta, no capítulo três, corresponde à não aplicação das regras de mapeamento sobre esta classe, em vez disso é proposta a sua criação como tabela adicional. A figura 4.5 demonstra o resultado da aplicação das regras de mapeamento entre os dois modelos.

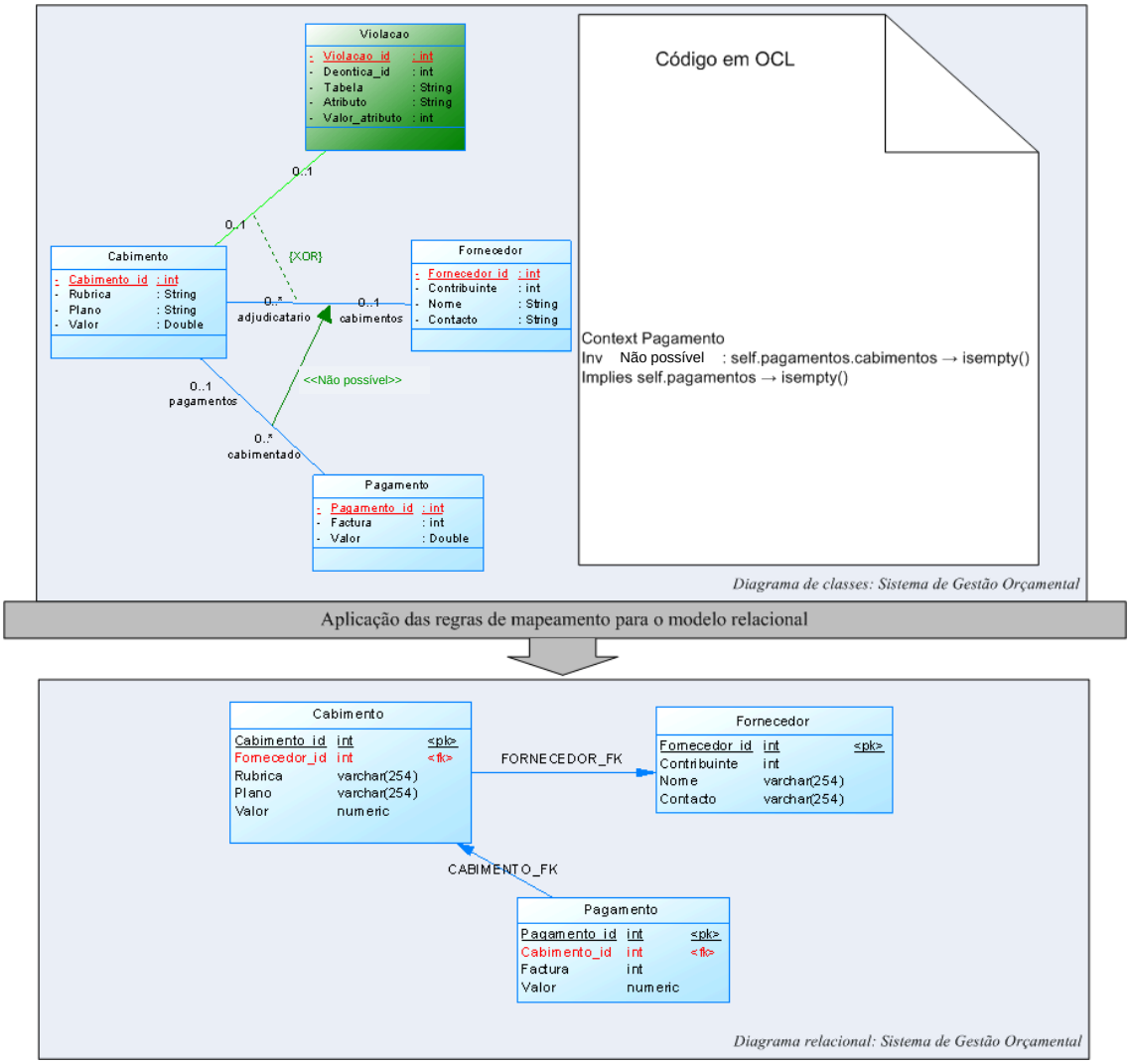


Figura 4.5. Aplicação das regras de mapeamento entre os modelos de classes e relacional: sistema de gestão orçamental.

No “Anexo 1” é demonstrada, em detalhe, a aplicação das regras de mapeamento para o caso do sistema de gestão orçamental. O resultado final da aplicação das regras é visível no diagrama relacional da figura 4.5.

A segunda etapa da geração do modelo relacional corresponde à criação de tabelas adicionais e ao registo das restrições “Contrary to duties” nas mesmas. Assim, no caso concreto do sistema de gestão orçamental a restrição “Não possível” liga as duas associações que a compõem: a associação correspondente à obrigação inicial entre as classes “Cabimento” e “Fornecedor”, e a associação entre as classes “Cabimento” e “Pagamento”, correspondente à associação que não pode ocorrer, enquanto a obrigação

inicial não for cumprida. A figura 4.6 descreve os dados que devem ser guardados nas tabelas adicionais.

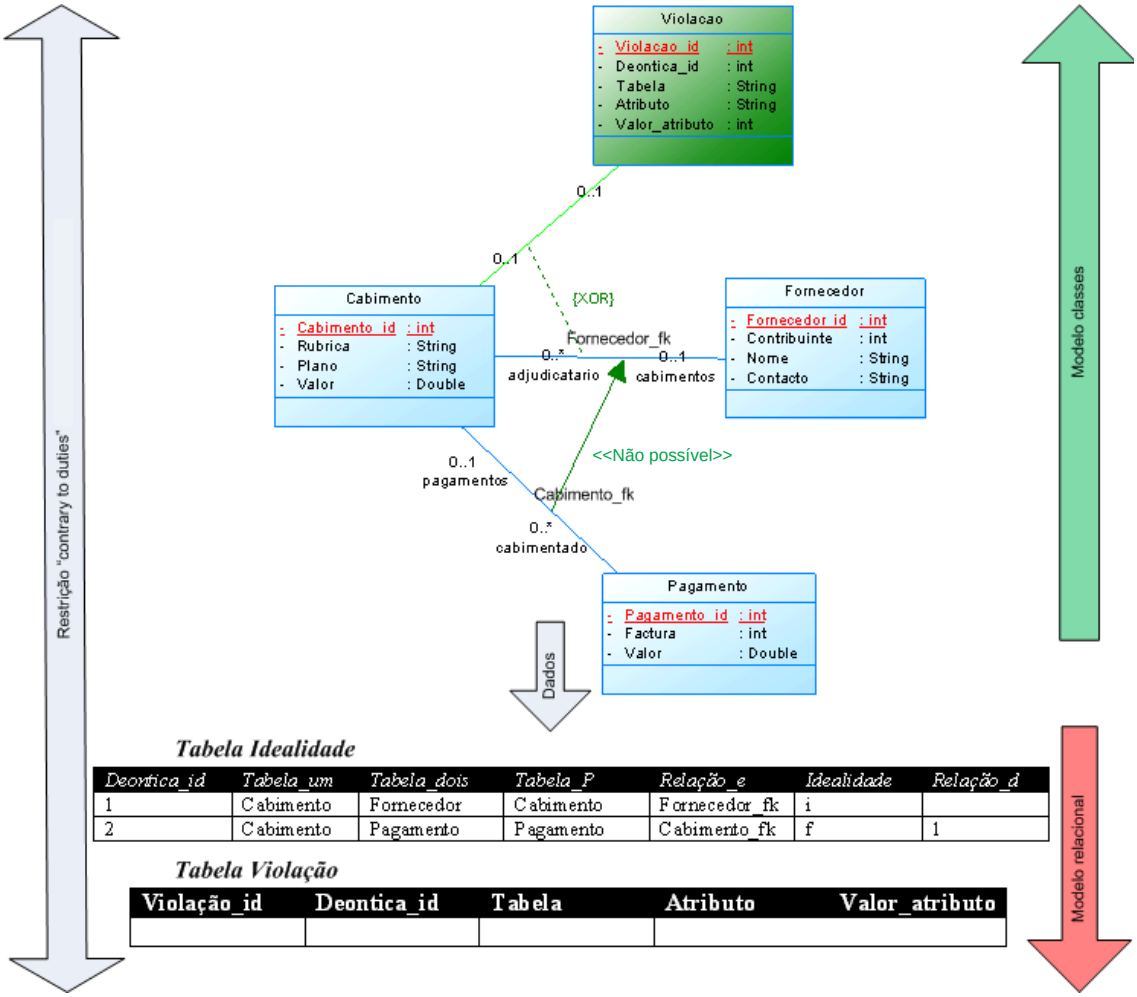


Figura 4.6. Criação e registo de dados nas tabelas adicionais, para o caso do sistema de gestão orçamental.

Os dados guardados pela tabela "Idealidade", figura 4.6, caracterizam a restrição "Contrary to duties" imposta ao sistema de gestão orçamental. A tabela "Violação" também é criada no modelo relacional como tabela adicional, tem como função proceder ao registo das violações entre as classes "Cabimento" e "Fornecedor".

Após aplicação das regras de mapeamento e da geração de tabelas complementares, resta implementar e descrever os triggers que suportam o comportamento semântico da restrição "Não possível" e concluir o mapeamento do caso real anteriormente descrito para

o modelo relacional. Os triggers são explicados no capítulo dois, e a sua utilização neste estudo é proposta no decorrer do capítulo três.

Genericamente, no caso da restrição “Não possível”, sempre que exista uma violação à associação de obrigação entre duas classes torna-se necessário que outra associação, que implementa a restrição deôntica “Não possível”, deixe de ser permitida. Na prática, a violação da obrigação inicial corresponde, no modelo relacional, à existência de um valor nulo no atributo que corresponde à chave estrangeira do relacionamento. Como no modelo relacional existem três comandos com capacidade de alterar dados: “INSERT”, “UPDATE”, e “DELETE”, cada vez que um desses comandos ocorrer é necessário accionar código, que verifique o valor da chave estrangeira, e decida as acções a tomar. Para isso é necessário implementar triggers, conforme proposto no decorrer do capítulo três. A figura 4.7 descreve a implementação dos triggers, aplicados ao caso real em estudo.

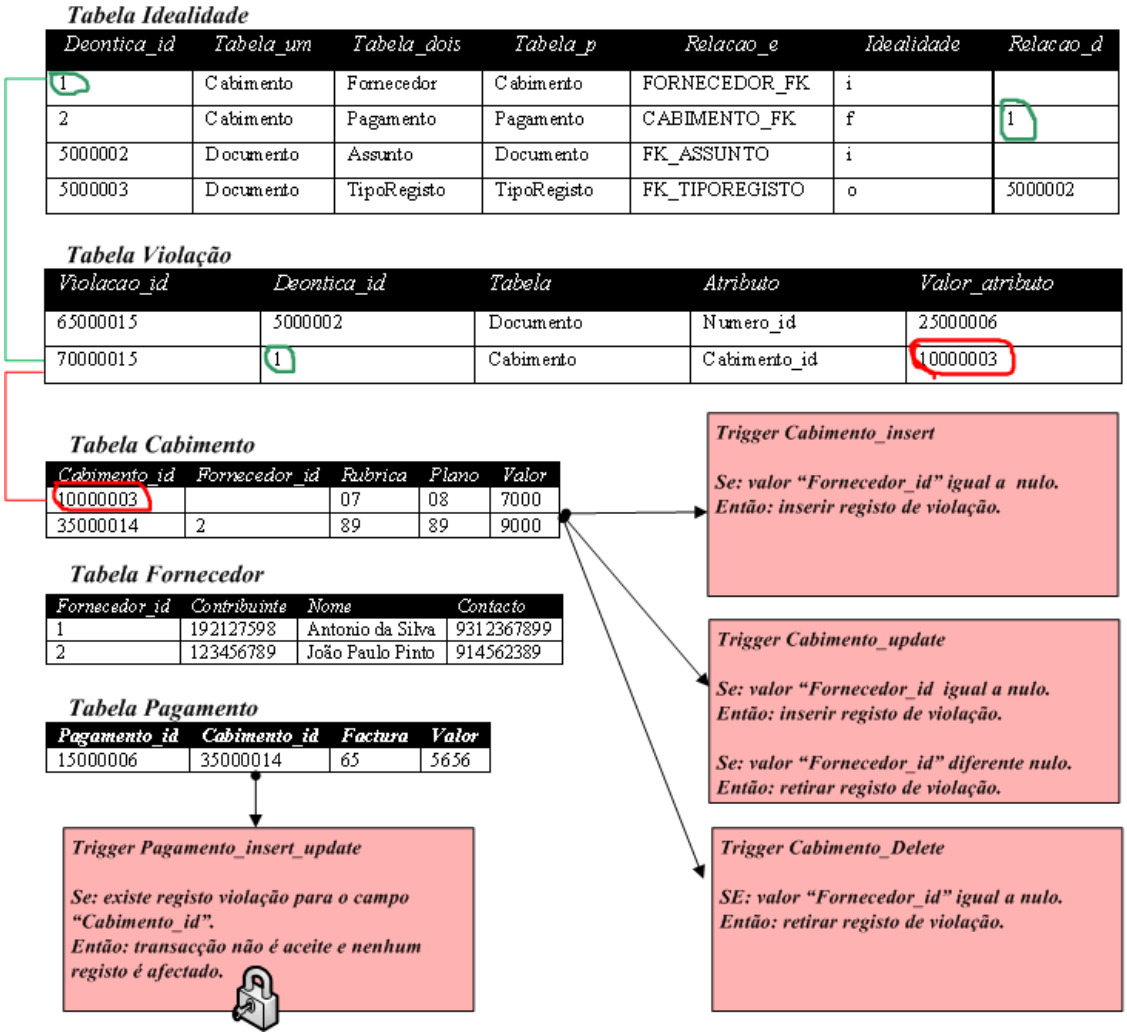


Figura 4.7. Resultado do mapeamento integral da restrição "Não possível" para o modelo relacional, e aplicação ao sistema de gestão orçamental.

A figura 4.7 descreve a estrutura relacional que suporta o tipo de restrições "Não possível", e demonstra a sua aplicação ao caso do sistema de gestão orçamental. Para melhorar a compreensão do funcionamento da arquitectura descrita são introduzidos alguns dados nas tabelas que compõem a arquitectura referida.

Analisando os dados da figura 4.7, verifica-se que o primeiro registo da tabela "Cabimento" não tem fornecedor atribuído, por essa razão existe um registo na tabela violação, que indica uma violação a uma restrição deôntica inicialmente imposta. A relação, entre as tabelas "Violação" e "Idealidade" associa o registo de violação com o tipo de restrição deôntica imposta. Neste caso, os registos da tabela "Idealidade" com identificadores únicos "1" e "2" estão relacionados entre si, e caracterizam a restrição

"Contrary to duties" "Não possível". O registo de violação foi introduzido por um dos triggers da tabela "Cabimento", em resposta à violação do registo com o identificador único "10000003", da mesma tabela. O trigger "Pagamento_insert_update" verifica que existe uma violação para o registo "10000003", e não permite a introdução de registos de pagamento, enquanto a situação de violação de mantiver. O código integral de cada um dos triggers da figura 4.7 está descrito no Anexo dois.

4.3.1 Descrição do sistema de gestão documental

Um sistema de gestão documental engloba várias fases procedimentais no tratamento lógico e no suporte físico da documentação. Algumas dessas fases são o registo de documentação vinda do exterior, a digitalização, o registo dos documentos produzidos pelos serviços, a definição do ciclo de vida dos documentos, a implementação de normas arquivísticas e de preservação digital, a validação lógica/física dos movimentos internos de documentos, a pesquisa simples/avançada/estruturada, a produção de relatórios estruturados, e o arquivo da documentação.

A fase do registo da documentação detém uma importância acrescida neste processo, uma vez que a qualidade da caracterização do documento depende em grande parte do rigor com que o operador introduz a informação respectiva. A dimensão da organização, a diferenciação de funções, e o grau de complexidade que os documentos tratam determinam o grau de profundidade com que as regras normativas da utilização do sistema de gestão documental deve ser precedido.

Em concreto, o sistema de gestão documental que serve de exemplo a este caso está em funcionamento na Câmara Municipal de Lisboa e amplamente disseminado por um conjunto significativo de serviços, alguns dos quais abertos ao público, o sistema conta com uma interface web, é centralizado e acessível a toda a organização.

No processo de registo de um documento, o assunto tem como função a compartimentação dos documentos. A importância de compartimentar por assunto é justificada pelo número elevado de documentos que dão entrada e são produzidos diariamente nos serviços. O assunto tem um papel importante na determinação do ciclo de vida da documentação (conforme o assunto o ciclo de vida dos documentos difere), garante

maior rapidez no acesso à informação, promove fiabilidade nos relatórios de gestão, e gera eficácia ao nível do arquivo da documentação. A sua introdução no registo do documento é portanto obrigatória.

