



Universidade Nova de Lisboa
OMNIS CIVITAS CONTRA SE DIVISA NON STABIT
Faculdade de Ciências e Tecnologia

Departamento de Informática

Secção de CTP

Projecto Final de Curso

Round-Trip Engineering

Avaliação do estado da arte

Miguel Malta Romeiras Mourão Maciel

Ano Lectivo 2002/2003

Orientador:

Prof. Fernando Brito e Abreu

Resumo

As vantagens associadas à utilização de ferramentas CASE numa organização são inquestionáveis. Entre estas vantagens encontra-se a uniformização do processo de desenvolvimento de software, automatização de actividades, integração das várias fases que compõem um projecto e demonstração de consistência. Diminuem assim o esforço do processo de desenvolvimento de software e contribuem para a melhoria da qualidade.

Em relação à automatização de actividades é importante referir o desenvolvimento cíclico, vulgarmente conhecido por *Round-Trip Engineering*. Este relatório final de curso tem o objectivo de avaliar o estado da arte do processo de *Round-Trip Engineering*, ao estudar profundamente três das mais conhecidas ferramentas com essa capacidade para o desenvolvimento em UML/Java. O estudo das ferramentas é composto por uma análise qualitativa, em que é estudado pormenorizadamente o processo de *Forward* e *Reverse Engineering* em cada elemento de uma fase de modelação e uma análise quantitativa, em que se propõe quantificar o sucesso do processo de *Round-Trip Engineering*.

Palavras Chave

UML, Java, *round-trip engineering*, *forward engineering*, *reverse engineering*, geração automática de código, desenvolvimento cíclico, comparação de ferramentas CASE, engenharia de software, evolução de software, manutenção de software.

Abstract

The advantages associated to the use of CASE tools in an organization are unquestionable. Among these advantages one can find the standardization of the software development process, automation of activities, integration of the several phases in a project and consistence demonstration. Thus, the development effort is decreased and the quality of the development process is improved.

Regarding the automation it is important to refer the cyclical development of code, commonly known for Round-Trip Engineering. This graduation report has the purpose of evaluating the state of the art of the process of Round-Trip Engineering, by studying three of the most widespread tools with that ability with UML/Java. The study of the tools is composed of a qualitative analysis in which it is studied in detail the process of Forward and Reverse Engineering in each element of a modeling phase and a quantitative analysis in which it is evaluated the success of the process of Round-Trip Engineering.

Keywords

UML, Java, round-trip engineering, forward engineering, reverse engineering, automated generation of source code, CASE tools comparison, software engineering, software evolution, software maintenance.

Agradecimentos

Gostaria de agradecer ao Prof. Fernando Brito e Abreu por todo o apoio, motivação, amabilidade e disponibilidade que me prestou ao longo deste projecto final de curso. Gostaria de lhe agradecer por me ter ensinado a dar o meu melhor no meu trabalho.

Gostaria de agradecer aos meus pais por terem possibilitado e proporcionado as condições necessárias para obter esta licenciatura.

Por fim gostaria de agradecer e dedicar este relatório de projecto final de curso aos meus pais e à Ana Maria por todo o apoio que me deram e que foi essencial para a sua conclusão.

Índice Geral

I - Introdução	1
I.1 - Objectivos	1
I.2 - Enquadramento do Projecto.....	1
I.3 - Organização do Relatório	2
II - Conceitos	5
II.1 - Forward Engineering	5
II.2 - Reverse Engineering.....	6
II.3 - Round-Trip Engineering.....	9
II.4 - Importação e exportação de modelos através de XMI.....	10
III - Metodologia	13
IV - Análise Qualitativa das ferramentas de RTE	17
IV.1 – Pacotes e Interdependências.....	17
IV.1.1 – Forward Engineering.....	20
IV.1.2 – Reverse Engineering	22
IV.1.3 – Resumo dos resultados e conclusões.....	24
IV.2 – Classes.....	25
IV.2.1 – Forward Engineering.....	26
IV.2.2 – Reverse Engineering	28
IV.2.3 – Resumo dos resultados e conclusões.....	30
IV.3 – Herança entre classes	31
IV.3.1 – Forward Engineering.....	32
IV.3.2 – Reverse Engineering	33
IV.3.3 – Resumo dos resultados e conclusões.....	34
IV.4 – Visibilidade	35
IV.4.1 – Forward Engineering.....	37
IV.4.2 – Reverse Engineering	40
IV.4.3 – Resumo dos resultados e conclusões.....	42
IV.5 – Associações Simples	43
IV.5.1 – Forward Engineering.....	46
IV.5.2 – Reverse Engineering	48
IV.5.3 – Resumo dos resultados e conclusões.....	50
IV.6 – Agregações	51
IV.6.1 – Forward Engineering.....	52
IV.6.2 – Reverse Engineering	54
IV.6.3 – Resumo dos resultados e conclusões.....	56

IV.7 – Composições	57
IV.7.1 – Forward Engineering.....	58
IV.7.2 – Reverse Engineering	60
IV.7.3 – Resumo dos resultados e conclusões.....	62
IV.8 – Classes Associativas.....	63
IV.8.1 – Forward Engineering.....	65
IV.8.2 – Reverse Engineering	68
IV.8.3 – Resumo dos resultados e conclusões.....	70
IV.9 – Interfaces	71
IV.9.1 – Forward Engineering.....	74
IV.9.2 – Reverse Engineering	76
IV.9.3 – Resumo dos resultados e conclusões.....	77
V - Análise Quantitativa das ferramentas de RTE	79
V.1 – Modelo de teste utilizado.....	79
V.2 – Resultados Obtidos	79
V.2.1 – Apresentação dos resultados.....	79
V.2.2 – Discussão dos resultados	80
V.3 - Conclusões	82
VI - Case-Study Cinema.....	85
VI.1 - Apresentação do Case-Study	85
VI.2 - Modelos resultantes e observações	86
VII – Apreciação Crítica e Conclusões	91
VIII – Evolução Futura.....	93
Bibliografia.....	95
Anexos.....	A1
Anexo A - Metamodelo UML 1.*	A1

Índice de Figuras

Figura 1 - Processo de <i>Forward Engineering</i>	5
Figura 2 - Processo de <i>Reverse Engineering</i>	7
Figura 3 - Processo de <i>Round-Trip Engineering</i>	9
Figura 4 – Gráfico de Gantt.....	13
Figura 5 – Metodologia da fase quantitativa	14
Figura 6 – Dados da ferramenta <i>Objectteering UML Modeler</i>	15
Figura 7 – Dados da ferramenta <i>Rational Rose Enterprise Edition</i>	15
Figura 8 – Dados da ferramenta <i>Borland Together Control Center</i>	16
Figura 9 - Representação gráfica de um pacote.....	17
Figura 10 – Modelo de teste para pacotes e interdependências.....	18
Figura 11 – Modelos resultantes para pacotes e interdependências	22
Figura 12 – Representação gráfica de uma classe	25
Figura 13 – Modelo de teste para classes, atributos e operações.....	25
Figura 14 – Modelos resultantes para classes, atributos e operações	28
Figura 15 – Modelo de teste para herança entre classes.....	31
Figura 16 – Modelos resultantes para herança entre classes	33
Figura 17 – Modelo de teste para visibilidade de atributos e operações	36
Figura 18 – Modelos resultantes para visibilidade de atributos e operações.....	40
Figura 19 – Modelo de teste para associações simples.....	43
Figura 20 – Modelos resultantes para associações simples.....	48
Figura 21 – Modelo de teste para agregações.....	51
Figura 22 – Modelos resultantes para agregações	54
Figura 23 – Modelo de teste para composições.....	57
Figura 24 – Modelos resultantes para composições	60
Figura 25 – Modelo de teste para classes associativas	63
Figura 26 – Modelos resultantes para classes associativas.....	68
Figura 27 – Representação gráfica normal de uma interface	71
Figura 28 - Representação gráfica compacta de uma interface	71
Figura 29 – Representação gráfica normal de uma relação de realização.....	72
Figura 30 - Representação gráfica compacta de uma relação de realização.....	72
Figura 31 – Modelo de teste para interfaces.....	72
Figura 32 – Modelos resultantes para interfaces	76
Figura 33 – Modelo original.....	81
Figura 34 – Modelo recuperado	81
Figura 35 – Modelo recuperado importado para efeitos de extracção de métricas	81

Figura 36 – Modelo de teste utilizado no case-study	85
Figura 37 – Modelo resultante para a ferramenta <i>Objectteering UML Modeler</i>	86
Figura 38 – Modelo resultante para a ferramenta Objectteering UML Modeler	87
Figura 39 – Modelo resultante para a ferramenta Rational Rose Enterprise Edition	88

Índice de Tabelas

Tabela 1 – Tabela comparativa para pacotes e interdependências	24
Tabela 2 – Tabela comparativa para classes, atributos e operações	30
Tabela 3 – Tabela comparativa para herança entre classes	34
Tabela 4 – Tabela comparativa para visibilidade de atributos e operações.....	42
Tabela 5 – Tabela comparativa para associações simples	50
Tabela 6 – Tabela comparativa para agregações	56
Tabela 7 – Tabela comparativa para composições	62
Tabela 8 – Tabela comparativa para classes associativas.....	70
Tabela 9 – Tabela comparativa para interfaces	77
Tabela 10 – Tabela de métricas da análise quantitativa	80

I - Introdução

I.1 - Objectivos

O projecto final de curso cujo relatório se apresenta tem como principal objectivo chegar a conclusões acerca do estado da arte do processo de *Round-Trip Engineering*.

Através da análise qualitativa de três ferramentas com capacidade para *Round-Trip Engineering* pretende-se tirar conclusões relativamente à capacidade das mesmas em efectuar o processo de *Forward* e *Reverse Engineering* para cada elemento utilizado numa fase de modelação. O estudo isolado de cada um destes elementos permite observar as correspondências que cada ferramenta efectua entre os dois níveis de abstracção presentes - modelação em UML e código fonte em Java.

A análise quantitativa permite quantificar a quantidade de informação que se perde no processo de *Round-Trip*. O objectivo desta fase é perceber até que ponto existem vantagens em usar *Round-Trip Engineering* num dado projecto de software, pesando vantagens e inconvenientes.

I.2 - Enquadramento do Projecto

O projecto aqui apresentado tem como objectivo a conclusão da Licenciatura em Engenharia Informática na Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa. Levou ainda à integração no grupo QUASAR – *Quantitative Approaches on software Engineering And Reengineering*, a convite do Prof. Fernando Brito e Abreu.

O grupo QUASAR tem como objectivo contribuir para o avanço do estado da arte de engenharia de software, tendo sido pioneiro na introdução de boas práticas na área de engenharia de software experimental. Para alcançar os seus objectivos, o grupo tem uma participação contínua em eventos científicos internacionais, como organizador ou participante. Há vários anos que o grupo tem vindo a desenvolver material académico na área da qualidade de software assim como ferramentas de apoio à investigação. É objectivo do grupo aumentar a participação em redes de

investigação internacionais de modo a aumentar a troca de experiências. Tem como principais áreas de investigação a formalização de métricas, engenharia de software baseada em componentes, modelos de estimação para desenvolvimento orientado por objectos, reengenharia de sistemas legados e automatização de engenharia de software.

No corrente ano o grupo QUASAR passou a integrar a ESERNET (Experimental Software Engineering NETwork) que tem como objectivo aumentar a competitividade dos organismos europeus de software. Tem como principais parceiros Blekinge Institute of Technology, Fraunhofer Institut Experimentelles Software Engineering, Norwegian University of Science and Technology, VTT Electronics e European Software Institute (ESI).

1.3 - Organização do Relatório

No capítulo II deste relatório (Conceitos) são apresentados os conceitos teóricos relacionados com o tema em estudo. Com uma divisão em secções são apresentados os processos de *Forward Engineering*, *Reverse Engineering* e *Round-Trip Engineering*. A última secção é dedicada ao uso de XMI para importação e exportação de modelos em UML.

O capítulo III refere-se à metodologia adoptada no decorrer do projecto final de curso, cujo relatório final aqui se apresenta. São referidas e descritas as várias fases, de modo a melhor se perceber a maneira como foi efectuado este projecto.

No capítulo IV (Análise Qualitativa das ferramentas de *Round-Trip Engineering*) são apresentados os vários estudos efectuados nos elementos UML seleccionados. Existe uma secção para cada um deles: pacotes e interdependências, classes, herança entre classes, visibilidade, associações simples, agregações, composições, classes associativas e interfaces. Para cada um destes elementos em estudo existem secções dedicadas aos processos de *Forward* e *Reverse Engineering*, seguindo-se uma secção dedicada a conclusões.

O capítulo V (Análise Quantitativa das ferramentas de *Round-Trip Engineering*) é dedicado à apresentação de estatísticas descritivas das perdas de informação de desenho no processo de *Round-Trip Engineering*. É utilizado um caso de estudo

relativamente complexo para a exemplificação. Divide-se em três secções sendo que a primeira é dedicada a apresentação do modelo utilizado na análise. Segue-se uma secção com os resultados obtidos e uma outra com as conclusões.

O capítulo VI apresenta um Case-study relativo a um sistema de compra de bilhetes de um cinema. Após a apresentação do modelo utilizado apresentam-se os modelos resultantes para cada uma das ferramentas, assim como algumas observações.

No capítulo VII encontra-se a secção dedicada à Apreciação Crítica e Conclusões, onde são tiradas as conclusões acerca do estado da arte do processo de *Round-Trip Engineering*. Nesta secção está incluída uma súmula das conclusões derivadas do trabalho descrito nos capítulos anteriores deste relatório.

O capítulo VIII (Evolução Futura) identifica possíveis melhorias no estudo efectuado com o intuito de estimular estudos posteriores nesta área.

No final deste relatório encontra-se a bibliografia na qual se apoiou este relatório e inclui tanto URL's de *Internet* como livros e artigos consultados. Segue-se a secção dedicada aos anexos.

II - Conceitos

II.1 - Forward Engineering

Em teoria deve ser possível gerar a implementação (codificação) de forma automática a partir de uma especificação formal de um problema (conceito referido na literatura como *from concept to code*). Garante-se assim uma intervenção humana menor, o que pode levar a uma maior correcção e a uma poupança de recursos (esforço e tempo). O processo é ilustrado na *Figura 1*.

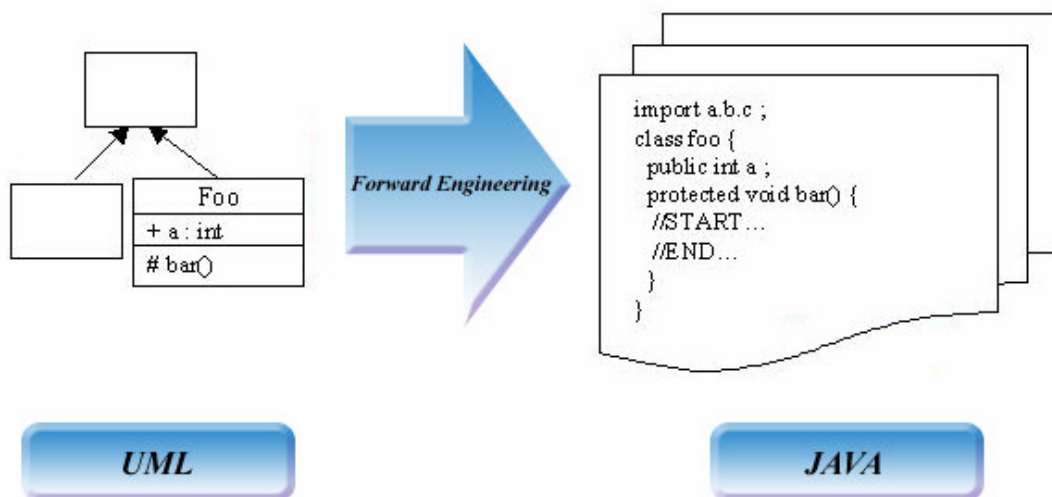


Figura 1 - Processo de *Forward Engineering*

Forward Engineering é a capacidade de gerar código fonte a partir de um modelo. A utilidade de usar *Forward Engineering* surge pela primeira vez para gerar código logo após a especificação de um modelo de desenho. Permite poupar o trabalho repetitivo de criar classes, cabeçalhos, atributos e métodos, resultando portanto na poupança de recursos. A criação do “esqueleto” do código fonte passa assim a estar a cargo de uma ferramenta, que será provavelmente a mesma onde foi definido o modelo. Em alternativa o modelo pode ser exportado e importado por uma outra ferramenta que faça o *Forward Engineering*. Esta última estratégia é denominada de *best of breed* e consiste em seleccionar as melhores ferramentas para cada uma das funcionalidades que se prevêem necessárias durante o processo de desenvolvimento de um projecto de software.

A técnica de *Forward Engineering* é também usada extensivamente durante o desenvolvimento de um projecto. Tradicionalmente o programador é um agente criativo que tem necessidade de tomar decisões na codificação, o que pode ter como consequência alterações no desenho da aplicação ou sistema em desenvolvimento. Essas alterações de desenho têm lugar na altura de tomar decisões acerca de como codificar certas partes da especificação. Muitas vezes essas decisões podem não ser óbvias para terceiros e não são incluídas na documentação. Ao longo do tempo, o código do programa é modificado de modo a remover erros e/ou a permitir a adaptação de acordo com alterações na especificação. Infelizmente essas modificações tomam muitas vezes lugar não sendo reflectidas correctamente na especificação. O código torna-se assim mais difícil de entender e conseqüentemente aumenta a dificuldade de o manter.

Para evitar a situação citada anteriormente, o ideal será mudar a especificação de acordo com as alterações pretendidas e fazer reflectir essas modificações no código. Mantém-se assim uma coerência entre os dois níveis de abstracção. Uma pré-condição necessária para este processo decorrer sem problemas é a existência de informação fiável sobre como fazer a relação entre a especificação e o código. Sendo dois níveis de abstracção completamente distintos, essa relação não é linear. É importante ter em linha de conta que as alterações que se fazem na especificação (tipicamente pequenas face à dimensão do projecto) devem ser reflectidas no código de modo a manter a correcção do mesmo e ainda a manter intacta a informação que este contém. Muitas das alterações que se pretendem fazer são ao nível da optimização, sendo esta também uma das principais razões para se chegar a uma situação de discrepância entre os dois níveis de abstracção quando não se usa *Forward Engineering*.

II.2 - Reverse Engineering

Em termos gerais o processo de *Reverse Engineering* é a análise de um sistema ou produto de modo a poder determinar como ele funciona (ou seja, a recuperar o seu modelo de funcionamento), de forma a poder reutilizá-lo ou para introduzir melhorias no mesmo. Esta definição é retirada de outras indústrias e está agora frequentemente associada à área de desenvolvimento de software.

Reverse Engineering é a capacidade de construir um modelo a partir de código fonte, fazendo a correspondência entre os dois níveis de abstracção. Em detalhe, o processo consiste em analisar um pedaço de código fonte e extrair informação relevante, de modo a poder construir posteriormente um modelo gráfico de fácil interpretação para um Engenheiro de software. O processo é ilustrado na *Figura 2*.