O cenário descrito implica o cumprimento de um requisito obrigatório, (afectação do assunto ao documento), sem o qual não é possível a geração do documento.

1) *O requisito obrigatório é cumprido* (é estabelecida a relação directa entre o documento e o assunto) – O documento é gerado.

2) *O requisito obrigatório não é cumprido* (não é estabelecida a relação directa entre o documento e o assunto) – O documento não é gerado.

Conforme ilustrado na figura 4.8, o cumprimento do requisito obrigatório garante a integridade do sistema.

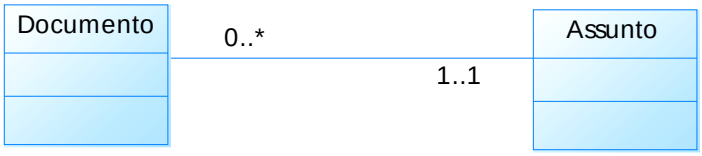


Figura 4.8. Restrição obrigatória entre a classe Documento e a classe Assunto

No entanto, nem sempre a utilização, em real, do sistema permite o cumprimento da restrição obrigatória. Na organização em estudo, a Câmara Municipal de Lisboa, existem variados serviços a fazer atendimento ao público, que registam a documentação proveniente dos munícipes no sistema de gestão documental. Os assuntos a tratar são variados e por vezes o funcionário que atende tem dificuldades em perceber em que categoria de assunto deve incluir aquele caso em particular. Dizer ao munícipe que volte mais tarde porque o assunto ainda não está caracterizado no sistema não é, obviamente, uma opção, pela péssima imagem que o serviço estaria a dar.

Para contornar este problema poder-se-ia pensar em retirar a restrição inicialmente imposta na figura 4.8 e permitir a geração de documentos sem assunto, mas esta seria uma solução muito fraca porque para além de empobrecer a representação da realidade, seria um factor potenciador de inconsistência.

Para garantir a consistência e ao mesmo tempo representar a realidade é necessário manter a restrição obrigatória da figura 4.8 e ter capacidade para registar as violações decorrentes de situações excepcionais. É necessário portanto enriquecer o diagrama de classes com elementos capazes de representar não só a restrição que idealmente devia ser obrigatória, mas também a sua violação e para isso a utilização do conceito de restrição deôntica é o mais adequado, conforme ilustrado na figura 4.9. A inclusão do operador "XOR" garante exclusividade entre as duas associações i.e. ou a associação Documento/Assunto se verifica ou a associação Documento /Violação tem que ser cumprida.

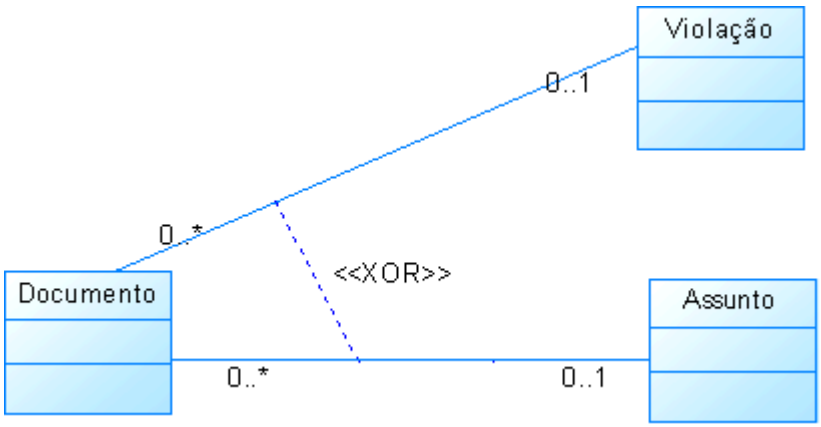


Figura 4.9. Representação da restrição deôntica Obrigação aplicada no caso concreto em estudo.

Na prática, retornando à situação do funcionário que não sabe o assunto, se a violação é registada no sistema o documento poderá mais tarde ser visto por pessoal especializado que fará a afectação do caso particular referido a uma categoria de assunto que julgue adequada, ou no limite propor a abertura de uma nova categoria de assunto, resolvendo desta forma a violação.

Aparentemente a violação será resolvida num prazo considerado razoável, o tempo necessário para afectar o caso a uma categoria de assunto, mas numa situação limite em que tem que ser criada uma nova categoria a violação poder-se-á diluir no tempo. A razão da demora reside na dependência que dois importantes sub-processos de gestão documental têm relativamente ao assunto. A determinação do ciclo de vida dos documentos e a definição das normas de preservação digital são os dois sub-processos que

dependem do assunto e envolvem recursos humanos especializados que justamente têm competência para ajustar os sub-processos no contexto da nova categoria.

Resumindo, verifica-se que vão existir documentos sem assunto durante um período de tempo considerável. Apesar da situação parecer sustentável, a geração de relatórios para a gestão pode gerar conflitos se, por exemplo, aparecerem os tais documentos sem assunto. Dependendo do gestor e dos seus conhecimento de sistemas de informação explicar o porquê desta situação pode não ser tarefa fácil.

A solução para este problema passa por classificar este tipo de registos. Como os registos de documento podem ser classificados como "vivo" e "arquivado", a solução passa por obrigar a que os documentos sem assunto sejam classificado como "pré-registo."

Efectivamente, o sistema atribui automaticamente a condição de "vivo" quando numera um documento e "arquivado" quando um documento é arquivado. No caso de o documento não ter assunto, passa a ser obrigatória a classificação do mesmo como "pré-registo", o que significa que violação da associação Documento/Assunto obriga a que a associação entre Documento/Tipo registo seja obrigatória e que o tipo de registo assuma o valor "pré-registo". A figura 4.10 ilustra a situação descrita.

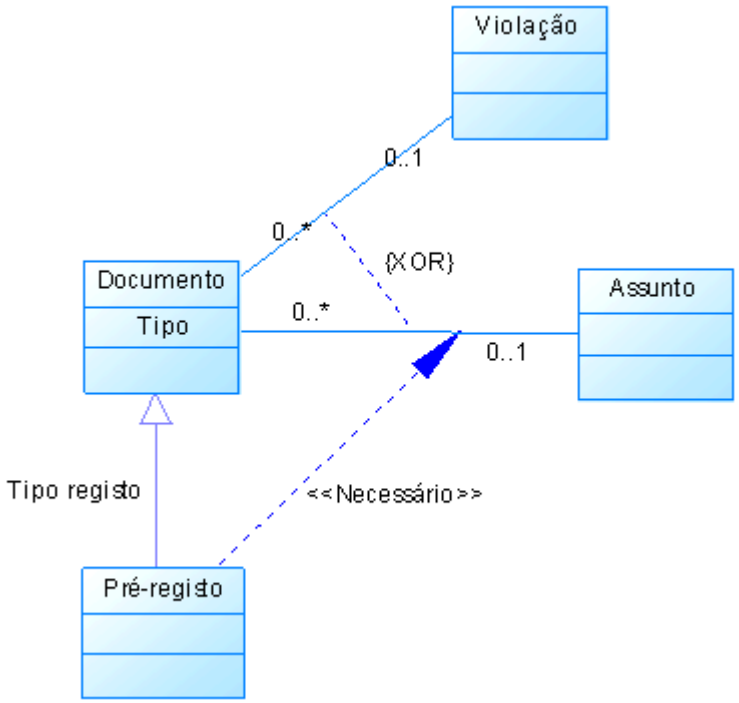


Figura 4.10. Representação da restrição "Necessário" que significa que os registos de documentos sem assunto têm que ser classificados como "pré-registo".

A aplicação em UML do estereótipo “Necessário” indica que existe uma obrigação decorrente de uma violação a uma associação de dados obrigatória, neste caso ao ser violada a associação entre as classes Documento/Assunto a associação Tipo registo torna-se obrigatória e tem que assumir o valor “pré-registo”.

4.3.2 Representação do caso real no diagrama de classes

A caracterização de restrições “Contrary to duties no diagrama de classes, aplicado ao caso real do sistema de gestão documental, é similar ao processo descrito anteriormente, neste capítulo, para o sistema de gestão orçamental, o que difere é o tipo de restrição utilizada. As etapas para a representação de restrições “Contrary to duties” e a sintaxe do código em OCL foram anteriormente propostas, no decorrer do capítulo três do presente estudo. Na figura 4.11 o caso do sistema de gestão documental é representado graficamente, conjuntamente com o código, escrito em OCL, que adiciona o significado semântico esperado.

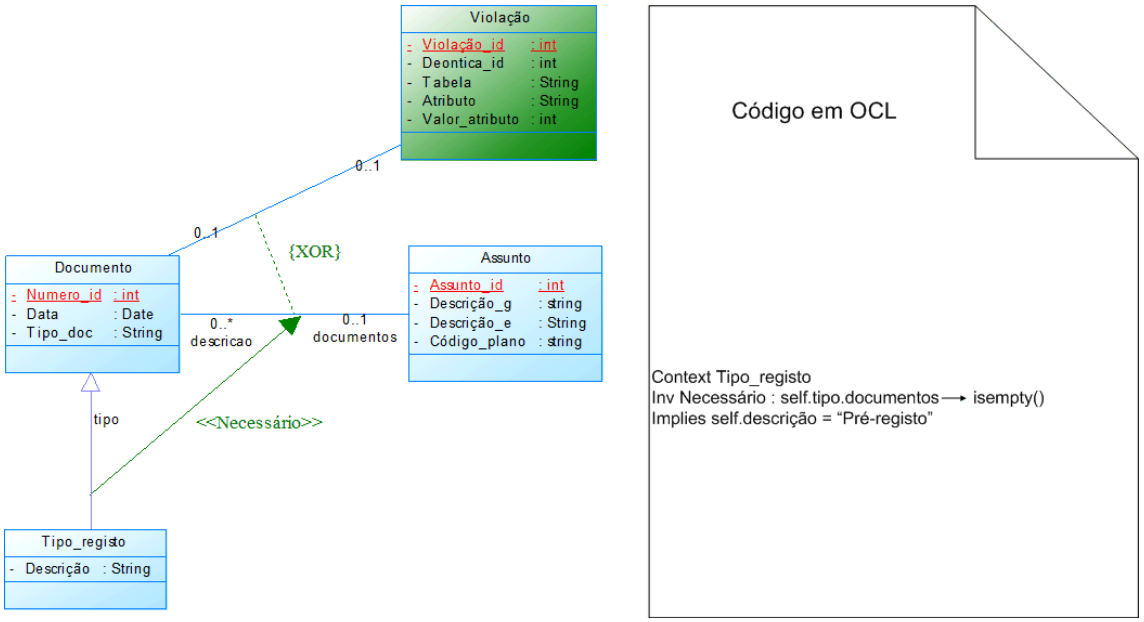


Figura 4.11. Representação da restrição “Necessário” aplicada ao sistema de gestão documental.

No sistema de gestão documental a adição de código, em OCL, complementa a caracterização do comportamento esperado do sistema nas situações em que a obrigação inicial é violada. No caso do sistema de gestão documental a obrigação inicial corresponde à associação entre as classes “Documento ” e “Assunto”. Sempre que essa obrigação é violada, a associação entre “Documento” e “Tipo_registo” passa a ser necessária, ou seja obrigatória sem excepções, para além disso o campo “Descrição” assume o valor “Pré-registo”.

4.3.3 Mapeamento para o modelo relacional

Após representação gráfica e semântica da restrição “ Necessário”, é preciso proceder ao mapeamento para o modelo relacional. O mapeamento para o modelo relacional obedece a uma metodologia proposta durante o capítulo três deste estudo. A primeira etapa dessa metodologia corresponde à aplicação das regras generalistas de mapeamento do modelo de classes para o modelo relacional; a segunda etapa procede à criação de tabelas adicionais e ao registo de restrições “Contrary to duties”; por fim a geração de triggers adiciona o comportamento esperado de cada restrição. As regras generalistas de mapeamento entre os modelos de classes e relacional são explicadas em detalhe no capítulo dois, as tabelas adicionais e a criação de triggers são propostos no decorrer do capítulo três. A figura 4.12 demonstra o resultado da aplicação das regras de mapeamento entre os dois modelos.

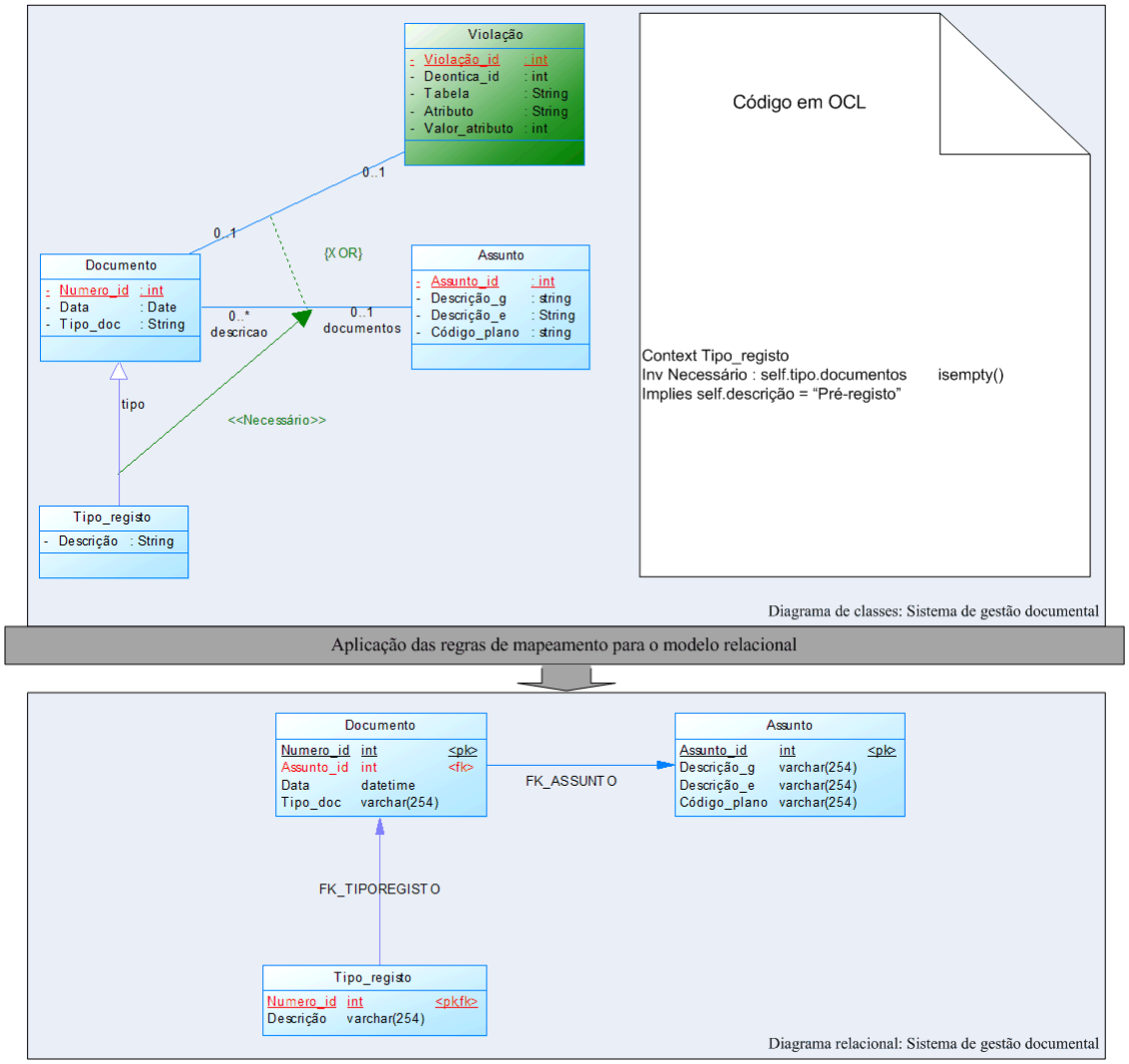


Figura 4.12. Aplicação das regras de mapeamento entre os modelos de classes e relacional: sistema de gestão documental.

No Anexo 1 é demonstrada, em detalhe, a aplicação das regras de mapeamento para o caso do sistema de gestão documental. O resultado final da aplicação das regras é visível no diagrama relacional da figura 4.12.

A segunda etapa da geração do modelo relacional corresponde à criação de tabelas adicionais e ao registo das restrições "Contrary to duties". No caso do sistema de gestão documental a restrição "Necessário" liga as duas associações que a compõem: a associação correspondente à obrigação inicial entre as classes "Documento" e "Assunto", e a associação entre as classes "Documento" e "Tipo_registo", correspondente à associação

que se torna obrigatória sem excepções, enquanto a obrigação inicial não for cumprida. A figura 4.13 descreve os dados que devem ser guardados nas tabelas adicionais.