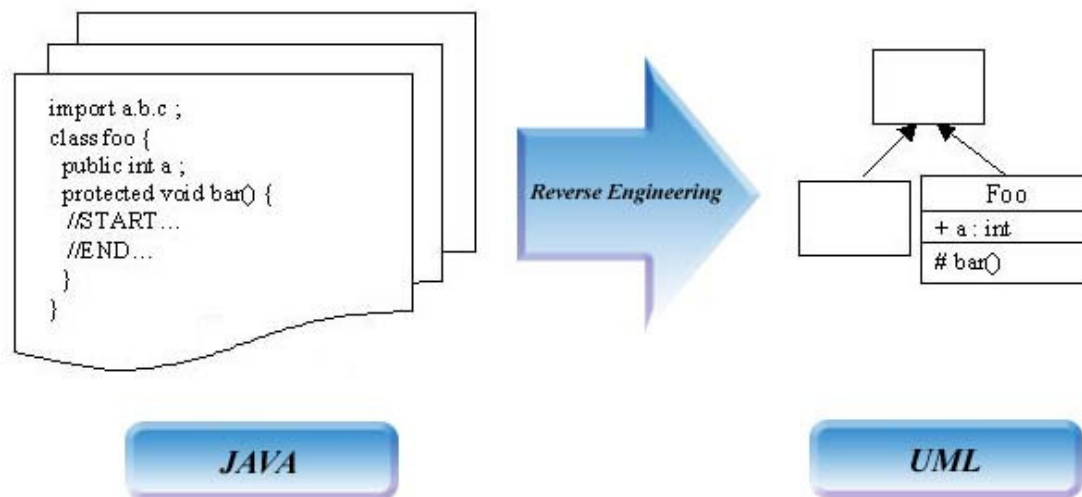


Figura 2 - Processo de *Reverse Engineering*

Em todos os sectores de negócio surgiu a necessidade de adaptar sistemas de informação à *Web*, o que criou uma grande necessidade de desenvolver métodos, ferramentas e infra-estruturas para explorar e eventualmente modificar as aplicações existentes de forma eficiente. Esta situação representa uma motivação adicional para a utilização de *Reverse Engineering*.

Se todos os sistemas desenvolvidos fossem baseados numa transformação entre os dois níveis de abstracção em questão (desenho e código fonte) não haveria necessidade de usar *Reverse Engineering*. Contudo é necessário ter consciência da realidade que são os sistemas legados em que o desenho que possa existir não está de acordo com o código desenvolvido. Esses sistemas e mesmo outros, com o desenho actualizado, têm erros e necessidades de melhorias ao longo do tempo. A evolução é uma necessidade inquestionável, existindo assim uma necessidade de compreender os programas existentes. Como muitas vezes os sistemas em questão são mal desenvolvidos logo desde a fase de desenho e têm documentação incompleta, inexistente ou mesmo errada, a tarefa de compreender sistemas para os

poder alterar torna-se um desafio. Assim surge a necessidade de usar *Reverse Engineering*.

Para compreender software existente é necessário extrair informação acerca dos artefactos que usa e relações entre eles. No caso da linguagem Java, esses artefactos podem ser classes, interfaces, variáveis e métodos. As relações podem ser entre classes, classes e interfaces ou mesmo chamadas de métodos para outros métodos. Toda esta informação é dita estática. Também pode ser extraída informação dinâmica que inclui, por exemplo, informação acerca da concorrência.

No caso da integração de um novo elemento numa equipa que esteja a trabalhar num projecto sobre o qual não exista, ou exista pouca, informação acerca do desenho e da modelação, a utilização de *Reverse Engineering* torna-se útil na medida em que permite ter uma visualização da arquitectura em questão. Outra possível situação é aquela em que um projecto desenhado ou implementado por um produtor de software passa a estar a cargo de um novo produtor ou nova equipa.

Utilizar *Reverse Engineering* é útil se for necessário criar um modelo pela primeira vez a partir de código fonte. Este poderá ser o caso em que a implementação não foi precedida por uma fase de modelação. É também útil para sincronizar um modelo com o código fonte, reflectindo assim as alterações efectuadas ao mesmo. A capacidade de visualizar todos os elementos do código através de diagramas permite entender em “directo” o impacto que as alterações efectuadas numa determinada localização de código fonte têm no sistema em desenvolvimento.

Reverse Engineering permite criar documentação referente a sistemas em situações em que não existe informação a eles associada e as pessoas responsáveis pelo seu desenvolvimento não estão disponíveis.

O processo de *Reverse Engineering* não se aplica apenas a código legado nas condições referidas anteriormente. O método permite a um Engenheiro de software garantir que o modelo está a ser cumprido de acordo com a especificação. Permite-lhe ter uma imagem geral do sistema e produzir documentação para o mesmo. Por exemplo, em casos em que um determinado programa apresenta instabilidades ocasionais, torna-se importante ter um historial (e conseqüentemente uma ordenação) das várias alterações que foram sendo efectuadas ao longo do tempo.

II.3 - Round-Trip Engineering

Round-Trip Engineering (RTE) define-se como sendo a utilização em ciclo dos dois processos referidos anteriormente neste capítulo – *Forward Engineering* e *Reverse Engineering*. A *Figura 3* ilustra o processo de *Round-Trip Engineering*.

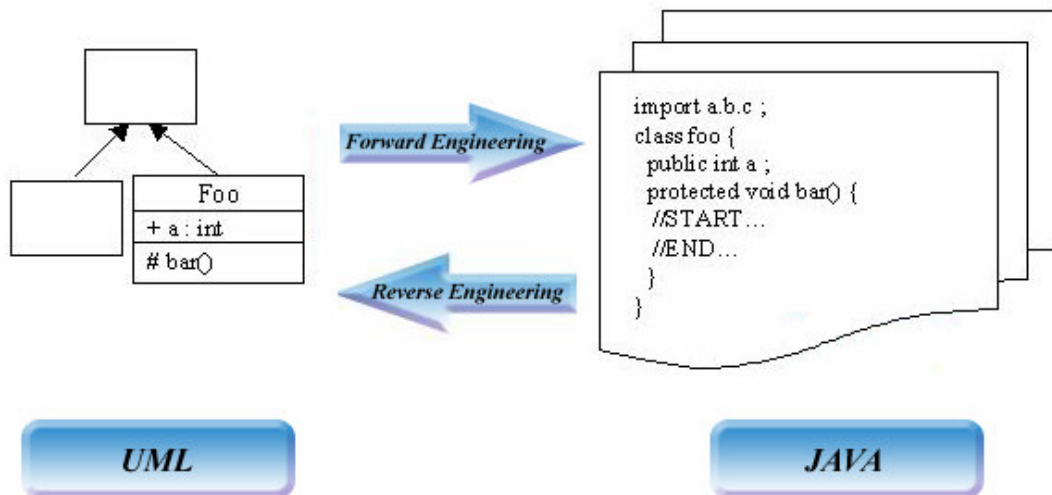


Figura 3 - Processo de *Round-Trip Engineering*

O processo de *Round-Trip Engineering* consiste em:

1. Desenhar o sistema em detalhe;
2. Gerar código para o sistema desenhado, usando ferramentas de geração automática de código (*Forward Engineering*);
3. Actualizar de forma automática e periódica (ou quando se achar adequado) o modelo de modo a reflectir o código desenvolvido com exactidão (*Reverse Engineering*).

Uma vez tendo os dois níveis de abstracção ligados através de *Forward* ou *Reverse Engineering*, as alterações efectuadas a um deles podem ser rapidamente propagadas para o outro.

O processo de *Round-Trip Engineering* permite uma sincronização constante entre os modelos representados numa linguagem de modelação – por exemplo UML – e os modelos representados em código fonte – por exemplo Java.

O *Round-Trip Engineering* permite a um arquitecto de sistemas desenvolver uma estrutura de classes básica a partir de um nível de abstracção superior, mesmo não tendo que se preocupar com detalhes ou com todas as propriedades das classes. Ao ter definidas as classes e respectivas relações, o arquitecto de sistemas pode deixar a definição em detalhe dos elementos nas mãos de implementadores (programadores). Resumidamente, o arquitecto efectua o processo de *Forward Engineering* ao modelo de modo a gerar código; o programador implementa e faz alterações que serão actualizadas no modelo através de *Reverse Engineering*. Podem existir várias iterações deste processo até se chegar a um sistema robusto.

A utilização de *Round-Trip Engineering* agrega as vantagens dos processos de *Forward* e *Reverse Engineering* anteriormente citadas, pelo que reduz a longo prazo os custos associados à geração e actualização de modelos, assim como à geração de código.

II.4 - Importação e exportação de modelos através de XMI

Assiste-se constantemente a um esforço da indústria para melhorar a troca de informação através da *Web*. Um dos frutos deste esforço é o XML, uma norma do W3C para permitir a troca de informação estruturada através da *Web*.

Até há pouco tempo não existia um formato padrão aceite pela indústria para permitir a troca de informação descrita em UML. Existia sim um grande número de formatos proprietários desenvolvidos para serem usados por ferramentas específicas.

O XMI (*XML Metadata Interchange*) é uma norma do OMG para interoperação de metadados. Esta norma resultou da colaboração entre o OMG e alguns líderes da indústria como a Unisys, IBM e a Oracle. Desenvolvido inicialmente para aplicação em metadados de modelos em UML, encontra-se neste momento em vias de extensão a outros campos como por exemplo *Data Warehousing*. Aplicado a modelos em UML, o XMI pretende disponibilizar uma estrutura de representação que permita a interoperação e utilização dos ditos modelos independentemente da plataforma, linguagem e ferramentas CASE.

Actualmente um número crescente de ferramentas suporta o formato XMI, quer na versão de base quer sobre a forma de “*plugins*”. Espera-se que esta tendência se venha a acentuar.