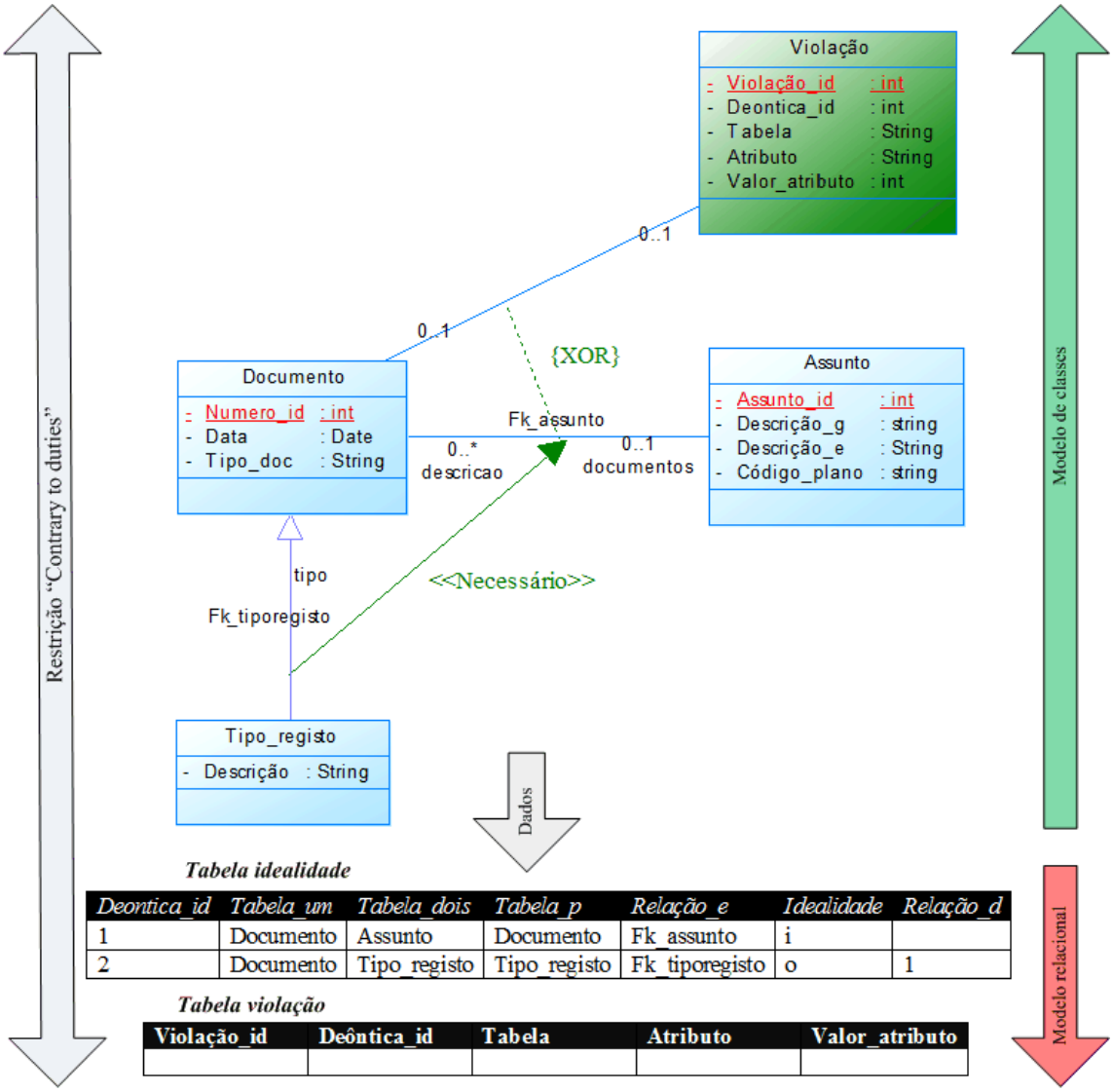


Figura 4.13. Criação e registo de dados nas tabelas adicionais, para o caso do sistema de gestão documental.

Após aplicação das regras de mapeamento e da geração de tabelas complementares, resta implementar e descrever os triggers que suportam o comportamento semântico da restrição "Necessário" e concluir o mapeamento do caso real do sistema de gestão documental para o modelo relacional. Os triggers são explicados no capítulo dois, e a sua utilização é proposta no decorrer do capítulo três.

A figura 4.14 descreve a implementação dos triggers, aplicados ao caso real em estudo.

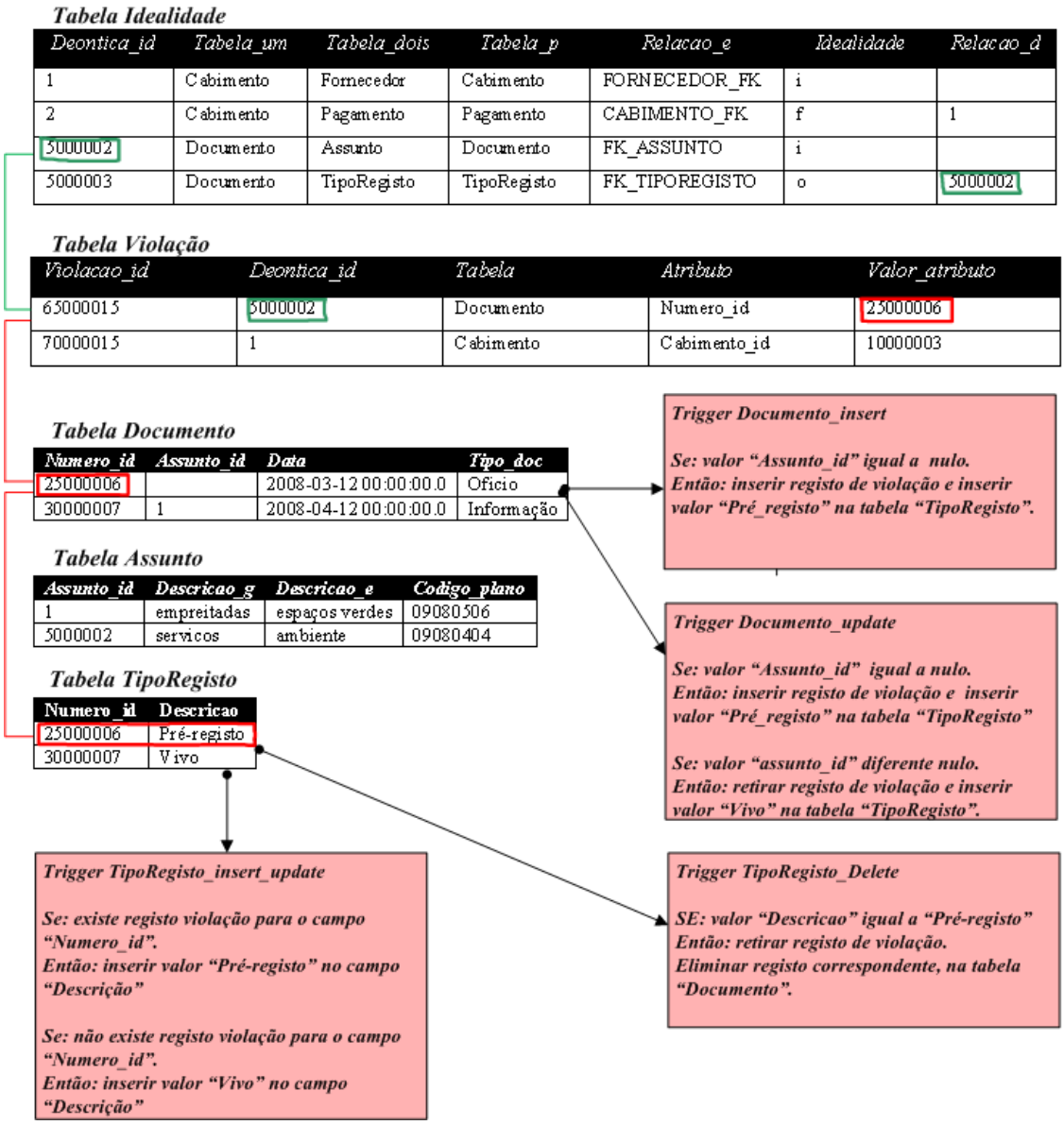


Figura 4.14. Resultado do mapeamento integral da restrição "Não possível" para o modelo relacional, e aplicação ao sistema de gestão documental.

A figura 4.14 descreve a estrutura relacional que suporta o tipo de restrições "Necessário", e demonstra a sua aplicação ao caso do sistema de gestão documental.

Analisando os dados da figura 4.14, verifica-se que o primeiro registo da tabela "Documento" não tem assunto, por essa razão existe um registo na tabela violação que indica uma violação da obrigação inicial. O registo de violação foi introduzido por um dos

triggers da tabela “Documento” em resposta à violação do registo com o identificador único “25000006” da mesma tabela, em paralelo é cumprida a necessidade de afectar o campo “Descrição” da tabela “Tipo_registo” com o valor “Pré-registo”. O código integral de cada um dos triggers da figura 4.14 está descrito no Anexo dois.

4.4 Geração automática do modelo relacional

No decorrer do presente capítulo foram descritos dois casos reais em que as restrições “Necessário” e “Não Possível”, propostas anteriormente no decorrer do capítulo três, se aplicam. No capítulo três foi, também, proposta uma metodologia genérica para geração automática do modelo relacional a partir da caracterização, no diagrama de classes, das restrições “Contrary to duties” “Necessário” ou “Não Possível”. A metodologia referida implementa uma aplicação, desenvolvida na linguagem de programação “Prolog”, que lê informação relevante a partir do diagrama de classes, e que processa os dados para formar o código, na linguagem “Transact SQL”, capaz de criar o modelo relacional correspondente.

A aplicação, escrita em Prolog, e atrás referida, lê toda informação proveniente do diagrama de classes com recurso a um ficheiro de dados, em seguida processa os dados e cria um ficheiro de saída com código em “Transact SQL. A figura 4.15 descreve o funcionamento da aplicação Prolog.

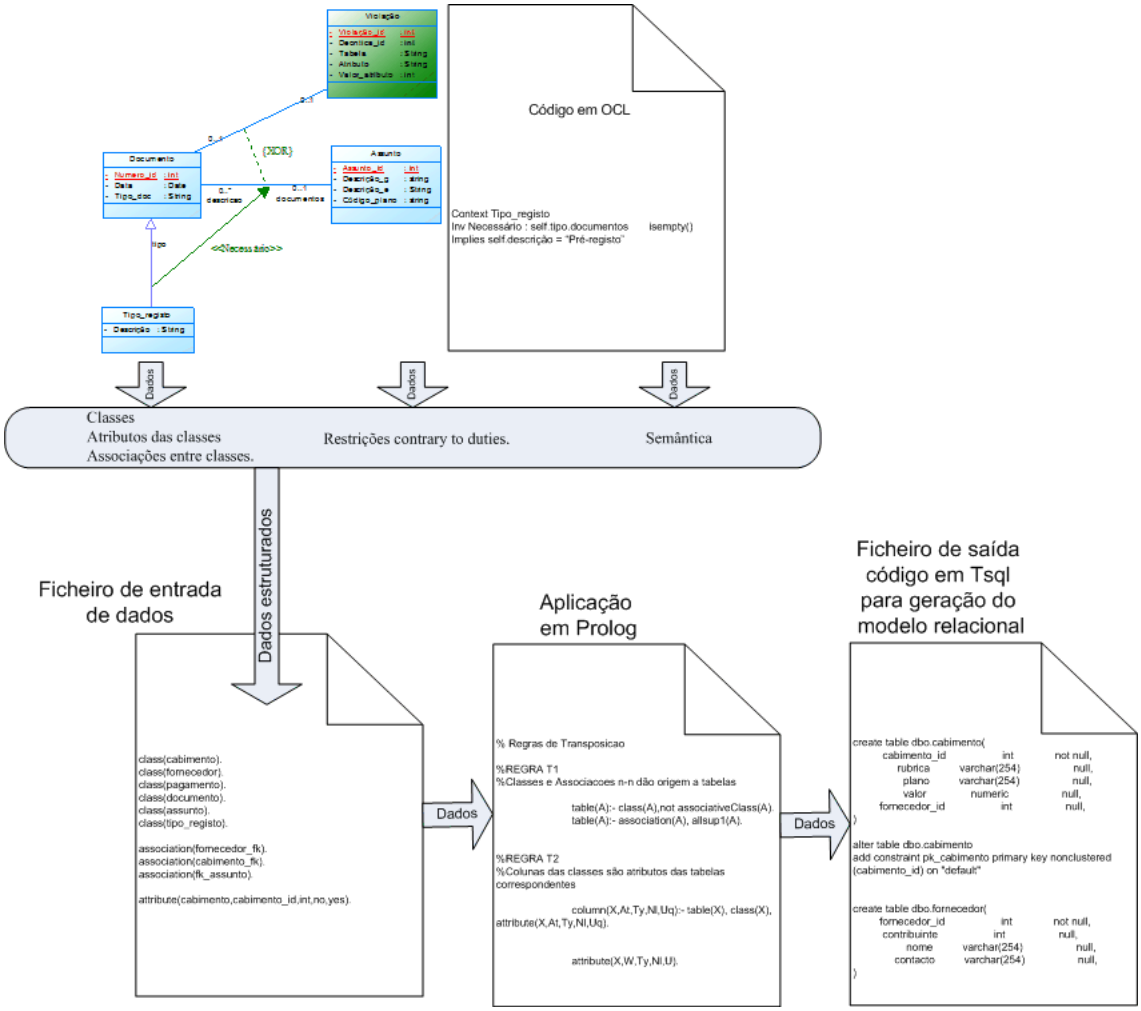


Figura 4.15. Funcionamento da aplicação em Prolog.

Na figura 4.15, o ficheiro que contém o código da aplicação em Prolog, carrega os dados provenientes do ficheiro de entrada de dados, processa a informação, e gera no ficheiro de saída código na linguagem "Transact SQL" que permite a criação do modelo relacional correspondente.

O ficheiro de entrada de dados concentra toda a informação do diagrama de classes de forma estruturada, possibilitando à aplicação de Prolog a sua leitura e carregamento, no anexo quatro é descrita a estrutura genérica dos dados de entrada. Após leitura do ficheiro de dados de entrada a aplicação em Prolog processa a informação através de regras definidas em código. As regras são aplicadas aos dados de forma sequencial, o primeiro conjunto de regras corresponde à aplicação das regras genéricas de mapeamento entre os modelos de classes e relacional, que geram, após processamento, um bloco de código no

ficheiro de saída; em seguida a aplicação cria o código necessário para a criação da estrutura das tabelas adicionais "Idealidade" e "Violação", anteriormente propostas no decorrer do capítulo três, de seguida os dados que caracterizam as restrições "Contrary to duties" são transformados em código para que seja possível o seu registo nas tabelas adicionais; por último é criado no ficheiro de saída o código para criação dos triggers que implementam a semântica das restrições "Contrary to duties". A figura 4.16 ilustra o processo.

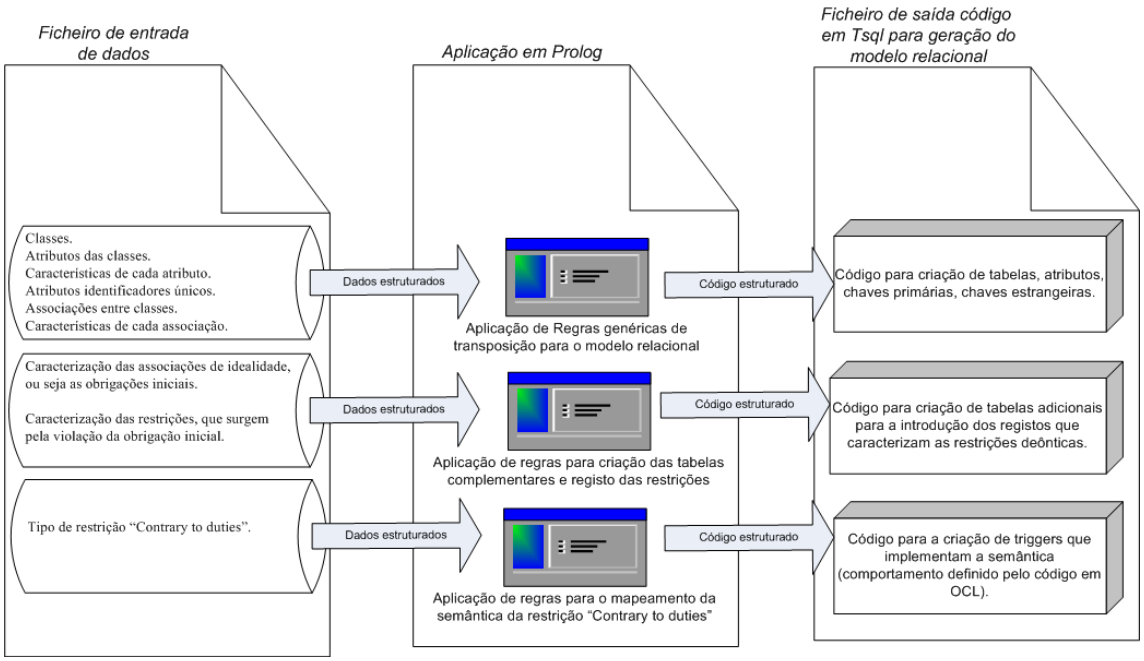


Figura 4.16. Processo de geração automática do modelo relacional com recurso a uma aplicação em Prolog.

Na figura 4.16 a aplicação de Prolog lê a informação estruturada do ficheiro de entrada de dados, aplica as regras que permitem a tradução para o modelo relacional, e escreve, no ficheiro de saída, o código em "Transact SQL" para a criação integral do modelo relacional. A aplicação da solução proposta ao sistema de gestão orçamental e ao sistema de gestão documental conjuntamente com o código integral da aplicação, escrita em Prolog, podem ser consultados no Anexo três do presente estudo. Para aplicação da solução apresentada a outros casos, é necessário introduzir a informação dos mesmos no

ficheiro de entrada de dados e correr a aplicação de Prolog, as instruções para a execução da aplicação em Prolog estão descritas no Anexo cinco do presente estudo.

Capítulo 5 – Síntese e considerações finais

5.1 Síntese.

5.2 Considerações finais.

5.3 Sugestões para investigações futuras.

5.1 Síntese

No decorrer do presente estudo foi apresentada a problemática que envolve a modelação de sistemas de informação, para o efeito foram descritas situações genéricas em que o recurso exclusivo a restrições fortes não consegue representar propriedades importantes da realidade, para a solução do problema foi proposta a utilização de restrições deônticas, em detrimento das restrições fortes, normalmente utilizadas.

No desenvolvimento da dissertação foi proposta, pelo autor, uma metodologia assente no levantamento e descrição de casos reais de funcionamento de sistemas de informação que, pela sua especificidade, justifiquem a utilização de restrições deônticas". No sentido de possibilitar a representação de restrições "Contrary to duties", no diagrama de classes da UML, foi utilizada a proposta de representação gráfica de Ramos (2003), o autor da dissertação propôs a utilização de código, desenvolvido em OCL, para adicionar o significado semântico às mesmas.

Para garantir a persistência da informação, foi escolhido o modelo relacional, no entanto como as restrições deônticas e o código em OCL são representados no diagrama de classes, que implementa o modelo orientado ao objecto, tornou-se necessário mapear os elementos representados no diagrama de classes para o correspondente diagrama relacional. Com recurso a regras de mapeamento genéricas entre os dois modelos é possível traduzir a maior parte dos elementos, para os restantes, sem tradução, foi proposta, pelo autor da dissertação, uma estrutura composta por duas tabelas adicionais, conforme, inicialmente, sugerido por Ramos (2003). A função das tabelas é servir de repositório da informação relativa a restrições "Contrary to duties", para implementar a semântica de cada uma dessas restrições o autor da dissertação propôs a criação de triggers específicos, conforme referido por Ramos (2003).

O contexto e o enquadramento da dissertação são explicados: a UML; o diagrama de classes; o modelo relacional; as regras genéricas de transposição do diagrama de classes para o modelo relacional; as restrições deônticas; e a OCL. Para tornar mais claro a sua utilização, no decorrer da dissertação, são dados alguns exemplos práticos.

No processo de representação das duas restrições "Contrary to duties" propostas: "Necessário" e "Não Possível" é proposta uma arquitectura genérica que utiliza os elementos de modelação da UML; os seus mecanismos de extensão, conforme proposta de

representação gráfica de Ramos (2003); e uma estrutura de código genérica, em OCL, proposta pelo autor da dissertação.

Na passagem das restrições para o modelo relacional é proposta uma metodologia repartida por três fases: aplicação das regras generalistas de mapeamento entre os dois modelos, conforme descrito por Ramos (2006); registo da informação que caracteriza o tipo de restrições em tabelas adicionais, conforme proposto inicialmente por Ramos (2003), e com a estrutura genérica, a implementar, proposta pelo autor da dissertação; e finalmente com a formação de triggers que implementam a semântica das restrições, propostos inicialmente por Ramos (2003), e com uma proposta de estrutura genérica e posterior desenvolvimento pelo autor da dissertação.

Para que seja possível a geração automática do diagrama relacional é proposta, pelo autor da dissertação, uma arquitectura genérica e posterior desenvolvimento de código, em Prolog, que a partir da leitura do diagrama de classes se encarrega de fazer o mapeamento e criação automática do modelo relacional correspondente.

Foram levantados e posteriormente descritos dois casos reais de utilização de sistemas de informação que pela especificidade se enquadraram na utilização das restrições “Contrary to duties” “Necessário” e “Não Possível”, os dois casos encontram-se em funcionamento na Câmara Municipal de Lisboa, e correspondem respectivamente ao sistema de gestão documental e ao sistema de gestão orçamental. A metodologia proposta no decorrer do capítulo três foi aplicada a cada caso descrito, o que correspondeu à caracterização gráfica e semântica dos sistemas no diagrama de classes, e à geração automática do modelo relacional que lhes corresponde.

No inicio do presente estudo foi proposto o desenvolvimento de código em OCL para adicionar a semântica, ou seja o comportamento esperado das restrições deônticas propostas, o objectivo inicial foi atingido com êxito no decorrer da pressente investigação. O mapeamento do comportamento esperado para o diagrama relacional foi, também, proposto e atingido com êxito através do desenvolvimento de uma estrutura de triggers de estrutura fixa, desenvolvidos em “Transact SQL”. Por fim a proposta de desenvolver uma aplicação que se encarrega de ler a informação do diagrama de classes, incluindo as restrições deônticas propostas, e proceder à criação automática do modelo relacional que lhe corresponde, incluindo a semântica, foi alcançado com êxito.

5.2 Considerações finais

A utilização de conceitos provenientes da lógica deôntica adiciona maior versatilidade à modelação dos sistemas de informação. Através do enriquecimento diagramático e da adição de semântica a novos símbolos gráficos foi possível representar genericamente, no diagrama de classes da UML, dois tipos de situações de utilização real de sistemas de informação que pelo carácter atípico não são possíveis de modelar com o recurso exclusivo a restrições fortes. Através de uma metodologia genérica que implementa um processo automático de geração do modelo de dados, foi possível generalizar os procedimentos a aplicar a estes casos e assim possibilitar a criação dos respectivos modelos relacionais, de forma automática. O processo referido contribui para o aperfeiçoamento das ferramentas de geração automática de bases de dados; para o enriquecimento da documentação gráfica dos sistemas de informação, e para a melhoria da comunicação e compreensão colectivas dos sistemas de informação, dentro e fora das organizações.

5.3 Sugestões para investigações futuras

No presente estudo a proposta da arquitectura das tabelas adicionais, cuja função se descreve como repositório da informação de Restrições “Contrary to duties” pode ser melhorado, se necessário, para incluir a caracterização de novas restrições “Contrary to duties”, ou em alternativa serem retirados alguns dos atributos que se verifiquem desnecessários.

A implementação do comportamento intrínseco dos conceitos de “Obrigação” e “Proibição” no contexto da formação de restrições “Contrary to duties” poderá ser alvo de uma investigação futura, possibilitando a implementação da violação sucessiva de normas e registo consequente do número de violações correspondente, enriquecendo desta forma as possibilidades de representação gráfica da UML em situações em que as restrições fortes não solucionam os problemas decorrentes da utilização real dos sistemas de informação.

Um estudo mais aprofundado que possibilite generalizar a utilização de restrições "Contrary to duties", incluindo a possibilidade de conjunção, disjunção e negação das mesmas poderá levar à utilização dos conceitos deônticos em funções genéricas de implementação deste tipo de restrições, tais como "Add in" ou "Plug in", a adicionar a software de modelação de software já existente.

A procura, levantamento e documentação de novos casos atípicos que surjam decorrentes do funcionamento em real de sistemas de informação pode ajudar a demonstrar o comportamento genérico do tipo de restrições propostas e servir de alavanca para demonstrar a generalização do seu comportamento num conjunto mais alargado de situações possíveis, por exemplo em casos de conjunção ou disjunção de conceitos deônticos.

O acompanhamento da lógica deôntica, e os seus avanços podem servir para ser adaptados a situações futuras que justifiquem a utilização, por exemplo de novos conceitos deônticos, aplicados à problemática da modelação de sistemas de informação, e que conjuntamente com o levantamento de situações atípicas de utilização real de sistemas de informação possam constituir a base para novas propostas de representação gráfica e/ou semântica.

A OCL, linguagem de modelação utilizada no decorrer do presente estudo pode ser alvo de futuras investigações, em duas vertentes essenciais, a primeira corresponde à análise a efectuar às potencialidades do compilador, já existente, nesta linguagem; a segunda mais orientada para o problema específico da geração automática do modelo relacional, corresponde à verificação da possibilidade de tradução das principais funções da OCL para funções de manipulação dos elementos do diagrama relacional, caso o compilador de OCL não permita fazê-lo. Como a OCL é baseada na lógica matemática, a tradução das funções da OCL para uma linguagem de programação é outra das hipóteses a considerar.

Bibliografia

BOOCH, G.; RUMBAUGH, J.; & JACOBSON, I. 1998. *The Unified Modeling Language User Guide*: Addison-Wesley object technology series.

CARMO, J. ; DEMOLOMBE, R. ; & JONES, A. 2001. *An application of deontic logic to information system constraints*: Fundamenta Informaticae 34 (2001) 1-19 IOS Press.

CARMO, J.; & JONES, A. 2002. *Deontic Logic and Contrary-to-Duties:* Handbook of Philosophical Logic, Second edition, volume 8, D.M. Gabbay e F. Guenthner (eds.), Kluwer Academic Publishers, Dordrecht, Holland, pp. 265-343.

CODD, E.F. 1990. *The relational model for database management.* (version 2): Addison Wesley Publishing Company.

RAMOS, P. 2003. *Extending the UML class diagram with deontic constraints*: In 7th World multiconference on systemics, cybernetics and informatics (SCI'03). USA: Orlando, 2003.

RAMOS, P. 2006 *Desenhar bases de dados com UML.* Lisboa: Edições Sílabo, 2006.

RUMBAUGH, J. ; JACOBSON, I.; & BOOCH, G. 1999. *The Unified Modeling Language Reference Manual.* Addison-Wesley object technology series.

SILVA, A. ; & VIDEIRA, C. 2005. UML - Metodologias e Ferramentas CASE 2ª Edição Volume 1 : Edições Centro Atlântico.

WARMER, J; & KLEPPE, A. 2003. *The object constraint language*: Addison-Wesley object technology series

Anexos

Anexo I Aplicação das regras de mapeamento entre o modelo de classes e o modelo relacional.

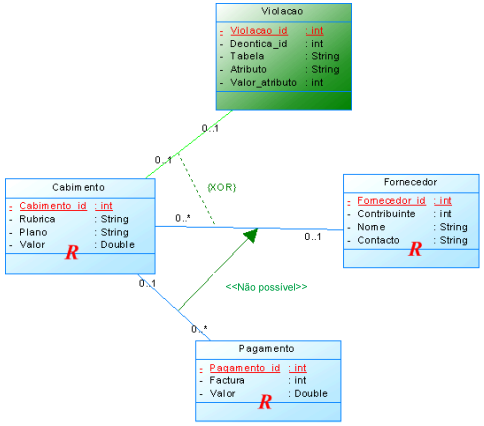
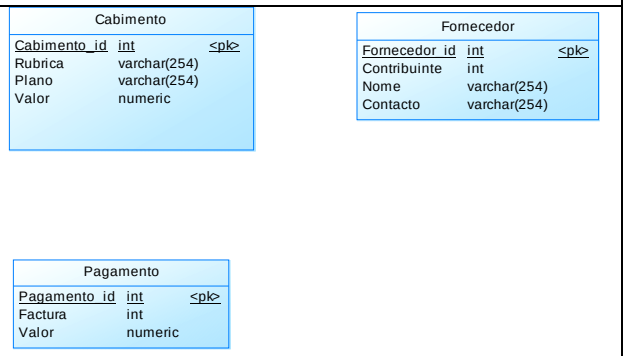
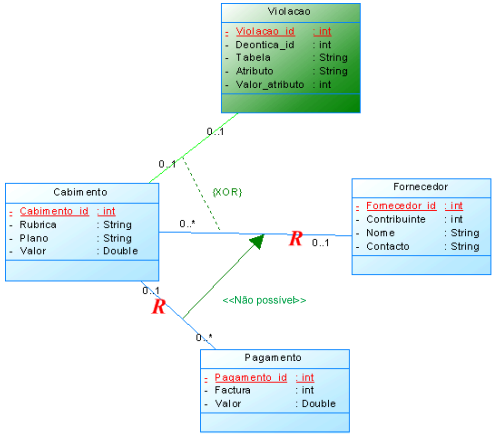
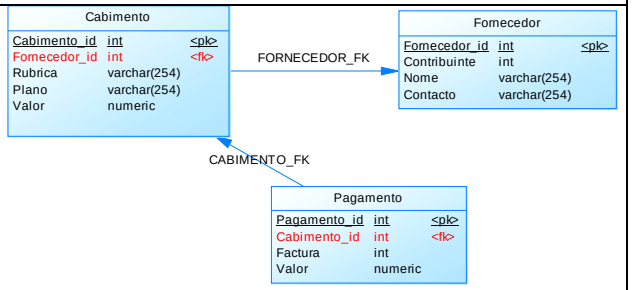
Anexo II Código dos triggers que implementam o modelo relacional.

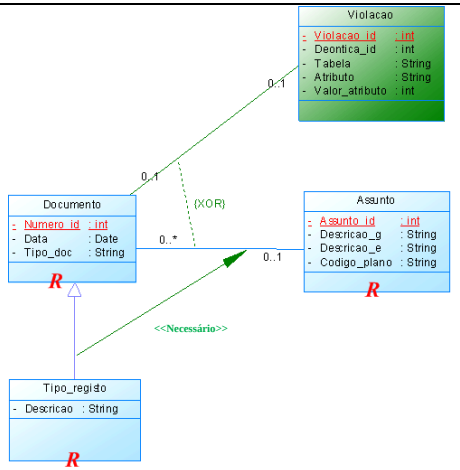
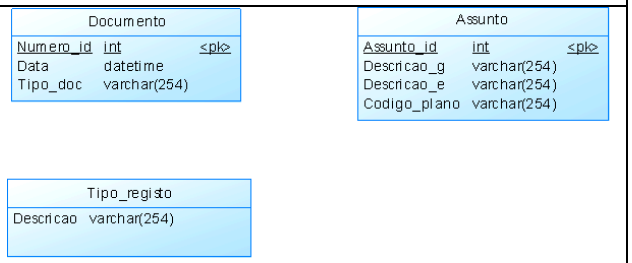
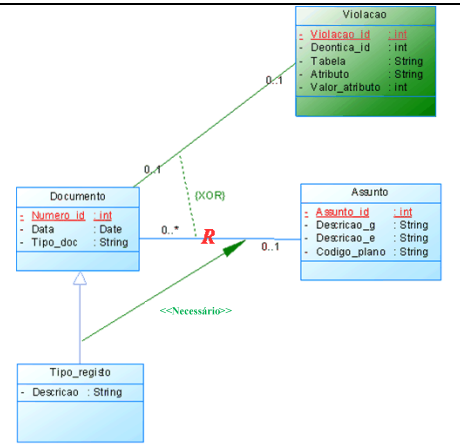
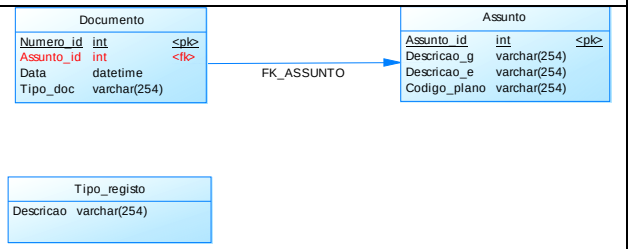
Anexo III Código da aplicação em Prolog para geração automática do modelo relacional.