O potencial do XMI é inegável. Permite que equipas de desenvolvimento usem a estratégia de *best-of-breed* na escolha das ferramentas, uma vez que existe um formato padronizado de troca de meta-informação dos modelos. Em secções anteriores deste relatório é referido que podem ser utilizadas várias ferramentas durante o processo de *Round-Trip Engineering*. Por exemplo poderá ser utilizada uma ferramenta de modelação na qual apenas se define a fase de desenho e ser usada outra ferramenta para fazer a geração de código (*Forward Engineering*). Para actualizar o modelo de acordo com alterações efectuadas durante a codificação (*Reverse Engineering*) pode ser usada uma outra ferramenta. A escolha das ferramentas pode assim ser dependente de vários factores como o custo das licenças, as capacidades técnicas de cada ferramenta para determinada fase ou os conhecimentos técnicos e/ou capacidade de adaptação da equipa para uma dada ferramenta, sem haver necessidade de preocupações ao nível da compatibilidade.

Os modelos são representados num formato ASCII pelo que podem ser facilmente trocados entre aplicações ou mesmo ser legíveis por indivíduos com capacidades técnicas para o fazer. O XMI é um formato *stream*, pelo que tanto pode ser armazenado num repositório de dados como ser acedido via Internet a partir de uma base de dados.

Não era fácil encontrar até há pouco tempo na Internet modelos de análise e desenho em UML com a complexidade habitual dos casos reais. Apenas se encontravam exemplos pequenos, geralmente produzidos por académicos e num formato qualquer proprietário (por exemplo, no formato “*petal*” do *Rose*), o que dificultava a sua utilização. Essa situação contrastava (e ainda contrasta) com a do código fonte em que existem inúmeros repositórios com projectos completos, complexos e em várias linguagens de programação. Actualmente a situação começa a tomar outros rumos com a crescente divulgação do XMI.

III - Metodologia

O trabalho subjacente a este projecto final de curso foi dividido em várias actividades, com menos ou maior grau de sobreposição. O encadeamento e paralelismo das mesmas está descrito na *Figura 4*.

ID	Tarefa	2003					
		Mai	Jun	Jul	Ag	Set	Out
1	Instalação e aprendizagem das ferramentas	[Barra azul cobrindo Maio, Junho e Julho]					
2	Revisão bibliográfica	[Barra azul cobrindo Maio, Junho e Julho]					
3	Construção de casos de estudo	[Barra azul cobrindo Junho e Julho]					
4	Análise Qualitativa	[Barra azul cobrindo Julho e Agosto]					
5	Importação e exportação de modelos	[Barra azul cobrindo Setembro]					
6	Análise Quantitativa	[Barra azul cobrindo Setembro e Outubro]					
7	Elaboração do relatório	[Barra azul cobrindo Julho, Agosto, Setembro e Outubro]					

Figura 4 – Gráfico de Gantt

A primeira actividade foi dedicada à adaptação e aprendizagem das ferramentas utilizadas durante o projecto – *Objectteering UML Modeler*, *Rational Rose Enterprise Edition* e *Borland Together Control Center*. Foi também efectuada alguma pesquisa essencialmente na *Internet* acerca dos parâmetros de afinação (“*tunning*”) das mesmas ferramentas, de modo a obter os melhores resultados possíveis. Foram efectuadas várias experiências durante esta fase, com vários projectos (usando código fonte de alguns e modelos de desenho de outros) de modo a chegar a conclusões acerca de como atingir os melhores resultados possíveis.

Em paralelo com a actividade anterior foi efectuada uma revisão bibliográfica sobre o tema de *Round-Trip Engineering*, tanto ao nível de literatura impressa (encontrada em bibliotecas) como ao nível de informação acessível através da *Internet*. Dessa actividade resultou a escrita do capítulo II deste relatório.

A actividade seguinte foi dedicada à construção de casos de estudo e à análise qualitativa das ferramentas de *Round-Trip Engineering*. De modo a obter resultados

que reflectissem de forma inequívoca cada um dos aspectos em estudo foram utilizados modelos muito simples criados à medida que eram necessários. Pretendeu-se aqui garantir uma cobertura tão completa quanto possível das abstrações de modelação utilizadas nos diagramas de classes. Os modelos permitiram tirar conclusões muito detalhadas acerca de cada um dos elementos em estudo uma vez que a partir do modelo foi gerado código fonte (*Forward Engineering*) e posteriormente a partir desse código foi gerado o modelo correspondente (*Reverse Engineering*). Foram deste modo estudadas as duas fases que compõem o processo de *Round-Trip Engineering*. Os resultados obtidos passaram por uma breve comparação com exemplos existentes na bibliografia consultada (quando existentes) de modo a confirmar alguns aspectos das conclusões. Desta actividade resultou a escrita do capítulo IV deste relatório.

Após a actividade de experimentação de várias importações e exportações de modelos utilizando XMI, teve lugar a análise quantitativa das ferramentas de *Round-Trip Engineering*. Esta foi efectuada segundo a metodologia que se apresenta na *Figura 5*.

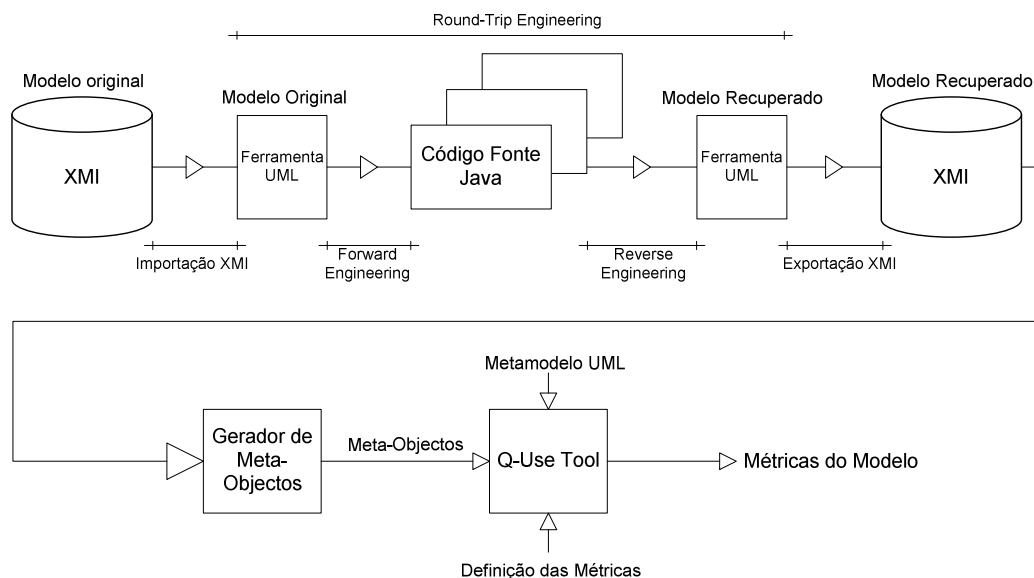


Figura 5 – Metodologia da fase quantitativa

O modelo original, no formato XMI, foi importado por cada uma das ferramentas em estudo. Tendo cada uma delas o mesmo modelo, foi possível fazer a análise quantitativa e comparar os resultados de forma imparcial e inequívoca. Foi realizado

o processo de *Round-Trip Engineering* sobre o modelo em causa em cada uma das ferramentas de modo a obter um novo modelo. Cada um destes modelos foi exportado para XMI de modo a ser possível a extracção das métricas. Foram gerados os Meta-Objectos (através do respectivo gerador) que, juntamente com o Metamodelo UML e definição das métricas tornaram possível à ferramenta *Q-Use* fazer a extracção das métricas relevantes para a análise quantitativa das ferramentas. Desta actividade resultou o capítulo V deste relatório.

Dados das ferramentas

Segue-se uma breve descrição das principais ferramentas utilizadas neste trabalho, assim como dos módulos e/ou *plugins* que foram necessários (*Figuras 6, 7 e 8*).