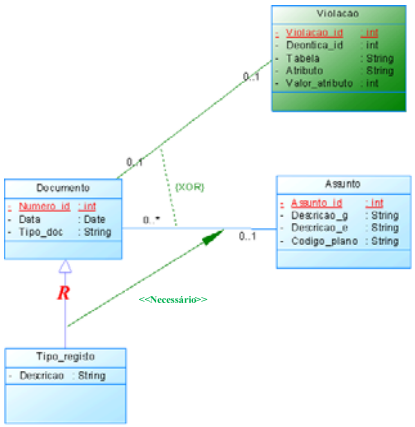
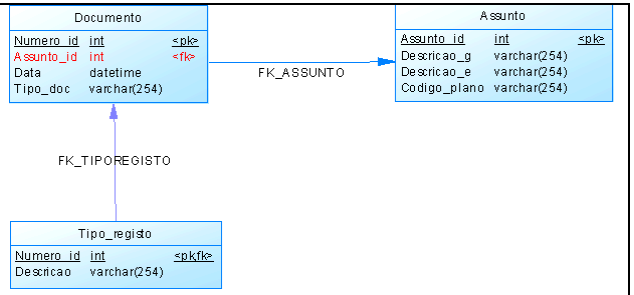
Anexo IV Estrutura dos dados de entrada para a aplicação Prolog.

Anexo V Instruções para execução da aplicação Prolog.

Anexo I Aplicação das regras de mapeamento entre o modelo de classes e o modelo relacional.

Aplicação das regras de mapeamento para o modelo relacional - Sistema de gestão orçamental			
Diagrama de classes	*Regra de mapeamento*	*Descrição*	*Modelo relacional gerado*
	**Regra 1 (classes)** Todas as classes e associações do tipo «muitos para muitos» dão origem a tabelas, e nada mais dá origem a tabelas…; **Regra 2 (Atributos de classes)** Todos os atributos de uma classe são atributos da tabela que implementa a classe;	A aplicação das regras de mapeamento (um e dois) gera no modelo relacional, à direita, as tabelas e campos (a azul.) No diagrama de classes, à esquerda, os elementos marcados com "R", correspondem aos elementos a que se aplica a "regra 1" e "regra 2"	
	**Regra 3 (Atributos de associações)** Todos os atributos de uma associação são atributos da tabela que ou (i) implementa a associação ou (ii) herda as chaves primárias das restantes tabelas que implementam as classes envolvidas na associação…; **Regra 5 (Associações do tipo «um para muitos»)** A tabela cujos registos são susceptíveis de serem relacionados com apenas um registo da outra tabela (lado muitos) herda como chave estrangeira a chave da(s) tabela(s) cuja correspondência é unitária (lado um);	As duas associações marcadas com "R", no diagrama de classes, geram relacionamentos entre tabelas, no modelo relacional. Com a aplicação da "regra 5", são criadas chaves estrangeiras nas tabelas "Cabimento" e "Pagamento".(a vermelho).	

Aplicação das regras de mapeamento para o modelo relacional - Sistema de gestão documental			
Diagrama de classes	*Regra de mapeamento*	*Descrição*	*Modelo relacional gerado*
	**Regra 1 (classes)** Todas as classes e associações do tipo «muitos para muitos» dão origem a tabelas, e nada mais dá origem a tabelas…; **Regra 2 (Atributos de classes)** Todos os atributos de uma classe são atributos da tabela que implementa a classe;	A aplicação das regras de mapeamento (um e dois) gera, no modelo relacional, as tabelas e campos (a azul.) No diagrama de classes, os elementos marcados com “R”, correspondem aos elementos a que se aplica a “regra 1” e “regra 2”	
	**Regra 3 (Atributos de associações)** Todos os atributos de uma associação são atributos da tabela que ou (i) implementa a associação ou (ii) herda as chaves primárias das restantes tabelas que implementam as classes envolvidas na associação…; **Regra 5 (Associações do tipo «um para muitos»)** A tabela cujos registos são susceptíveis de serem relacionados com apenas um registo da outra tabela (lado muitos) herda como chave estrangeira a chave da(s) tabela(s) cuja correspondência é unitária (lado um);	A associação marcada com “R”, no diagrama de classes, gera relacionamentos entre tabelas, no modelo relacional. Com a aplicação da “regra 5”, é criada uma chave estrangeira na tabela “Documento”, (a vermelho).	

	**Regra 8 (Generalizações)** Duas situações distintas podem ocorrer: a) … b) As subclasses só têm identidade enquanto associadas à supraclasse: a chave da tabela que implementa a supraclasses é obtida através dos atributos da própria tabela. As tabelas correspondentes às subclasses herdam a mesma chave da tabela que implementa a supraclasse.	A associação marcada com "R", no diagrama de classes, gera, após aplicação da regra oito, uma chave primária na tabela "TipoRegisto", que corresponde à chave primária da tabela "Documento". Como foi gerada uma tabela que implementa uma subclasse, a chave primária corresponde também a chave estrangeira da supraclasse "Documento".	

Anexo II Código dos triggers que implementam o modelo relacional.

Tabela Cabimento

```
create trigger dbo.cabimento_del on dbo.Cabimento for delete as

BEGIN
   if exists (select * from deleted,Violacao where
   (Violacao.Tabela='Cabimento' and Violacao.Atributo='Cabimento_id' and
   deleted.Cabimento_id=Violacao.Valor_atributo))
  BEGIN
    declare
      @atributo int,
      @errno    int,
      @errmsg   varchar(255)

    set @atributo=(select Cabimento_id from deleted)
    delete from Violacao where
    (Violacao.Tabela='Cabimento' and Violacao.Atributo='Cabimento_id' and
Valor_atributo=@atributo)
  END

  return

error:
    raiserror @errno @errmsg
    rollback  transaction
END

……


create trigger dbo.cabimento_ins on dbo.Cabimento for insert as

if exists (select * from inserted where
inserted.Fornecedor_id=NULL)
BEGIN
  declare
    @d_id   int,
    @ca_id   int,
    @errno   int,
    @errmsg varchar(255)

  set @d_id=(select Deontica_id from dbo.Idealidade where
Relacao_e='FORNECEDOR_FK')
  set @ca_id=(select Cabimento_id from inserted)
  insert into deontica.dbo.Violacao (Deontica_id, Tabela, Atributo, Valor_atributo)
  values (@d_id, 'Cabimento', 'Cabimento_id', @ca_id)
```

```
  return

error:
    raiserror @errno @errmsg
    rollback  transaction
END


……..


create trigger dbo.cabimento_updt on dbo.Cabimento for update as

BEGIN
  declare
    @errno   int,
    @errmsg varchar(255)

  if exists (select * from deleted,inserted where
  deleted.Fornecedor_id=NULL and inserted.Fornecedor_id!=NULL)
  BEGIN
    declare
      @cab_id int

    set @cab_id=(select Cabimento_id from deleted)
    delete from deontica.dbo.Violacao where
    (Violacao.Tabela='Cabimento' and Violacao.Atributo='Cabimento_id' and
Valor_atributo = @cab_id)
  END

  if exists (select * from deleted,inserted where
  deleted.Fornecedor_id!=NULL and inserted.Fornecedor_id=NULL )
  BEGIN
    declare @cab_id2 int,
       @d_id2 int

    if exists (select * from inserted, Pagamento where
    inserted.Cabimento_id=Pagamento.Cabimento_id)
    BEGIN
      print ' Nao e possivel alterar o fornecedor, ja existem pagamentos'
      rollback transaction
      return
    END

    set @d_id2=(select Deontica_id from dbo.Idealidade where
Relacao_e='FORNECEDOR_FK')
    set @cab_id2=(select Cabimento_id from deleted)
    insert into deontica.dbo.Violacao (Deontica_id, Tabela, Atributo, Valor_atributo)
```

```
   values (@d_id2, 'Cabimento', 'Cabimento_id', @cab_id2)
  END

  return

error:
   raiserror @errno @errmsg
   rollback  transaction

END
```

……..

Tabela Documento

```
create trigger dbo.documento_ins on dbo.Documento for insert as

BEGIN
  declare
    @errno   int,
    @errmsg varchar(255)

  if exists (select * from inserted where
  inserted.Assunto_id=NULL)
  BEGIN
    declare @d_id int,
       @docp_id int

    set @d_id=(select Deontica_id from dbo.Idealidade where Relacao_e='FK_ASSUNTO')
    set @docp_id=(select Numero_id from inserted)
    insert into deontica.dbo.Violacao (Deontica_id, Tabela, Atributo, Valor_atributo)
    values (@d_id, 'Documento', 'Numero_id', @docp_id)
    insert into deontica.dbo.TipoRegisto (Numero_id, Descricao) VALUES (@docp_id,
'Pre-registo')
  END

  ELSE
  BEGIN
    declare
      @docv_id int

    set @docv_id=(select Numero_id from inserted)
    insert into deontica.dbo.TipoRegisto (Numero_id, Descricao) VALUES (@docv_id,
'Vivo')
  END

  return
```

```
error:
   raiserror @errno @errmsg
   rollback  transaction
END

……..


create trigger dbo.documento_updt on dbo.Documento for update as

BEGIN
  declare
    @errno   int,
    @errmsg varchar(255)

  if exists (select * from deleted,inserted where
  deleted.Assunto_id=NULL and inserted.Assunto_id!=NULL)
  BEGIN
    declare
      @doc_id int

    set @doc_id=(select Numero_id from deleted)
    delete from deontica.dbo.Violacao WHERE
    (Violacao.Tabela='Documento' and Violacao.Atributo='Numero_id' and Valor_atributo
= @doc_id)
    update deontica.dbo.TipoRegisto SET Descricao = 'Vivo' WHERE
     Numero_id= @doc_id
  END

  if exists (select * from deleted,inserted where
  deleted.Assunto_id!=NULL and inserted.Assunto_id=NULL)
  BEGIN
    declare
      @doc_id2 int,
      @d_id2 int

    set @d_id2=(select Deontica_id from dbo.Idealidade where
Relacao_e='FK_ASSUNTO')
    set @doc_id2=(select Numero_id from deleted)
    insert into deontica.dbo.Violacao (Deontica_id, Tabela, Atributo, Valor_atributo)
    values (@d_id2, 'Documento', 'Numero_id', @doc_id2)
    update deontica.dbo.TipoRegisto SET Descricao = 'Pre-registo' WHERE
     Numero_id = @doc_id2
  END

return
```

```
error:
   raiserror @errno @errmsg
   rollback  transaction
END
```

……..

Tabela Pagamento

```
create trigger dbo.pagamento_ins on dbo.Pagamento for insert, update as

if exists (select * from inserted,Violacao where
(Violacao.Tabela='Cabimento' and Violacao.Atributo='Cabimento_id' and
inserted.Cabimento_id=Violacao.Valor_atributo))
BEGIN
  declare @viola_id int,
      @ideal_id char

  set @viola_id = (select Violacao.Deontica_id from inserted,Violacao where
  (Violacao.Tabela='Cabimento' and Violacao.Atributo='Cabimento_id' and
inserted.Cabimento_id=Violacao.Valor_atributo))
  set @ideal_id =(select Idealidade.Idealidade from Idealidade where
  @viola_id=Idealidade.Relacao_d)
  if @ideal_id='f'
  BEGIN
    rollback transaction
    print ' Proibido inserir pagamentos, nao existe codigo de fornecedor'
  END
END
```

…….

Tabela TipoRegisto

```
create trigger dbo.tiporegisto_del on dbo.TipoRegisto for delete as

BEGIN
  declare
    @reg int,
    @errno   int,
    @errmsg varchar(255)

  set @reg=(select Numero_id from deleted)
  delete from dbo.Documento WHERE
  Documento.Numero_id=@reg
  print 'tipo registo apagado'
```

```
  if exists (select * from deleted,Violacao where
  (Violacao.Tabela='Documento' and Violacao.Atributo='Numero_id' and
deleted.Numero_id=Violacao.Valor_atributo))
  BEGIN
    declare
      @atributo int

    set @atributo=(select Numero_id from deleted)
    delete from Violacao where
    (Violacao.Tabela='Documento' and Violacao.Atributo='Numero_id' and
Valor_atributo=@atributo)
  END

return

error:
    raiserror @errno @errmsg
    rollback  transaction
END

……


create trigger dbo.tiporegisto_updt on TipoRegisto for insert, update as

BEGIN
  declare
    @errno   int,
    @errmsg varchar(255)

  if exists (select * from inserted,Violacao where
  Violacao.Tabela='Documento' and Violacao.Atributo='Numero_id' and
inserted.Numero_id=Violacao.Valor_atributo)
  BEGIN
    declare
      @num_id  int

    set @num_id=( select Numero_id from inserted)
    update deontica.dbo.TipoRegisto set Descricao = 'Pré-registo' where
    Numero_id =@num_id
  END

  ELSE
  BEGIN
    declare
      @num_id2  int

    set @num_id2=( select Numero_id from inserted)
```

```
   update deontica.dbo.TipoRegisto SET Descricao = 'Vivo' WHERE Numero_id
=@num_id2
  END

return

error:
   raiserror @errno @errmsg
   rollback  transaction
END
……
```

Anexo III Código da aplicação em Prolog para geração automática do modelo relacional.

Ficheiro de entrada de dados (sistemas de gestão documental e orçamental)

```
class(cabimento).
class(fornecedor).
class(pagamento).
class(documento).
class(assunto).
class(tipo_registo).

attribute(cabimento,cabimento_id,int,no,yes).
attribute(cabimento,rubrica,string,yes,no).
attribute(cabimento,plano,string,yes,no).
attribute(cabimento,valor,numeric,yes,no).
attribute(fornecedor,fornecedor_id,int,no,yes).
attribute(fornecedor,contribuinte,int,yes,no).
attribute(fornecedor,nome,string,yes,no).
attribute(fornecedor,contacto,string,yes,no).
attribute(pagamento,pagamento_id,int,no,yes).
attribute(pagamento,factura,int,yes,no).
attribute(pagamento,valor,numeric,yes,no).
attribute(documento,numero_id,int,no,yes).
attribute(documento,data,date,yes,no).
attribute(documento,tipo_doc,string,yes,no).
attribute(assunto,assunto_id,int,no,yes).
attribute(assunto,descricao_g,string,yes,no).
attribute(assunto,descricao_e,string,yes,no).
attribute(assunto,codigo_plano,string,yes,no).
attribute(tipo_registo,descricao,string,yes,no).

attributeID(cabimento,cabimento_id).
attributeID(fornecedor,fornecedor_id).
attributeID(pagamento,pagamento_id).
attributeID(documento,numero_id).
attributeID(assunto,assunto_id).

association(fornecedor_fk).
association(cabimento_fk).
association(fk_assunto).

associationMember(fornecedor_fk,fornecedor,fornecedor,0,1).
associationMember(fornecedor_fk,cabimento,cabimento,0,*).
associationMember(cabimento_fk,cabimento,cabimento,0,1).
associationMember(cabimento_fk,pagamento,pagamento,0,*).
associationMember(fk_assunto,documento,documento,0,*).
associationMember(fk_assunto,assunto,assunto,0,1).

generalization(documento,1).
specification(documento,tipo_registo).
registration(pré-registo,vivo).

ideality(1,cabimento,fornecedor,fornecedor_fk,i).
ideality(2,documento,assunto,fk_assunto,i).
restrition(3,documento,tipo_registo,fk_tiporegisto,o,2).
restrition(4,cabimento,pagamento,cabimento_fk,f,1).
```