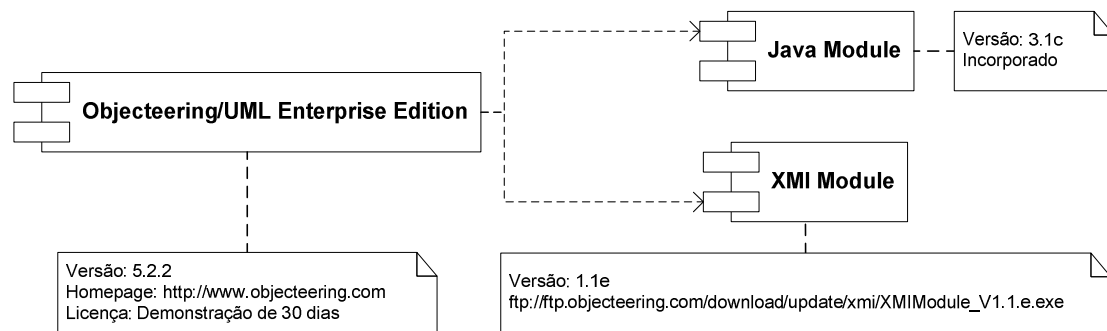


Figura 6 – Dados da ferramenta *Objecteering UML Modeler*

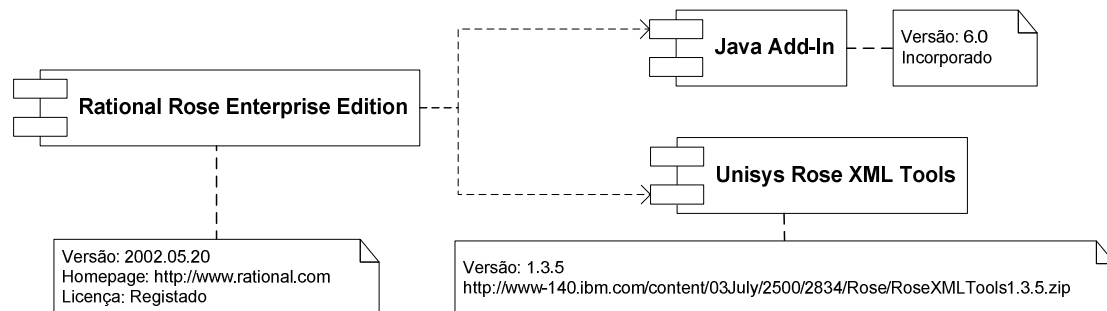


Figura 7 – Dados da ferramenta *Rational Rose Enterprise Edition*

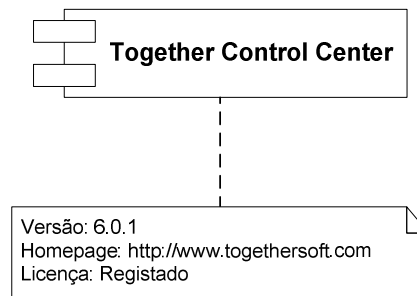


Figura 8 – Dados da ferramenta *Borland Together Control Center*

IV - Análise Qualitativa das ferramentas de RTE

Tal como referido na Organização do relatório (*Capítulo 1.3*), para cada uma das abstrações em estudo neste capítulo, existe um sub-capítulo onde se faz a apresentação teórica resumida da abstracção em estudo e se apresenta o modelo de teste, seguindo-se um sub-capítulo onde se apresentam os resultados (código fonte) do processo de *Forward Engineering* sobre o referido modelo e um outro com os resultados do *Reverse Engineering* efectuado sobre o código fonte anteriormente gerado. Existe um encadeamento forte entre os três sub-capítulos anteriormente referidos.

IV.1 – Pacotes e Interdependências

Um pacote é um elemento de uso geral que organiza elementos de um modelo em grupos. Estes elementos podem ser classes, *use cases*, diagramas ou mesmo outros pacotes. Cada elemento pode estar dentro de um único pacote, podendo assim a hierarquia ser representada por uma árvore. Como os pacotes podem referenciar outros pacotes, a rede de utilização dos mesmos pode ser representada usando um grafo.

Um sistema pode deste modo ser pensado como sendo um único pacote de alto nível que contém toda a especificação do mesmo. Um pacote tem a indicação da sua visibilidade, assunto estudado numa fase posterior deste relatório.

Em termos gráficos um pacote é um ícone com forma de pasta (*Figura 9*).

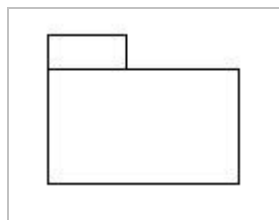


Figura 9 - Representação gráfica de um pacote

Um pacote pode ter uma dependência de/para outro pacote. Para simplificar pode-se afirmar que um pacote é o fornecedor e o outro o cliente.

A noção de dependência entre pacotes indica que as classes contidas dentro do pacote “cliente” podem:

- ☞ Herdar classes do pacote “fornecedor”;
- ☞ Conter instâncias de classes do pacote “fornecedor”;
- ☞ Usar classes do pacote “fornecedor”;
- ☞ Dependem das classes do pacote “fornecedor”.

O modelo utilizado para testar as capacidades das ferramentas em estudo em efectuarem *Round-Trip Engineering* sobre modelos com pacotes e interdependências está representado na *Figura 10*.

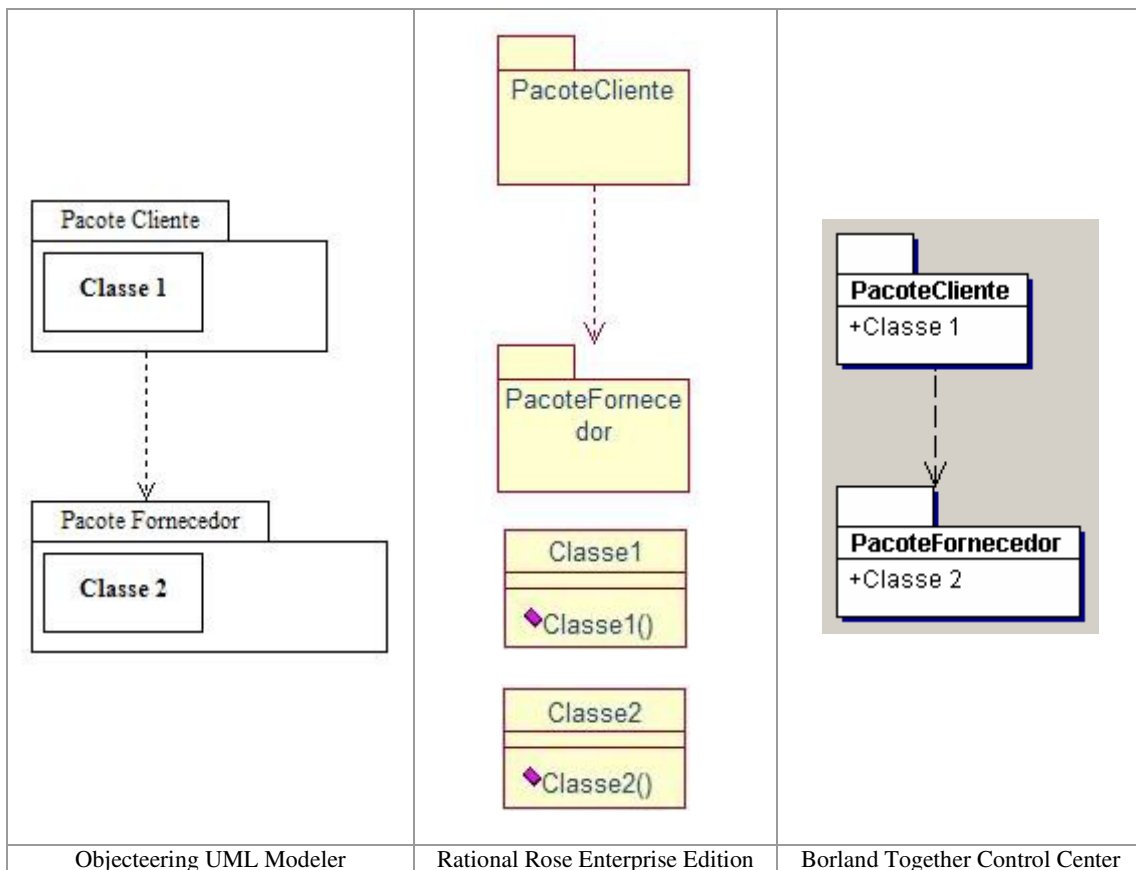


Figura 10 – Modelo de teste para pacotes e interdependências

No caso do modelo utilizado como exemplo na ferramenta *Rational Rose Enterprise Edition* a representação gráfica diverge um pouco das restantes ferramentas uma vez que esta utiliza um sistema de navegação pelos pacotes e não permite visualizar o conteúdo (neste caso as classes) dos mesmos. O modelo é, porém, igual aos utilizados pelas outras ferramentas embora esta tenha colocado um construtor por

omissão em cada uma das classes antes do início do processo de geração de código. Apresenta-se este modelo, em vez do modelo sem os construtores porque na realidade foi a partir deste que o processo de *Forward Engineering* teve lugar. O mesmo se passou em todas as outras abstracções em estudo neste relatório. Foi também nesta fase que a ferramenta corrigiu os nomes dos pacotes e das classes que continham espaços. As duas *strings* que compunham os nomes foram concatenadas.

IV.1.1 – Forward Engineering

Objectteering UML Modeler

```
package Pacote_Cliente;  
  
import Pacote_Fornecedor.*;  
  
public class Classe_1  
{  
}
```

```
package Pacote_Fornecedor;  
  
public class Classe_2  
{  
}
```

Cada uma das classes foi criada dentro de uma directoria com o nome do pacote onde foi incluída no modelo. Confirma-se que a geração de código fonte foi efectuada com sucesso, uma vez que a classe “Classe_1” foi criada com a indicação de que importa o pacote “Pacote_Fornecedor”. Ao importar o pacote está automaticamente a importar todas as classes que o mesmo contém. De referir que, uma vez que na fase de modelação os pacotes e as classes tinham nomes ilegais para código Java (uma vez que continham espaços), na geração de código fonte os espaços foram substituídos pelo carácter “_”.

Rational Rose Enterprise Edition

```
package PacoteCliente;  
  
public class Classe1  
{  
    /* @roseuid 3ED9DE68012D */  
    public Classe1()  
    {  
    }  
}
```

```
package PacoteFornecedor;  
  
public class Classe2  
{  
    /* @roseuid 3ED9DE68017D */  
    public Classe2()  
    {  
    }  
}
```

Verifica-se que o código gerado pela ferramenta contém a indicação correcta dos pacotes aos quais as classes pertencem e criou uma directoria com o nome dos pacotes. O código gerado pela ferramenta não tem indicações acerca da importação pretendida, verificando-se assim uma perda de informação no processo de *Forward Engineering*. É importante referir que a ferramenta em questão coloca um identificador (“RoselD”) por cada método existente na classe. Este identificador é utilizado para efeitos de detecção de alterações no nome dos métodos no código fonte. Não tem qualquer efeito em termos de facilitar o processo de *Reverse Engineering* por parte da ferramenta.