Ficheiro de código em Prolog

```
% Needed to assert and retract

:-dynamic association/1.
:-dynamic associationMember/5.
:-dynamic attributeID/2.
:-dynamic attribute/5.
:-dynamic aggregation/3.
:-dynamic aggregationPart/4.



% Load database
:-consult(base2).

%AXIOMAS de MANIPULACAO  (eliminar classes associativas que nao são n-%n
e agregações)
  map:-    (not(axiomaM2)),
      (not(axiomaM3)),
      (not(axiomaM4)),
      (not(axiomaM5)),
      (not(axiomaM6)),
      (not(axiomaM7)),
      (not(axiomaM8)),
      (not(axiomaM9)),
      (not(axiomaM10)),
      (not(axiomaM11)).




  %Axioma M1
  associativeClass(X):-class(X),association(X).

  %associacoes  "um" para "mais do que um"
  %Axioma M2
  axiomaM2:-  associativeClass(A),
      associationMember(A,X,R,LM,UM),
      sup1(UM),
          associationMember(A,Y,R1,LM1,1),
           associationMember(B,A,R2,LM2,UM2),
      print('passou2'),nl,
             assert(associationMember(B,X,R2,LM2,UM2)),
           retract(associationMember(B,A,R2,LM2,UM2)),
      fail.

  %Axioma M3
  axiomaM3:-  associativeClass(A),
      associationMember(A,X,R,LM,UM),
      sup1(UM),
          associationMember(A,Y,R1,LM1,1),
           attribute(A,At,Ty,Nl,Uq),
      print('passou3'),nl,
          assert(attribute(X,At,Ty,Nl,Uq)),
```

```
          retract(attribute(A,At,Ty,Nl,Uq)),
      fail.


%associacoes em que um dos limites inferiores é 0 e todos os limites
%superiores não são >1
  %Axioma M4
  axiomaM4:-  associativeClass(A),
       not(existMembersMultUm_sup1(A)),
          associationMember(A,Y,R1,0,UM1),
       associationMember(B,A,R2,LM2,UM2),
      print('passou4'),nl,
            assert(associationMember(B,Y,R2,LM2,UM2)),
          retract(associationMember(B,A,R2,LM2,UM2)),
      fail.

  %Axioma M5
  axiomaM5:-  associativeClass(A),
       not(existMembersMultUm_sup1(A)),
          associationMember(A,Y,R1,0,UM1),
           attribute(A,At,Ty,Nl,Uq),
      print('passou5'),nl,
            assert(attribute(Y,At,Ty,Nl,Uq)),
          retract(attribute(A,At,Ty,Nl,Uq)),
      fail.

      %associacoes em que nenhum limite inferiores é 0 e todos os
      %limites superiores não são >1
  %Axioma M6
  axiomaM6:-  associativeClass(A),
       not(existMembersMultUm_sup1(A)),
       not(existMembersMultLm_0(A)),
          associationMember(A,Y,R1,LM1,UM1),
       associationMember(B,A,R2,LM2,UM2),
      print('passou6'),nl,
            assert(associationMember(B,Y,R2,LM2,UM2)),
          retract(associationMember(B,A,R2,LM2,UM2)),
      fail.

  %Axioma M7
  axiomaM7:-  associativeClass(A),
       not(existMembersMultUm_sup1(A)),
       not(existMembersMultLm_0(A)),
          associationMember(A,Y,R1,LM1,UM1),
      attribute(A,At,Ty,Nl,Uq),
      print('passou7'),nl,
            assert(attribute(Y,At,Ty,Nl,Uq)),
          retract(attribute(A,At,Ty,Nl,Uq)),
      fail.

      %quando a associação é >1 para >1 os atributosID dos argumentos
      %passam como atributosID da classe associativa
  % é apenas para facilitar mais adiante
  %Axioma M8
  axiomaM8:-  associativeClass(A),
       not(existMembersMultUm_1(A)),
          associationMember(A,Y,R1,LM1,UM1),
      attribute(Y,At,Ty,Nl,Uq),
      attributeID(Y,At),
```

```
      print('passou8'),nl,
             assert(attribute(A,At,Ty,Nl,Uq)),
             assert(attributeID(A,At)),
      fail.


  %Agregação
  %Axioma M9
  axiomaM9:-  aggregation(A,Lm,Um),
      aggregationPart(A,B,Lm1,Um1),
      cat([B,'_',A],Whole,Join),
      print('passou9'),nl,
      assert(association(Whole)),
      assert(associationMember(Whole,A,A,Lm,Um)),
      assert(associationMember(Whole,B,B,Lm1,Um1)),
            retract(aggregation(A,Lm,Um)),
      retract(aggregationPart(A,B,Lm1,Um1)),
      fail.

  %COMPOSICAO
  axiomaM10:-  composition(A),
      compositionPart(A,B,LM,UM),
      attributeID(A,X),
      attribute(A,X,Ty,Nl,Uq),
      assert(attributeID(B,X)),
      assert(attribute(B,X,Ty,no,no)),fail.

  %GENERALIZACAO
      axiomaM11:-generalization(G,1),
      specification(G,S),
      attributeID(G,Y),
      not(attributeID(S,ID)),
      attribute(G,Y,Ty,Nl,Uq),
      assert(attributeID(S,Y)),
      assert(attribute(S,Y,Ty,no,yes)),fail.



% Regras de Transposicao

%REGRA T1
%Classes e Associacoes n-n dão origem a tabelas

  table(A):- class(A),not associativeClass(A).
  table(A):- association(A), allsup1(A).


%REGRA T2
%Colunas das classes são atributos das tabelas correspondentes

  column(X,At,Ty,Nl,Uq):- table(X), class(X), attribute(X,At,Ty,Nl,Uq).

%REGRA T3
%Atributos que fazem parte do ID da classe vao para a chave da tabela

  key(X,A):-table(X), attributeID(X,A).


%REGRA T4
```

```
%atributos das associacoes que deram origem a tabelas passam a colunas
%da tabela correspondente
%e cria a chave estrangeira

  foreignKey(R1,A,X):-   class(X),
          association(A),
          table(A),
          associationMember(A,X,R1,LM,UM).

  fkMember(R1,At):-   class(X),
          association(A),
          table(A),
          attributeID(X,At),
          associationMember(A,X,R1,LM,UM).

  column(A,W,Ty,no,no):-  class(X),
          association(A),
          table(A),
          associationMember(A,X,R1,LM,UM),
          attributeID(X,W),
          attribute(X,W,Ty,Nl,U).

  key(A,W):-      class(X),
          association(A),
          table(A),
          associationMember(A,X,R1,LM,UM),
          attributeID(X,W),
          attribute(X,W,Ty,Nl,U).



%REGRA 5
% Transposicao de 1-n e 1-1

  % pelo menos uma das associações é 1 ..1
  foreignKey(R1,Y,X):-  association(A),
          not class(A),
          associationMember(A,X,R1,1,1),
               associationMember(A,Y,R2,LM2,UM2),
          R2\=R1.

  %tabela filha com 0..1 ou 1..1 ou 0 ...* ou 1 ...*

  fkMember(R1,W):-  association(A),
        not class(A),
        associationMember(A,X,R1,1,1),
             associationMember(A,Y,R2,LM2,1),
        R1\=R2,
        attributeID(X,W),
        attribute(X,W,T,N,U).

  %tabela filha com 0..1 ou 1..1
  column(Y,W,Ty,no,no):-  association(A),
          not class(A),
          associationMember(A,X,R1,1,1),
               associationMember(A,Y,R2,LM2,1),
          R1\=R2,
          attributeID(X,W),
```

```
        attribute(X,W,Ty,Nl,U).

%tabela filha com 0 ...* ou 1 ...*
column(Y,W,Ty,no,yes):-  association(A),
        not class(A),
        associationMember(A,X,R1,1,1),
             associationMember(A,Y,R2,LM2,UM2),
        R1\=R2,
        sup1(UM2),
        attributeID(X,W),
        attribute(X,W,Ty,Nl,U).


% nenhuma associações é 1 ..1 mas existe uma 0 ...1

foreignKey(R2,Z,Y):-  association(A),
        not class(A),
        findall((A),(associationMember(A,_,_,1,1)), []),
        %not(associationMember(A,X,RX,1,1)),
        associationMember(A,Y,R2,0,1),
             associationMember(A,Z,R3,LM3,UM3),
        R3\=R2.

%tabela filha com 0..1 ou 0..* ou 1..*

fkMember(R2,W):-  association(As),
      not class(As),
      findall((A),(associationMember(As,_,_,1,1)), []),
      %not(associationMember(As,X,RX,1,1)),
      associationMember(As,Y,R2,0,1),
           associationMember(As,Z,R3,LM3,UM3),
      R3\=R2,
      attributeID(Y,W),
      attribute(Y,W,T,N,U).

%tabela filha com 0..1
column(Z,W,T,yes,no):-  association(As),
        not class(As),
        %not(associationMember(As,X,RX,1,1)),
        findall((A),(associationMember(As,_,_,1,1)), []),
        associationMember(As,Y,R2,0,1),
             associationMember(As,Z,R3,LM3,1),
        R3\=R2,
        attributeID(Y,W),
        attribute(Y,W,T,N,U).

%tabela filha com 0..* ou 1..*
column(Z,W,T,yes,yes):-  association(As),
        not class(As),
        %not(associationMember(As,X,RX,1,1)),
        findall((A),(associationMember(As,_,_,1,1)), []),
        associationMember(As,Y,R2,0,1),
             associationMember(As,Z,R3,LM3,UM3),
        R3\=R2,
        sup1(UM3),
        attributeID(Y,W),
        attribute(Y,W,T,N,U).
```

```
  %tabela filha com 0..1, 1..1 e 1..* ou 0..*
  foreignKey(R2,Z,Y):-  association(A),
          not class(A),
          associationMember(A,X,R1,1,1),
          associationMember(A,Y,R2,0,1),
               associationMember(A,Z,R3,LM3,UM3),
          R3\=R2,
          sup1(UM3).

  fkMember(R2,W):-  association(A),
        not class(A),
        associationMember(A,X,R1,1,1),
        associationMember(A,Y,R2,0,1),
             associationMember(A,Z,R3,LM3,UM3),
        R3\=R2,
        sup1(UM3),
        attributeID(Y,W),
        attribute(Y,W,T,N,U).

  column(Z,W,T,yes,yes):-  association(A),
          not class(A),
          associationMember(A,X,R1,1,1),
           associationMember(A,Y,R2,0,1),
               associationMember(A,Z,R3,LM3,UM3),
          R3\=R2,
          sup1(UM3),
          attributeID(Y,W),
          attribute(Y,W,T,N,U).



%REGRA T6 - generalizacao

    foreignKey(G,S,G):-  generalization(G,1),
          specification(G,S).
          %cat([S,'_',G,'_FK'],Whole,Join).

      fkMember(G,Y):-  generalization(G,1),
          specification(G,S),
          %cat([S,'_',G,'_FK'],Whole,Join),
          attributeID(G,Y),
          attribute(G,Y,Ty,Nl,Uq).

      column(S,Y,Ty,no,yes):-  generalization(G,1),
          specification(G,S),
          %cat([S,'_',G,'_FK'],Whole,Join),
          attributeID(G,Y),
          attribute(G,Y,Ty,Nl,Uq).

      key(S,Y):-  generalization(G,1),
      specification(G,S),
      attributeID(G,Y),
      not(hasAttributeID(S)),
      not(attributeID(S,ID)),
      attribute(G,Y,Ty,Nl,Uq).


% Predicados auxiliares
```

```
%verifica se um valor é *

equal*(*).

%verifica se valor é superior >1

sup1(*).
sup1(N):- not(equal*(N)), N>1.

%verifica se conjunto contem valor superior >1

contains*([*|_]).

contains*([N|_]):- not(equal*(N)),N>1.

contains*([_|Tail]):- contains*(Tail).

existMembersMultUm_sup1(A):-
findall((UM),(associationMember(A,Y,R,LM,UM)), Set),
                        contains*(Set).

%verifica se conjunto contem valor inferior a 2

contains1([*|_]):- fail.

contains1([N|_]):- not(equal*(N)), N<2.

contains1([_|Tail]):- contains1(Tail).

existMembersMultUm_1(A):- findall((UM),
         (associationMember(A,Y,R,LM,UM)), Set),
                     contains1(Set).

allsup1(A):- findall((UM),
     (associationMember(A,Y,R,LM,UM)), Set),
             not contains1(Set).

hasAttributeID(Table):- attributeID(Table,ID).


relacional:- tell('tsql.txt'),
     not(printrelacional),
     not(printchaveestrangeira(X)),nl,nl,
     not(printcomplementares),nl,nl,
     not(restricao),
     not(triggers),
     told.

printrelacional:- table(X),nl,
      print('create table dbo.'),
      print(X),
      print('('),nl,
      not(printcolunas(X)),
      print(')'),nl,nl,
       not(printchave(X)),nl,nl,
      fail.
```

```
printchave(Tab):- key(Tab,A),
      print('alter table dbo.'),
      print(Tab),nl,
      print('add constraint '),
      print('pk_'),
      print(Tab),
      print(' primary key nonclustered ('),
      print(A),
      print(')'),
      print(' on "default"'),
      fail.

printchaveestrangeira(Tab):-  foreignKey(R2,T,Y),
          T=Tab,
          print('alter table dbo.'),
          print(T),print(' '),nl,
          print('add constraint '),
          print(T),
          print('_fk foreign key ('),
          not(printchaveestrangeiramember(R2)),
          print(')'),nl,
          print('references dbo.'),
          print(R2),
          print(' ('),
          not(printchaveestrangeiramember(R2)),
          print(')'),nl,nl,nl,
          fail.

printchaveestrangeiramember(Chave):-  fkMember(C,X),
              C=Chave,
              print(X),
              print(' ' ),
              fail.


printcolunas(Tab):-  column(Tb,C,T,N,U),
        Tb=Tab,
        tab(2),
        fwrite(a,20,0,C),
        tab(2),
        (T='string'->fwrite(a,20,0,'varchar(254)');
        T='double'->fwrite(a,20,0,'numeric');
        T='date'->fwrite(a,20,0,'datetime');fwrite(a,20,0,T)),
        tab(2),
        (N='yes'->fwrite(a,20,0,'null');fwrite(a,20,0,'not null')),
        print(','),
        nl,
        fail.


      %criação de tabelas complementares

printcomplementares:-   print('create table dbo.idealidade ('),nl,
        print('  deontica_id          integer                        not
null,'),nl,
        print('  tabela_um            varchar(254)
null,'),nl,
```

```
        print('     tabela_dois          varchar(254)
null,'),nl,
        print('     tabela_p             varchar(254)
null,'),nl,
        print('     relacao_e            varchar(254)
null,'),nl,
        print('     idealidade           char(1)
null,'),nl,
        print('     relacao_d            integer
null'),nl,
        print(')'),nl,nl,
        print('alter table dbo.idealidade'),nl,
          print('add constraint pk_idealidade primary key nonclustered
(deontica_id)'),nl,
           print('on "default"'),nl,nl,
        print('create table dbo.violacao ('),nl,
           print('  violacao_id          int
not null,'),nl,
           print('  deontica_id          integer
null,'),nl,
           print('  tabela               varchar(254)
null,'),nl,
           print('  atributo             varchar(254)
null,'),nl,
           print('  valor_atributo       int
null'),nl,
        print(')'),nl,nl,
        print('alter table dbo.violacao'),nl,
           print('add constraint pk_violacao primary key nonclustered
(violacao_id)'),nl,
           print('on "default"'),nl,nl,
        fail.


%Registos da tabela idealidade, com informação das restrições %deônticas

restricao:-      ideality(Cni,Classe1,Classe2,Relacao_i,Idealidade),
        restrition(Cnr,Classe1,Classe3,Relacao_d,Restricao,Crel),
        print('INSERT INTO dbo.idealidade ('),

print('deontica_id,tabela_um,tabela_dois,tabela_p,relacao_e,idealidade)')
,nl,
        print('VALUES ('),
        print(Cni),print(','),print('"'),
        print(Classe1),print('"'),print(','),print('"'),
        print(Classe2),print('"'),print(','),print('"'),
        print(Classe1),print('"'),print(','),print('"'),
        print(Relacao_i),print('"'),print(','),print('"'),
        print(Idealidade),print('"'),
        print(')'),nl,nl,nl,
        print('INSERT INTO dbo.idealidade ('),

print('deontica_id,tabela_um,tabela_dois,tabela_p,relacao_e,idealidade,re
lacao_d)'),nl,
        print('VALUES ('),
        print(Cnr),print(','),print('"'),
        print(Classe1),print('"'),print(','),print('"'),
        print(Classe3),print('"'),print(','),print('"'),
        print(Classe1),print('"'),print(','),print('"'),
```

```
        print(Relacao_d),print('"'),print(','),print('"'),
        print(Restricao),print('"'),print(','),
        print(Crel),
        print(')'),nl,
        print('go'),nl,nl,nl,
         fail.

      %Triggers que implementam o comportamento esperado.


triggers:-      ideality(Cni,Classe1,Classe2,Relacao_i,Idealidade),
        restrition(Cnr,Classe3,Classe4,Relacao_d,Restricao,Cni),
        attributeID(Classe1,Pkey1),
        attributeID(Classe2,Pkey2),

%Triggers que implementam o comportamento a restrição contrary to %duties
"Proibição".

        (Idealidade='i',Restricao='f')->(

          %Trigger que implementa o comportamento do sistema, quando é
apagado um registo na tabela a
          %que corresponde

          print('create trigger dbo.'),
          print(Classe1),
          print('_delete on dbo.'),
          print(Classe1),
          print(' for delete as'),nl,nl,
          print('BEGIN'),nl,
          print('  if exists (select * from deleted,violacao where '),nl,
          print('  (violacao.tabela='),print('"'),
          print(Classe1),print('"'),
          print(' and violacao.atributo='),print('"'),
          print(Pkey1),print('"'),
          print(' and deleted.'),
          print(Pkey1),
          print('=violacao.valor_atributo))'),nl,
          print('  BEGIN'),nl,
          print('    declare'),nl,
          print('      @atributo int,'),nl,
          print('      @errno    int,'),nl,
          print('      @errmsg   varchar(255)'),nl,nl,
          print('    set @atributo=(select '),
          print(Pkey1),
          print(' from deleted)'),nl,
          print('    delete from violacao where '),nl,
          print('    (violacao.tabela='),print('"'),
          print(Classe1),print('"'),
          print(' and violacao.atributo='),print('"'),
          print(Pkey1),print('"'),
          print(' and valor_atributo=@atributo)'),nl,
          print('  END'),nl,nl,
          print('  return'),nl,nl,
          print('error:'),nl,
          print('  raiserror @errno @errmsg'),nl,
              print('  rollback  transaction'),nl,
          print('END'),nl,
          print('go'),nl,nl,nl,nl,
```

```
          %Trigger que implementa o comportamento do sistema, quando é
inserido um registo na  tabela
          %a que corresponde.

          print('create trigger dbo.'),
          print(Classe1),
          print('_insert on dbo.'),
          print(Classe1),
          print(' for insert as'),nl,nl,
          print('if exists (select * from inserted where '),nl,
          print('inserted.'),
          print(Pkey2),
          print('=NULL)'),nl,nl,
          print('BEGIN'),nl,
          print('  declare'),nl,
          print('    @d_id int,'),nl,
          print('    @ca_id int,'),nl,
          print('    @errno    int,'),nl,
          print('    @errmsg   varchar(255)'),nl,nl,
          print('  set @d_id=(select deontica_id from dbo.idealidade
where relacao_e="'),
          print(Relacao_i),
          print('")'),nl,
          print('  set @ca_id=(select '),
          print(Pkey1),
          print(' from inserted)'),nl,
          print('  insert into dbo.violacao (deontica_id, tabela,
atributo, valor_atributo)'),nl,
          print('  values (@d_id, '),
          print('"'),
          print(Classe1),
          print('","'),
          print(Pkey1),
          print('",'),
          print('@ca_id)'),nl,nl,
          print('  return'),nl,nl,
          print('error:'),nl,
          print('  raiserror @errno @errmsg'),nl,
              print('  rollback  transaction'),nl,
          print('END'),nl,
          print('go'),nl,nl,nl,nl,


          %Trigger que implementa o comportamento do sistema, quando é
alterado um registo na tabela
          %a que corresponde.

          print('create trigger dbo.'),
          print(Classe1),
          print('_update on dbo.'),
          print(Classe1),
          print(' for update as'),nl,nl,
          print('BEGIN'),nl,
          print('  declare'),nl,
          print('    @errno    int,'),nl,
          print('    @errmsg   varchar(255)'),nl,nl,
          print('  if exists (select * from deleted,inserted where '),nl,
          print('  deleted.'),
```

```
          print(Pkey2),
          print('=NULL and inserted.'),
          print(Pkey2),
          print('!=NULL)'),nl,
          print('  BEGIN'),nl,
          print('    declare'),nl,
          print('      @cab_id int'),nl,nl,
          print('    set @cab_id=(select '),
          print(Pkey1),
          print(' from deleted)'),nl,
          print('    delete from violacao where '),nl,
          print('    (violacao.tabela='),
          print('"'),
          print(Classe1),
          print('" and violacao.atributo="'),
          print(Pkey1),
          print('" and valor_atributo=@cab_id)'),nl,
          print('  END'),nl,nl,
          print('  if exists (select * from deleted,inserted where '),nl,
          print('  deleted.'),
          print(Pkey2),
          print('!=NULL and inserted.'),
          print(Pkey2),
          print('=NULL)'),nl,
          print('  BEGIN'),nl,
          print('    declare'),nl,
          print('      @cab_id2 int,'),nl,
          print('      @d_id2 int'),nl,nl,
          print('    if exists (select * from inserted,'),
          print(Classe4),
          print(' where '),nl,
          print('    inserted.'),
          print(Pkey1),
          print('='),
          print(Classe4),
          print('.'),
          print(Pkey1),
          print(')'),nl,
          print('    BEGIN'),nl,
          print('      print " Nao e possivel alterar o fornecedor, ja
existem pagamentos"'),nl,
          print('      rollback transaction'),nl,
          print('      return'),nl,
          print('    END'),nl,nl,
          print('    set @d_id2=(select deontica_id from dbo.idealidade
where relacao_e="'),
          print(Relacao_i),
          print('"'),
          print(')'),nl,
          print('    set @cab_id2=(select '),
          print(Pkey1),
          print(' from deleted)'),nl,
          print('    insert into dbo.violacao (deontica_id, tabela,
atributo, valor_atributo)'),nl,
          print('    values (@d_id2,'),
          print('"'),
          print(Classe1),
          print('",'),
          print('"'),
```

```
          print(Pkey1),
          print('",'),
          print('@cab_id2)'),nl,
          print('  END'),nl,nl,
          print('  return'),nl,nl,nl,
          print('error:'),nl,
          print('  raiserror @errno @errmsg'),nl,
              print('  rollback  transaction'),nl,nl,
          print('END'),nl,
          print('go'),nl,nl,nl,nl,

          %Trigger que implementa o comportamento do sistema, quando é
inserido ou alterado um registo na tabela
          %a que corresponde.

          print('create trigger dbo.'),
          print(Classe4),
          print('_insert_up on dbo.'),
          print(Classe4),
          print(' for insert,update as'),nl,nl,
          print('if exists (select * from inserted,violacao where '),nl,
          print('(violacao.tabela="'),
          print(Classe1),
          print('" and violacao.atributo="'),
          print(Pkey1),nl,
          print('" and inserted.'),
          print(Pkey1),
          print('=violacao.valor_atributo))'),nl,
          print('BEGIN'),nl,
          print('  declare'),nl,
          print('    @viola_id int,'),nl,
          print('    @ideal_id char,'),nl,
          print('    @errno    int,'),nl,
          print('    @errmsg   varchar(255)'),nl,nl,
          print('  set @viola_id = (select violacao.deontica_id from
inserted,violacao where '),nl,
          print('  (violacao.tabela="'),
          print(Classe1),
          print('" and violacao.atributo="'),
          print(Pkey1),
          print('" and '),nl,
          print('  inserted.'),
          print(Pkey1),
          print('=violacao.valor_atributo))'),nl,
          print('  set @ideal_id =(select idealidade.idealidade from
idealidade where'),nl,
          print('  @viola_id=idealidade.relacao_d)'),nl,
          print('  if @ideal_id="f"'),nl,
          print('  BEGIN'),nl,
          print('    rollback transaction'),nl,
          print('    print "Proibido inserir pagamentos, não existe
código fornecedor"'),nl,
          print('  END'),nl,
          print('  return'),nl,nl,nl,
          print('error:'),nl,
          print('  raiserror @errno @errmsg'),nl,
              print('  rollback  transaction'),nl,nl,
          print('END'),nl,
          print('go'),nl,nl,nl,nl,fail).
```

```
triggers:-      ideality(Cni,Classe1,Classe2,Relacao_i,Idealidade),
        restrition(Cnr,Classe3,Classe4,Relacao_d,Restricao,Cni),
        attributeID(Classe1,Pkey1),
        attributeID(Classe2,Pkey2),
        registration(Viol,Nviol),

        %Triggers que implementam o comportamento da restrição contrary
to duties "Obrigação".

        (Idealidade='i',Restricao='o')->(

          %Trigger que implementa o comportamento do sistema, quando
          %é inserido um registo na  tabela a que corresponde.

          print('create trigger dbo.'),
          print(Classe1),
          print('_insert on dbo.'),
          print(Classe1),
          print(' for insert as'),nl,nl,
          print('BEGIN'),nl,
          print('  declare'),nl,
          print('    @errno    int,'),nl,
          print('    @errmsg   varchar(255)'),nl,nl,
          print('if exists (select * from inserted where '),nl,
          print('inserted.'),
          print(Pkey2),
          print('=NULL)'),nl,
          print('  BEGIN'),nl,
          print('    declare'),nl,
          print('      @d_id    int,'),nl,
          print('      @docp_id int'),nl,nl,
          print('    set @d_id = (select deontica_id from dbo.idealidade
where '),nl,
          print('    relacao_e="'),
          print(Relacao_i),
          print('")'),nl,
          print('    set @docp_id=(select '),
          print(Pkey1),
          print(' from inserted)'),nl,
          print('    insert into
dbo.violacao(deontica_id,tabela,atributo,valor_atributo)'),nl,
          print('    values (@d_id,"'),
          print(Classe1),
          print('","'),
          print(Pkey1),
          print('",@docp_id)'),nl,
          print('    insert into dbo.'),
          print(Classe4),
          print('('),
          print(Pkey1),
          print(', descricao)'),nl,
          print('     VALUES (@docp_id,"'),
          print(Viol),
          print('")'),nl,
          print('  END'),nl,nl,
          print('  else'),nl,
```

```
print('  BEGIN'),nl,
print('    declare'),nl,
print('      @docv_id int'),nl,nl,
print('    set @docv_id=(select '),
print(Pkey1),
print(' from inserted)'),nl,
print('    insert into dbo.'),
print(Classe4),
print('('),
print(Pkey1),
print(',descricao) VALUES (@docv_id,"'),
print(Nviol),
print('")'),nl,
print('  END'),nl,
print('  return'),nl,nl,nl,
print('error:'),nl,
print('  raiserror @errno @errmsg'),nl,
    print('  rollback  transaction'),nl,nl,
print('END'),nl,
print('go'),nl,nl,nl,nl,

%Trigger que implementa o comportamento do sistema, quando
%é alterado um registo na  tabela a que corresponde.

print('create trigger dbo.'),
print(Classe1),
print('_update on dbo.'),
print(Classe1),
print(' for update as'),nl,nl,
print('BEGIN'),nl,
print('  declare'),nl,
print('    @errno    int,'),nl,
print('    @errmsg   varchar(255)'),nl,nl,
print('  if exists (select * from deleted,inserted where '),nl,
print('  deleted.'),
print(Pkey2),
print('=NULL'),
print(' and inserted.'),
print(Pkey2),
print('!=NULL)'),nl,
print('  BEGIN'),nl,
print('    declare'),nl,
print('      @doc_id    int'),nl,nl,
print('    set @doc_id = (select '),
print(Pkey1),
print(' from deleted)'),nl,
print('    delete from dbo.violacao WHERE'),nl,
print('    (violacao.tabela="'),
print(Classe1),
print('" and violacao.atributo="'),
print(Pkey1),nl,
print('    " and valor_atributo = @doc_id)'),nl,
print('    update dbo.'),
print(Classe4),nl,
print('     SET descricao ="'),
print(Nviol),
print('" WHERE '),
print(Pkey1),
print('=@doc_id'),nl,
```

```
          print('  END'),nl,nl,
         print('  if exists (select * from deleted,inserted where '),nl,
         print('  deleted.'),
         print(Pkey2),
         print('!=NULL'),
         print(' and inserted.'),
         print(Pkey2),
         print('=NULL)'),nl,
         print('  BEGIN'),nl,
         print('    declare'),nl,
         print('      @doc_id2    int,'),nl,
         print('      @d_id2    int'),nl,nl,
         print('    set @d_id2= (select deontica_id from dbo.idealidade
where '),nl,
         print('    relacao_e="'),
         print(Relacao_i),
         print('")'),nl,
         print('    set @doc_id2=(select '),
         print(Pkey1),
         print(' from deleted)'),nl,
         print('    insert into
dbo.violacao(deontica_id,tabela,atributo,valor_atributo)'),nl,
         print('    values (@d_id2,"'),
         print(Classe1),
         print('","'),
         print(Pkey1),
         print('",@doc_id2)'),nl,
         print('    update dbo.'),
         print(Classe4),nl,
         print('     SET descricao ="'),
         print(Viol),
         print('" WHERE '),
         print(Pkey1),
         print('=@doc_id2'),nl,
         print('  END'),nl,nl,
         print('  return'),nl,nl,nl,
         print('error:'),nl,
         print('  raiserror @errno @errmsg'),nl,
             print('  rollback  transaction'),nl,nl,
         print('END'),nl,
         print('go'),nl,nl,nl,nl,

         %Trigger que implementa o comportamento do sistema, quando
         %é apagado um registo, na  tabela a que corresponde.

         print('create trigger dbo.'),
         print(Classe4),
         print('_delete on dbo.'),
         print(Classe4),
         print(' for delete as'),nl,nl,
         print('BEGIN'),nl,
         print('  declare'),nl,
         print('    @reg    int,'),nl,
         print('    @errno    int,'),nl,
         print('    @errmsg   varchar(255)'),nl,nl,
         print('  set @reg=(select '),
         print(Pkey1),
         print(' from deleted)'),nl,
         print('  delete from dbo.'),
```

```
          print(Classe1),
          print(' WHERE '),
          print(Classe1),
          print('.'),
          print(Pkey1),
          print('=@reg'),nl,
          print('  print "tipo registo apagado"'),nl,
          print('  if exists (select * from deleted,violacao where '),nl,
          print('  (violacao.tabela='),print('"'),
          print(Classe1),print('"'),
          print(' and violacao.atributo='),print('"'),
          print(Pkey1),print('"'),
          print(' and deleted.'),
          print(Pkey1),
          print('=violacao.valor_atributo))'),nl,
          print('  BEGIN'),nl,
          print('    declare'),nl,
          print('      @atributo   int'),nl,nl,
          print('    set @atributo=(select '),
          print(Pkey1),
          print(' from deleted)'),nl,
          print('    delete from violacao where '),nl,
          print('    (violacao.tabela='),print('"'),
          print(Classe1),print('"'),
          print(' and violacao.atributo='),print('"'),
          print(Pkey1),print('"'),
          print(' and valor_atributo=@atributo)'),nl,
          print('  END'),nl,nl,
          print('  return'),nl,nl,
          print('error:'),nl,
          print('  raiserror @errno @errmsg'),nl,
              print('  rollback  transaction'),nl,
          print('END'),nl,
          print('go'),nl,nl,nl,nl,

          %Trigger que implementa o comportamento do sistema, quando
          %é apagado um registo, na  tabela a que corresponde.

          print('create trigger dbo.'),
          print(Classe4),
          print('_insert_up on dbo.'),
          print(Classe4),
          print(' for insert,update as'),nl,nl,
          print('BEGIN'),nl,
          print('  declare'),nl,
          print('    @errno    int,'),nl,
          print('    @errmsg   varchar(255)'),nl,nl,
          print('  if exists (select * from inserted,violacao where
'),nl,
          print('  (violacao.tabela='),print('"'),
          print(Classe1),print('"'),
          print(' and violacao.atributo='),print('"'),
          print(Pkey1),print('"'),
          print(' and inserted.'),
          print(Pkey1),
          print('=violacao.valor_atributo))'),nl,
          print('  BEGIN'),nl,
          print('    declare'),nl,
          print('      @num_id   int'),nl,nl,
```

```
print('    set @num_id=(select '),
print(Pkey1),
print(' from inserted)'),nl,
print('    update dbo.'),
print(Classe4),nl,
print('     SET descricao ="'),
print(Viol),
print('" WHERE '),
print(Pkey1),
print('=@num_id'),nl,
print('  END'),nl,nl,
print('  ELSE'),nl,
print('  BEGIN'),nl,
print('    declare'),nl,
print('      @num_id2   int'),nl,nl,
print('    set @num_id2=(select '),
print(Pkey1),
print(' from inserted)'),nl,
print('    update dbo.'),
print(Classe4),nl,
print('     SET descricao ="'),
print(Nviol),
print('" WHERE '),
print(Pkey1),
print('=@num_id2'),nl,
print('  END'),nl,nl,
print('  return'),nl,nl,
print('error:'),nl,
print('  raiserror @errno @errmsg'),nl,
    print('  rollback  transaction'),nl,
print('END'),nl,
print('go'),nl,nl,nl,nl,fail).
```

Ficheiro de saída e Transact SQL

```
create table dbo.cabimento(
          cabimento_id                  int             not null,
               rubrica          varchar(254)                 null,
                 plano          varchar(254)                 null,
                 valor               numeric                 null,
         fornecedor_id                  int                 null,
)

alter table dbo.cabimento
add constraint pk_cabimento primary key nonclustered (cabimento_id) on
"default"


create table dbo.fornecedor(
         fornecedor_id                  int             not null,
          contribuinte                  int                 null,
                  nome          varchar(254)                 null,
              contacto          varchar(254)                 null,
)

alter table dbo.fornecedor
add constraint pk_fornecedor primary key nonclustered (fornecedor_id) on
"default"


create table dbo.pagamento(
          pagamento_id                  int             not null,
               factura                  int                 null,
                 valor               numeric                 null,
          cabimento_id                  int                 null,
)

alter table dbo.pagamento
add constraint pk_pagamento primary key nonclustered (pagamento_id) on
"default"
```

```
create table dbo.documento(
            numero_id                  int             not null,
                 data             datetime                 null,
             tipo_doc          varchar(254)                null,
           assunto_id                  int                null,
)

alter table dbo.documento
add constraint pk_documento primary key nonclustered (numero_id) on
"default"


create table dbo.assunto(
            assunto_id                  int             not null,
          descricao_g          varchar(254)                null,
          descricao_e          varchar(254)                null,
         codigo_plano          varchar(254)                null,
)

alter table dbo.assunto
add constraint pk_assunto primary key nonclustered (assunto_id) on
"default"


create table dbo.tipo_registo(
            descricao          varchar(254)                null,
            numero_id                  int             not null,
)

alter table dbo.tipo_registo
add constraint pk_tipo_registo primary key nonclustered (numero_id) on
"default"

alter table dbo.cabimento
add constraint cabimento_fk foreign key (fornecedor_id )
references dbo.fornecedor (fornecedor_id )


alter table dbo.pagamento
```

```
add constraint pagamento_fk foreign key (cabimento_id )
references dbo.cabimento (cabimento_id )


alter table dbo.documento
add constraint documento_fk foreign key (assunto_id )
references dbo.assunto (assunto_id )


alter table dbo.tipo_registo
add constraint tipo_registo_fk foreign key (numero_id )
references dbo.documento (numero_id )




create table dbo.idealidade (
  deontica_id          integer                    not null,
  tabela_um            varchar(254)               null,
     tabela_dois          varchar(254)               null,
     tabela_p             varchar(254)               null,
     relacao_e            varchar(254)               null,
     idealidade           char(1)                    null,
     relacao_d            integer                    null
)

alter table dbo.idealidade
add constraint pk_idealidade primary key nonclustered (deontica_id)
on "default"

create table dbo.violacao (
  violacao_id          int                        not null,
  deontica_id          integer                    null,
  tabela               varchar(254)               null,
  atributo             varchar(254)               null,
  valor_atributo       int                        null
)

alter table dbo.violacao
add constraint pk_violacao primary key nonclustered (violacao_id)
```

```
on "default"



INSERT INTO dbo.idealidade
(deontica_id,tabela_um,tabela_dois,tabela_p,relacao_e,idealidade)
VALUES (1,"cabimento","fornecedor","cabimento","fornecedor_fk","i")


INSERT INTO dbo.idealidade
(deontica_id,tabela_um,tabela_dois,tabela_p,relacao_e,idealidade,relacao_
d)
VALUES (4,"cabimento","pagamento","cabimento","cabimento_fk","f",1)
go


INSERT INTO dbo.idealidade
(deontica_id,tabela_um,tabela_dois,tabela_p,relacao_e,idealidade)
VALUES (2,"documento","assunto","documento","fk_assunto","i")


INSERT INTO dbo.idealidade
(deontica_id,tabela_um,tabela_dois,tabela_p,relacao_e,idealidade,relacao_
d)
VALUES (3,"documento","tipo_registo","documento","fk_tiporegisto","o",2)
go


create trigger dbo.cabimento_delete on dbo.cabimento for delete as

BEGIN
  if exists (select * from deleted,violacao where
  (violacao.tabela="cabimento" and violacao.atributo="cabimento_id" and
deleted.cabimento_id=violacao.valor_atributo))
  BEGIN
    declare
      @atributo int,
      @errno    int,
      @errmsg   varchar(255)
```

```
    set @atributo=(select cabimento_id from deleted)
    delete from violacao where
    (violacao.tabela="cabimento" and violacao.atributo="cabimento_id" and
valor_atributo=@atributo)
  END

  return

error:
  raiserror @errno @errmsg
  rollback  transaction
END
go



create trigger dbo.cabimento_insert on dbo.cabimento for insert as

if exists (select * from inserted where
inserted.fornecedor_id=NULL)

BEGIN
  declare
    @d_id int,
    @ca_id int,
    @errno    int,
    @errmsg   varchar(255)

  set @d_id=(select deontica_id from dbo.idealidade where
relacao_e="fornecedor_fk")
  set @ca_id=(select cabimento_id from inserted)
  insert into dbo.violacao (deontica_id, tabela, atributo,
valor_atributo)
  values (@d_id, "cabimento","cabimento_id",@ca_id)

  return

error:
  raiserror @errno @errmsg
  rollback  transaction
```

```
END
go



create trigger dbo.cabimento_update on dbo.cabimento for update as

BEGIN
  declare
    @errno    int,
    @errmsg   varchar(255)

  if exists (select * from deleted,inserted where
  deleted.fornecedor_id=NULL and inserted.fornecedor_id!=NULL)
  BEGIN
    declare
      @cab_id int

    set @cab_id=(select cabimento_id from deleted)
    delete from violacao where
    (violacao.tabela="cabimento" and violacao.atributo="cabimento_id" and
valor_atributo=@cab_id)
  END

  if exists (select * from deleted,inserted where
  deleted.fornecedor_id!=NULL and inserted.fornecedor_id=NULL)
  BEGIN
    declare
      @cab_id2 int,
      @d_id2 int

    if exists (select * from inserted,pagamento where
    inserted.cabimento_id=pagamento.cabimento_id)
    BEGIN
      print " Nao e possivel alterar o fornecedor, ja existem pagamentos"
      rollback transaction
      return
    END
```

```
    set @d_id2=(select deontica_id from dbo.idealidade where
relacao_e="fornecedor_fk")
    set @cab_id2=(select cabimento_id from deleted)
    insert into dbo.violacao (deontica_id, tabela, atributo,
valor_atributo)
    values (@d_id2,"cabimento","cabimento_id",@cab_id2)
  END

  return


error:
  raiserror @errno @errmsg
  rollback  transaction

END
go



create trigger dbo.pagamento_insert_up on dbo.pagamento for insert,update
as

if exists (select * from inserted,violacao where
(violacao.tabela="cabimento" and violacao.atributo="cabimento_id
" and inserted.cabimento_id=violacao.valor_atributo))
BEGIN
  declare
    @viola_id int,
    @ideal_id char,
    @errno    int,
    @errmsg   varchar(255)

  set @viola_id = (select violacao.deontica_id from inserted,violacao
where
  (violacao.tabela="cabimento" and violacao.atributo="cabimento_id" and
  inserted.cabimento_id=violacao.valor_atributo))
  set @ideal_id =(select idealidade.idealidade from idealidade where
  @viola_id=idealidade.relacao_d)
  if @ideal_id="f"
```

```
  BEGIN
    rollback transaction
    print "Proibido inserir pagamentos, não existe código fornecedor"
  END
  return


error:
  raiserror @errno @errmsg
  rollback  transaction

END
go



create trigger dbo.documento_insert on dbo.documento for insert as

BEGIN
  declare
    @errno    int,
    @errmsg   varchar(255)

if exists (select * from inserted where
inserted.assunto_id=NULL)
  BEGIN
    declare
      @d_id    int,
      @docp_id int

    set @d_id = (select deontica_id from dbo.idealidade where
    relacao_e="fk_assunto")
    set @docp_id=(select numero_id from inserted)
    insert into dbo.violacao(deontica_id,tabela,atributo,valor_atributo)
    values (@d_id,"documento","numero_id",@docp_id)
    insert into dbo.tipo_registo(numero_id, descricao)
     VALUES (@docp_id,"pré - registo")
  END

  else
```

```
  BEGIN
    declare
      @docv_id int

    set @docv_id=(select numero_id from inserted)
    insert into dbo.tipo_registo(numero_id,descricao) VALUES
(@docv_id,"vivo")
  END
  return


error:
  raiserror @errno @errmsg
  rollback  transaction

END
go



create trigger dbo.documento_update on dbo.documento for update as

BEGIN
  declare
    @errno    int,
    @errmsg   varchar(255)

  if exists (select * from deleted,inserted where
  deleted.assunto_id=NULL and inserted.assunto_id!=NULL)
  BEGIN
    declare
      @doc_id    int

    set @doc_id = (select numero_id from deleted)
    delete from dbo.violacao WHERE
    (violacao.tabela="documento" and violacao.atributo="numero_id
    " and valor_atributo = @doc_id)
    update dbo.tipo_registo
     SET descricao ="vivo" WHERE numero_id=@doc_id
  END
```

```
  if exists (select * from deleted,inserted where
  deleted.assunto_id!=NULL and inserted.assunto_id=NULL)
  BEGIN
    declare
      @doc_id2    int,
      @d_id2    int

    set @d_id2= (select deontica_id from dbo.idealidade where
    relacao_e="fk_assunto")
    set @doc_id2=(select numero_id from deleted)
    insert into dbo.violacao(deontica_id,tabela,atributo,valor_atributo)
    values (@d_id2,"documento","numero_id",@doc_id2)
    update dbo.tipo_registo
     SET descricao ="pré - registo" WHERE numero_id=@doc_id2
  END

  return


error:
  raiserror @errno @errmsg
  rollback  transaction

END
go



create trigger dbo.tipo_registo_delete on dbo.tipo_registo for delete as

BEGIN
  declare
    @reg    int,
    @errno    int,
    @errmsg   varchar(255)

  set @reg=(select numero_id from deleted)
  delete from dbo.documento WHERE documento.numero_id=@reg
  print "tipo registo apagado"
```

```
  if exists (select * from deleted,violacao where
  (violacao.tabela="documento" and violacao.atributo="numero_id" and
deleted.numero_id=violacao.valor_atributo))
  BEGIN
    declare
      @atributo   int

    set @atributo=(select numero_id from deleted)
    delete from violacao where
    (violacao.tabela="documento" and violacao.atributo="numero_id" and
valor_atributo=@atributo)
  END

  return

error:
  raiserror @errno @errmsg
  rollback  transaction
END
go



create trigger dbo.tipo_registo_insert_up on dbo.tipo_registo for
insert,update as

BEGIN
  declare
    @errno    int,
    @errmsg   varchar(255)

  if exists (select * from inserted,violacao where
  (violacao.tabela="documento" and violacao.atributo="numero_id" and
inserted.numero_id=violacao.valor_atributo))
  BEGIN
    declare
      @num_id   int

    set @num_id=(select numero_id from inserted)
    update dbo.tipo_registo
```

```
     SET descricao ="pré - registo" WHERE numero_id=@num_id
  END

  ELSE
  BEGIN
    declare
      @num_id2   int

    set @num_id2=(select numero_id from inserted)
    update dbo.tipo_registo
     SET descricao ="vivo" WHERE numero_id=@num_id2
  END

  return

error:
  raiserror @errno @errmsg
  rollback  transaction
END
go
```