Borland Together Control Center

```
package PacoteCliente;  
  
public class Classe_1 {  
}
```

```
package PacoteFornecedor;  
  
public class Classe_2 {  
}
```

As classes foram criadas dentro de directorias com os nomes dos pacotes e a respectiva indicação de pertença ao respectivo pacote foi adicionada no cabeçalho de cada classe. A relação de dependência indicada no modelo não gerou qualquer referência no código, nem sobre a forma de importação nem sobre a forma de um comentário específico da ferramenta. Verifica-se assim uma perda de informação durante o processo de geração de código. Os nomes das classes foram corrigidos uma vez que continham espaços, o que não é permitido pela linguagem Java, à semelhança do que acontece com as outras linguagens de programação, em que o carácter “espaço” é o delimitador natural dos identificadores. Os espaços nos nomes foram substituídos pelo carácter “_”. De referir que durante a fase de modelação foi feita uma tentativa de dar o nome a cada pacote da mesma forma que no estudo das ferramentas anteriores, ou seja, contendo um espaço. A ferramenta apresentou prontamente uma mensagem a indicar que tal não era permitido.

IV.1.2 – Reverse Engineering

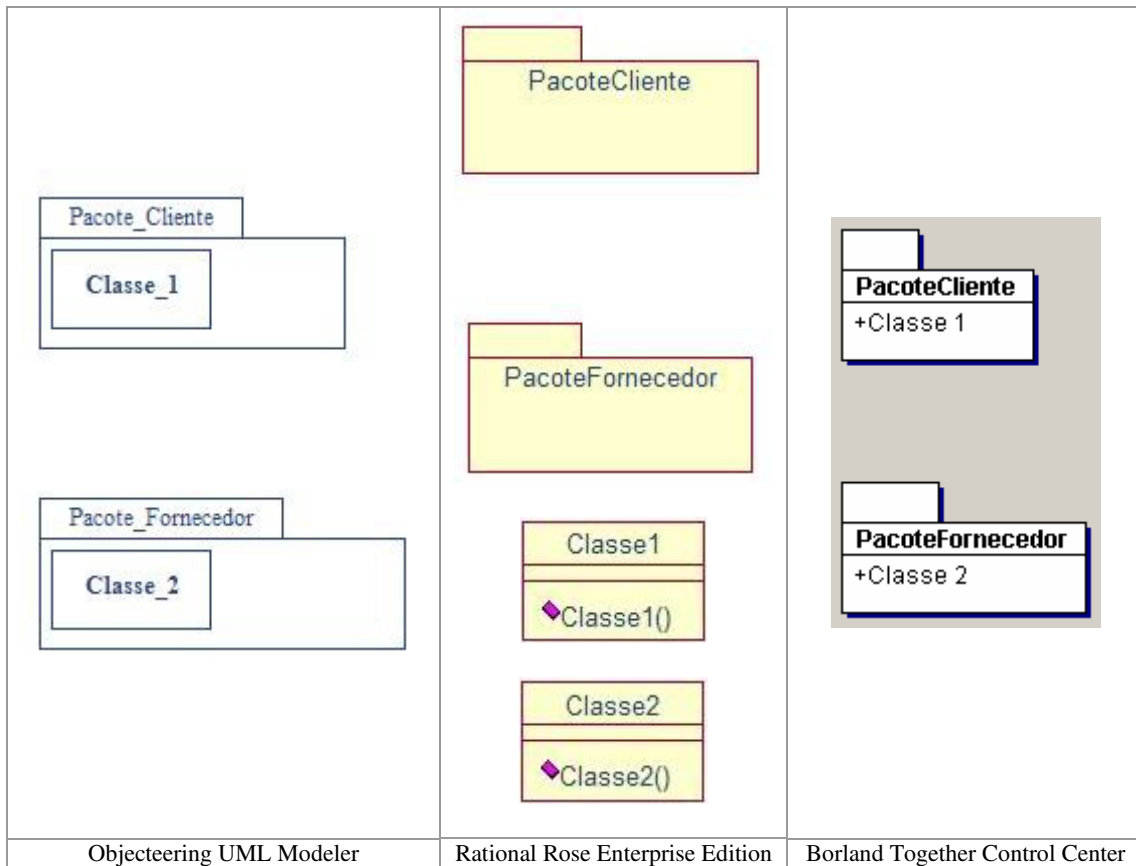


Figura 11 – Modelos resultantes para pacotes e interdependências

Objecteering UML Modeler

O modelo foi correctamente criado com os pacotes e respectivas classes. Como se pode observar perdeu-se no processo de *Reverse Engineering* a informação referente à importação existente no modelo original (relação de dependência). O processo de Reverse Engineering não teve em conta os “imports” existentes no código fonte.

Rational Rose Enterprise Edition

Verifica-se que o modelo obtido por *Reverse Engineering* contém a informação correcta acerca dos pacotes e das classes. Também os construtores que a ferramenta tinha gerado por omissão se encontram presentes no modelo. Uma vez que o código gerado pela ferramenta não tinha indicação da importação pretendida entre os pacotes, o modelo resultante não tem essa informação, como era esperado.

A ferramenta perdeu informação no processo de *Round-Trip Engineering* devido à geração de código (*Forward Engineering*), uma vez que o processo de *Reverse Engineering* não perdeu informação ao fazer o mapeamento entre o código fonte e o modelo apresentado nesta secção.

Borland Together Control Center

Observa-se que a ferramenta gerou correctamente o modelo em termos de pacotes e classes (com a respectiva inclusão das classes nos pacotes). Em termos de dependências, o modelo obtido não surpreende uma vez que, como referido anteriormente na análise do código fonte gerado, não existe nenhuma indicação acerca da dependência entre os dois pacotes. A ferramenta não guardou qualquer informação acerca da importação existente no modelo em estudo após o processo de *Forward Engineering*. À semelhança da ferramenta estudada anteriormente, o processo de *Round-Trip Engineering* perdeu informação devido ao *Forward Engineering*, uma vez que o processo de *Reverse Engineering* não originou perdas de informação.

IV.1.3 – Resumo dos resultados e conclusões



		Objecteering UML Modeler	Rational Rose Enterprise Edition	Borland Together Control Center
Forward Engineering	Criação das directorias correspondentes aos pacotes			
	Distribuição das classes pelos pacotes			
	Indicação de importação			
	Correcção de nomes ilegais			
Reverse Engineering	Distribuição das classes pelos pacotes			
	Geração da relação de dependência			
TOTAL		5/6	?/6	?/6

Tabela 1 – Tabela comparativa para pacotes e interdependências

Conclui-se assim que a ferramenta que melhor desempenhou a tarefa de efectuar o processo de *Round-Trip Engineering* sobre modelos com pacotes e interdependências foi o *Objecteering UML Modeler*. Esta ferramenta foi a única que gerou a indicação da importação pretendida, sendo esta a característica que a destacou das restantes ferramentas (na fase de *Forward Engineering*). Em termos de *Reverse Engineering* a prestação das ferramentas em estudo foi semelhante. É importante referir que mesmo as ferramentas que não geraram a anteriormente referida importação no código fonte foram analisadas em relação à existência de uma relação de importação no modelo resultante (que não poderá existir). Embora possa parecer que se está a penalizar injustamente as ferramentas, a verdade é que o erro que geraram numa fase se propagou para a outra, pelo que é importante realçar que a perda de informação não é de pouca importância.

IV.2 – Classes

Uma classe pode ser vista como um *template* para gerar objectos e a sua utilização é uma das concretizações do conceito de abstracção no paradigma da orientação por objectos. As classes descrevem objectos com propriedades (atributos) em comum e o comportamento de um objecto é descrito pelo conjunto de operações. Uma classe é representada em termos de diagrama como um rectângulo com três zonas, respectivamente contendo o nome da classe, seus atributos e uma lista de operações (*Figura 12*).

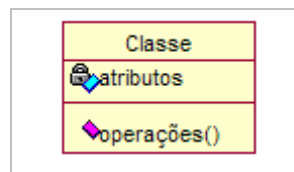


Figura 12 – Representação gráfica de uma classe

Nesta secção do relatório é feita a análise das ferramentas em estudo na perspectiva de analisar o seu comportamento em relação aos atributos e operações de cada classe. Fazendo parte da metodologia adoptada, foi utilizado o mesmo modelo em todas as ferramentas, de modo a permitir a melhor comparação possível entre as capacidades das mesmas. A *Figura 13* representa o modelo utilizado.

Objecteering UML Modeler	Rational Rose Enterprise Edition	Borland Together Control Center

Figura 13 – Modelo de teste para classes, atributos e operações

O modelo contém uma classe extremamente simples com três atributos e três operações. Mais uma vez não foi definido nenhum construtor de modo a verificar se as ferramentas definem um construtor por omissão nos casos em que este não está definido pela especificação durante a fase de desenho. Verifica-se que a ferramenta *Rational Rose Enterprise Edition* colocou um construtor na classe.

IV.2.1 – Forward Engineering

Objecteering UML Modeler

```
public class Circulo
{
    protected int centro;
    public int getCentro () {
        return this.centro;
    }

    protected float raio;
    public float getRaio () {
        return this.raio;
    }

    protected String cor;
    public String getCor () {
        return this.cor;
    }

    public void moverCirculo()
    {
    }

    public void crescerCirculo()
    {
    }

    public void alterarCorCirculo()
    {
    }
}
```

Como seria de esperar a classe foi criada com todos os atributos especificados no modelo. Todas as operações existentes no modelo estão definidas como sendo públicas (correcto, uma vez que o contrário não foi especificado na fase de modelação) e têm o tipo de retorno “void” (mais uma vez porque não foi especificado nenhum tipo de retorno no modelo). Para cada um dos atributos foi criada uma operação de retorno do respectivo valor, de modo a permitir o acesso externo a esses atributos. Estes selectores são gerados por omissão, havendo a possibilidade de desactivar esta funcionalidade. Por omissão não são gerados automaticamente modificadores.