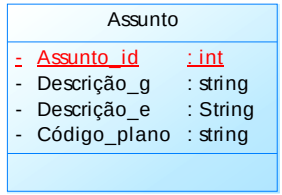
Anexo IV Estrutura dos dados de entrada para a aplicação Prolog

Estrutura do ficheiro de entrada de dados - Prolog

Dados relativos a classes

Predicado Prolog	*Proveniência/Destino*	*Sintaxe da estrutura*
class	Classe/Tabela	**Class**([*nome da classe]*).
attribute	Caracterização dos atributos da classe/Campos de tabela.	**attribute**([nome classe],[nome atributo],[tipo dados], [null Yes/no],[unique yes/no]).
attributeID	Chave única da classe/Chave primária da tabela.	**attributeID(**[nome classe],[nome atributo]).

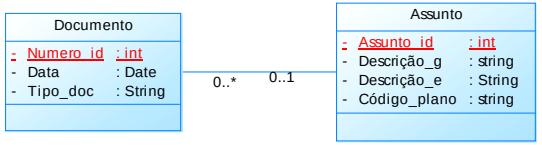
Exemplo gráfico

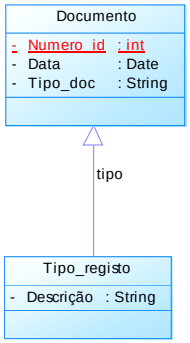


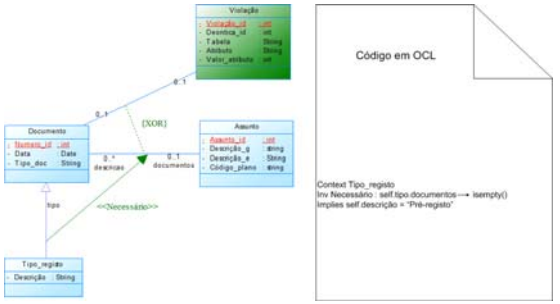
Dados relativos a associações

Axioma Prolog	*Proveniência/Destino*	*Sintaxe da estrutura*
association	Associação entre classes/Relacionamento entre tabelas.	**association** ([*nome da associação]*).
associationMember (nota: duas por associação)	Caracterização das associações entre classes/Caracterização das relações entre tabelas	**associationMember**([nome associação],[nome classe],[nome classe],[limite inferior cardinalidade], [limite superior cardinalidade]). (nota: duas por associação).

Exemplo gráfico



Dados relativos a generalizações			***Exemplo gráfico***
Axioma Prolog	*Proveniência/Destino*	*Sintaxe da estrutura*	
generalization	Generalização entre classes/Tabela pai	**generalization** ([nome classe],[tipo generalização]).	
specification	Especificação das classes envolvidas na generalização/Tabela filho	**specification**([nome classe],[nome subclasse]).	
registration	Atributo / Registo na tabela	**registration**([nome atributo classe],[“string”]).	

Dados relativos a restrições contrary- to-duties			***Exemplo gráfico***
Axioma Prolog	*Proveniência/Destino*	*Sintaxe da estrutura*	
ideality (Nota: corresponde aos dados da "Obrigação" inicial)	Estereótipo+ código em OCL proposto/Triggers no modelo relacional. (Nota: Corresponde ao mapeamento do estereótipo, e código OCL para triggers)	**Ideality(**[identificador único da restrição deôntica],[nome da classe que implementa a restrição deôntica inicial],[nome da classe que caracteriza a restrição deôntica inicial],[nome da associação que define a restrição deôntica inicial],[tipo de restrição dôntica]).	
restrition (Nota: corresponde aos dados da restrição que surge por violação da obrigação inicial)	Estereótipo+ código em OCL proposto/Triggers no modelo relacional. (Nota: Corresponde ao mapeamento do estereótipo, e código OCL para triggers	r**estrition(**[identificador único da restrição deôntica],[nome da classe que implementa a restrição deôntica inicial],[nome da classe que caracteriza a restrição deôntica que surge por violação da obrigação inicial], [nome da associação/generalização que define a restrição deôntica inicial],[tipo de restrição dôntica],[numero que corresponde à identificação da obrigação inicial]).	

Anexo V Instruções para correr o código da aplicação Prolog

Instruções

A aplicação para geração automática do modelo relacional foi desenvolvida em "LPA win-prolog 4.500". O funcionamento gera,l da aplicação referida, pressupõe o carregamento de um ficheiro de dados chamado "base2.pl", cujo conteúdo se encontra listado no anexo três. A estrutura de dados utilizada no referido ficheiro encontra-se descrita no anexo quatro.

Após leitura dos dados a aplicação Prolog processa a informação e gera o ficheiro "tsql.txt" que contém o código, em "Transact SQL", que possibilita a geração integral do modelo relacional. O código em "Transact SQL" foi preparado para ser executado em "Sybase Adaptive Server versão 15.0.2".

Pressupostos para correr a aplicação:

- *Instalação do "LPA win-prolog versão 4.500";*
- *Cópia dos ficheiros "base2.pl" e "crelacional.pl" para a directoria que contém o programa* "LPA win-prolog 4.500", normalmente: "c:\LPA Prolog 4.5";
- *Instalação da base de dados* "Sybase Adaptive Server versão 15.0.2";

Instruções para correr a aplicação:

1. *Dentro da directoria que contém o "LPA win-prolog versão 4.500" executar a aplicação "PRO386W";*
2. *Executar o comando de menu "File/Open", e posicionar na directoria do "LPA win-prolog"; abrir o ficheiro "crelacional.pl";*
3. *Executar o comando de menu " Run/Compile All";*
4. *Executar o comando de menu "Run/Application";*
5. *No formulário "Application", na combo box "Main" escolher a opção "map"; na combo box "Abort" escolher a opção "relacional";*

6. *Pressionar o botão "Ok";*
7. *No formulário "WIN-PROLOG" fazer "Ok".*

Se a execução da aplicação tiver sucesso é criado um ficheiro; "tsql.txt" que contém o código, em "Transact SQL", para geração automática do modelo relacional.

Para a geração do modelo relacional:

1. *Copiar todo o conteúdo do ficheiro "tsql.txt"*
2. *Abrir o "Sybase Adaptive Server versão 15.0.2"*
3. *Criar uma nova base de dados, por exemplo com o nome "deontica"*
4. *Com o cursor por cima da base de dados, com o botão direito do rato selecionar o aplicativo "Open Interactive SQL"*
5. *Na janela "SQL Statements" fazer "paste", de seguida executar o código*
6. *Deverá ter sido criada, na base de dados "deôntica" toda a estrutura relacional*

Curriculum Vitae