Rational Rose Enterprise Edition

```
public class Circulo
{
    private int centro;
    private float raio;
    private String cor;

    /** @roseuid 3EDA0F7001D1 */
    public Circulo()
    {}
    /** @roseuid 3EDA0F4700B9 */
    public void moverCirculo()
    {}
    /** @roseuid 3EDA0F4D020C */
    public void crescerCirculo()
    {}
    /** @roseuid 3EDA0F570148 */
    public void alterarCorCirculo()
    {}
}
```

O código gerado pela ferramenta está de acordo com o modelo. Os atributos foram gerados com a indicação de que são elementos privados e com os tipos correctos. Também as operações foram correctamente geradas, com a indicação de que são elementos públicos e com o tipo de retorno “void”, uma vez que não tinha sido definido nenhum tipo de retorno durante a fase de modelação. Verifica-se ainda a existência do construtor que a ferramenta tinha gerado antes de efectuar o processo de *Forward Engineering*.

Borland Together Control Center

```
public class Circulo {
    public void moverCirculo() {
    }
    public void crescerCirculo() {
    }
    public void alterarCorCirculo() {
    }
    private int centro;
    private float raio;
    private String cor;
}
```

Verifica-se a correcção do processo de *Forward Engineering* uma vez que a ferramenta gerou os atributos e as operações com os tipos correctos e com as indicações de visibilidade que se encontravam definidas no modelo (atributos com visibilidade privada e operações com visibilidade pública).

IV.2.2 – Reverse Engineering

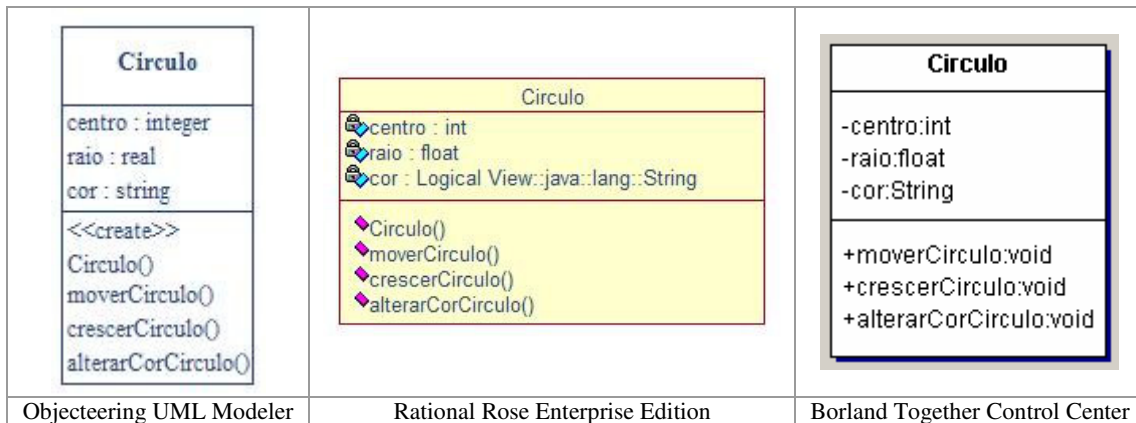


Figura 14 – Modelos resultantes para classes, atributos e operações

Objecteering UML Modeler

A classe encontra-se bem definida, com todos os atributos e operações existentes no modelo inicial. Os tipos encontram-se correctos e de acordo com a especificação. De referir que no processo de *Reverse Engineering* foi criado um construtor por omissão, uma vez que este não existia no código fonte. Os selectores que a ferramenta tinha gerado automaticamente durante o processo de *Forward Engineering* foram filtrados durante o processo de *Reverse Engineering*. Esta funcionalidade pode ser configurada na ferramenta, sendo este o comportamento por omissão.

Rational Rose Enterprise Edition

Verifica-se a correcção da ferramenta em efectuar *Reverse Engineering* ao código fonte anteriormente apresentado, tendo sido criado um modelo com a classe em causa com todos os atributos e operações existentes. Os tipos são os correctos de acordo com a especificação e as visibilidades encontram-se bem definidas.

De referir que o tipo de um dos atributos era “*String*”. No processo de *Reverse Engineering* a ferramenta indicou o endereço lógico completo para alcançar a dita classe: “*java.lang.String*”, uma vez que o projecto tinha sido declarado como sendo para desenvolvimento de código Java.

Borland Together Control Center

Como se observa pelo modelo acima apresentado a ferramenta efectuou com sucesso o processo de *Reverse Engineering*. Foram criados os atributos com os tipos concordantes com os existentes no código fonte e com as visibilidades correctas. As operações encontram-se também correctas e de acordo com a especificação, mantendo o tipo de retorno "void" como havia sido definido pela ferramenta durante a modelação e com as indicações de visibilidade correctas tal como presentes no código fonte.

IV.2.3 – Resumo dos resultados e conclusões




















		Objecteering UML Modeler	Rational Rose Enterprise Edition	Borland Together Control Center
Forward Engineering	Geração dos atributos e respectivos tipos			
	Geração das operações e respectivos tipos			
	Geração de um construtor por omissão			
	Geração de selectores por omissão			
Reverse Engineering	Geração dos atributos e respectivos tipos			
	Geração das operações e respectivos tipos			
	Geração de um construtor por omissão			
TOTAL		6/7	6/7	4/7

Tabela 2 – Tabela comparativa para classes, atributos e operações

Como se observa pela tabela comparativa acima apresentada, existem duas ferramentas que obtêm os melhores resultados, embora seja necessário analisar mais profundamente de modo a determinar a importância dos mesmos. Em relação à fase de *Forward Engineering*, a ferramenta *Objecteering UML Modeler* apenas não gerou um construtor por omissão, tendo efectuado todo o processo sobre o modelo em estudo com sucesso. Não se pode considerar este facto como uma incorrecção, uma vez que na fase de *Reverse Engineering* a ferramenta gerou um construtor. A ferramenta *Rational Rose Enterprise Edition* não gerou selectores por omissão, característica esta que não pode ser considerada como uma falha. A ferramenta *Borland Together Control Center* não gerou um construtor por omissão em nenhuma das fases do processo de *Round-Trip Engineering*. Também não gerou selectores por omissão. Esta foi a ferramenta pior classificada nesta fase do relatório, apesar de ter efectuado o processo com sucesso em termos das características mais importantes que se pretendiam testar.

IV.3 – Herança entre classes

O conceito de herança pode ser entendido como uma maneira de aumentar a reutilização, ao fazer uma classe partilhar operações e atributos de uma outra classe. Estas classes são denominadas respectivamente subclasse e super-classe. Para além de uma subclasse poder partilhar operações e atributos de uma super-classe pode ainda redefinir e acrescentar tanto operações como atributos, especializando assim a super-classe.

O modelo utilizado para testar as ferramentas em estudo em termos de *Round-Trip Engineering* sobre modelos que apresentem herança entre classes encontra-se representado na *Figura 15*.

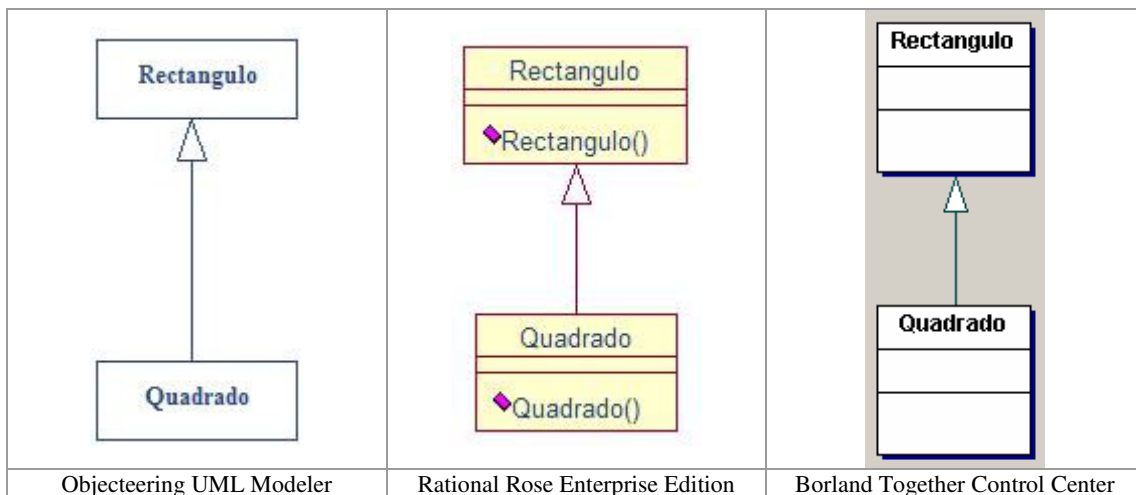


Figura 15 – Modelo de teste para herança entre classes

O modelo, contendo duas classes, apresenta uma relação de generalização definida entre elas. Pelo sentido da relação facilmente se verifica que a classe “Quadrado” é a subclasse e que a classe “Rectangulo” é a super-classe.

Verifica-se mais uma vez que a ferramenta *Rational Rose Enterprise Edition* gerou por omissão um construtor para ambas as classes.

IV.3.1 – Forward Engineering

Objecteering UML Modeler

```
public class Rectangulo
{
}
```

```
public class Quadrado
    extends Rectangulo
{
}
```

Como era esperado o código foi gerado sem problemas, tendo sido criadas as duas classes existentes no modelo e criada ainda a referência “extends”, indicando que a classe que a inclui estende a outra. Existe portanto a indicação que a classe “Quadrado” estende a classe “Rectangulo”.

Rational Rose Enterprise Edition

```
public class Rectangulo
{
    /** @roseuid 3EDA146D0155 */
    public Rectangulo()
    {
    }
}
```

```
public class Quadrado extends Rectangulo
{
    /** @roseuid 3EDA146D026D */
    public Quadrado()
    {
    }
}
```

Verifica-se a correcção do código fonte gerado uma vez que foram criadas duas classes, uma das quais – a subclasse “Quadrado” – com a indicação que estende a outra – a super-classe “Rectangulo”. Foi gerado código para os construtores definidos pela ferramenta (por omissão).

Borland Together Control Center

```
public class Rectangulo {
}
```

```
public class Quadrado extends Rectangulo {
}
```

O processo de *Forward Engineering* decorreu sem problemas tendo sido gerado código fonte de acordo com a especificação. A classe “Quadrado” tem a indicação de que “estende” a classe “Rectângulo”, o que reflecte correctamente a informação contida no modelo.

IV.3.2 – Reverse Engineering

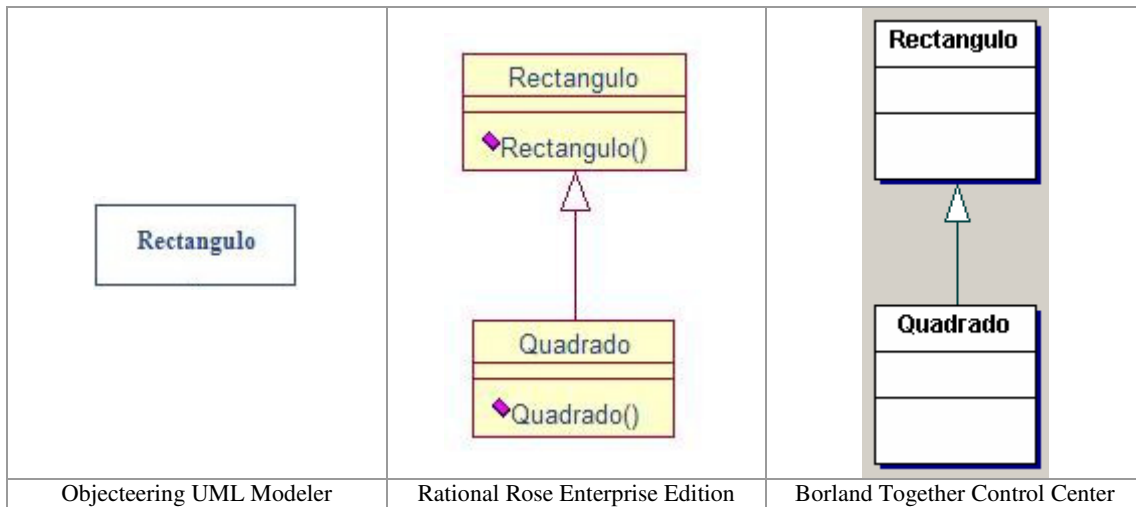


Figura 16 – Modelos resultantes para herança entre classes

Objectteering UML Modeler

A aplicação não efectuou *Reverse Engineering* sobre o código fonte, apenas conseguindo gerar a parte do modelo referente à classe “Rectangulo”. A classe “Quadrado” não foi incluída no modelo, podendo-se tirar a conclusão de esse facto se deveu à linha do código fonte onde se faz a referência de que a classe estende a outra. Foram feitas diversas tentativas de resolver o problema (inclusivamente contactar os serviços técnicos responsáveis pela ferramenta), mas todas sem sucesso, pelo que não foi de facto possível efectuar esta fase do estudo no elemento UML referente a esta secção do relatório.

Rational Rose Enterprise Edition

Como é visível o modelo é em tudo semelhante ao utilizado inicialmente, apresentando as duas classes existentes e a relação de generalização no sentido da subclasse para a super-classe. Verifica-se ainda que existe um construtor em ambas as classes, tendo estes sido anteriormente gerados pela ferramenta durante a geração de código. A ferramenta realizou o processo de *Round-Trip Engineering* sem qualquer perda de informação.

Borland Together Control Center

O modelo é igual ao apresentado inicialmente, contendo duas classes e a relação de generalização entre elas. A ferramenta efectuou com sucesso o processo de *Round-Trip Engineering* sobre o modelo em causa.

IV.3.3 – Resumo dos resultados e conclusões







		Objecteering UML Modeler	Rational Rose Enterprise Edition	Borland Together Control Center
Forward Engineering	Geração da “extensão” pretendida entre as classes			
Reverse Engineering	Indicação da herança entre as classes			
	TOTAL	?/2	2/2	2/2

Tabela 3 – Tabela comparativa para herança entre classes

Em termos de *Forward Engineering* todas as ferramentas em estudo se comportaram como esperado, uma vez que geraram a indicação de que uma classe “estende” a outra, de acordo com o modelo original. Em relação ao processo de *Reverse Engineering* apenas duas das ferramentas puderam ser testadas, uma vez que o *Objecteering UML Modeler* não realizou este processo. As restantes ferramentas realizaram todo o processo de *Round-Trip Engineering* com sucesso, sem qualquer perda de informação.

IV.4 – Visibilidade

Existem três tipos de visibilidade de atributos e operações: pública, protegida e privada.

A visibilidade **pública** (*public*), como o nome indica, permite ao elemento (válido tanto para atributos, como para operações) que a possui poder ser referenciado por qualquer outro elemento, independentemente da sua localização. Esta definição é válida tanto em UML como em Java. Um elemento com este tipo de visibilidade é prefixado com o símbolo “+”.

No caso de um elemento possuir visibilidade **protegida** (*protected*), o acesso ao mesmo está limitado a instâncias da própria classe e seus descendentes (por herança). A definição acima indicada é válida para UML que tem o mesmo significado em Java se for utilizado o classificador de visibilidade “*private protected*”. O classificador “*protected*” em Java estende a definição em UML na medida em que a visibilidade se estende a todas as classes definidas dentro do mesmo pacote. Em termos de simbologia, um elemento com este tipo de visibilidade é prefixado com “#”.

Pelo contrário, se um elemento tiver visibilidade **privada** (*private*), apenas as instâncias da própria classe o podem usar. Esta definição é válida tanto para UML como para Java. Se um elemento tiver visibilidade privada é prefixado com o símbolo “-”.

Para testar as capacidades das ferramentas em estudo ao realizar o processo de *Round-Trip Engineering* sobre modelos com elementos de diferentes visibilidades foi produzido o modelo de teste representado na *Figura 17*, onde existem exemplos para todos os tipos de visibilidade existentes. Foi definido um construtor explicitamente no modelo com o nome “abrirConta” e com o respectivo estereotipo. Na linguagem Java o construtor de uma classe deve ter o nome da mesma. Pretende-se assim verificar o comportamento das ferramentas em relação a modelos que contenham classes com um construtor definido que não obedeça a essa regra.



Figura 17 – Modelo de teste para visibilidade de atributos e operações

É importante referir que no caso do modelo de teste apresentado para a ferramenta *Objecteering UML Modeler*, o tipo do atributo “saldo” é “real” em vez de ser “double” como nas outras ferramentas. Este facto deve-se a uma característica da ferramenta que apenas permite escolher o tipo dos atributos a partir de uma lista pré-definida. Esta questão não é relevante para efeitos do estudo pretendido nesta fase do relatório, uma vez que não afecta a análise da ferramenta.

Na ferramenta *Borland Together Control Center* o construtor apresenta o tipo de retorno “void”. Sendo um construtor não deveria ter qualquer tipo de retorno mas a ferramenta automaticamente colocou esse tipo por omissão.

Verifica-se que os estereótipos utilizados para os construtores não são iguais nas três ferramentas, uma vez que o *Borland Together Control Center* utiliza o “<<constructor>>” por oposição ao “<<create>>” utilizado pelos restantes.

IV.4.1 – Forward Engineering

Objecteering UML Modeler

```
public class Conta
{
    private float saldo;
    public float getSaldo () {
        return this.saldo;
    }

    private String dataAbertura;
    public String getDataAbertura () {
        return this.dataAbertura;
    }

    private String dataUltimaOp;
    public String getDataUltimaOp () {
        return this.dataUltimaOp;
    }

    public Conta()
    {
    }

    public void consultarSaldo()
    {
    }

    public void debitarMontante()
    {
    }

    public void creditarMontante()
    {
    }

    protected void consultarDatas()
    {
    }

    private void alterarSaldo()
    {
    }
}
```

O código acima apresentado está de acordo com o modelo. Todas as visibilidades estão correctamente indicadas tal como foram definidas durante a fase de desenho. Os tipos dos atributos e de retorno das operações encontram-se correctas. O tipo do atributo “saldo” passou de “*real*” para “*float*”, de modo a adaptar o modelo à linguagem Java. É importante referir que apesar de ter sido definido um construtor (“abrirConta”, com o estereotipo “<<create>>”), a ferramenta definiu um construtor com o nome da classe. O construtor anteriormente definido foi eliminado.

Rational Rose Enterprise Edition

```
public class Conta
{
    private Double saldo;
    private String dataAbertura;
    private String dataUltimaOp;

    /** @roseuid 3EDA318300E4 */
    public Conta()
    {
    }

    /** @roseuid 3EDA30BB019A */
    public void consultarSaldo()
    {
    }

    /** @roseuid 3EDA30C001E8 */
    private void alterarSaldo()
    {
    }

    /** @roseuid 3EDA30E4037A */
    public void debitarMontante()
    {
    }

    /** @roseuid 3EDA30EA0301 */
    public void creditarMontante()
    {
    }

    /** @roseuid 3EDA30F503CF */
    protected void consultarDatas()
    {
    }

    /** @roseuid 3EDA30FA02FA */
    public void abrirConta()
    {
    }
}
```

Verifica-se que o código está de acordo com a informação contida no modelo. Tanto os atributos como as operações têm a visibilidade correcta e de acordo com a especificação. Apenas no construtor existe uma irregularidade, uma vez que considerou o construtor definido no modelo (“abrirConta”) como sendo uma operação com o tipo de retorno “void”. Foi gerado um construtor com o nome da classe.

Borland Together Control Center

```
public class Conta {
    public void consultarSaldo() {
    }

    private void alterarSaldo() {
    }

    public void debitarMontante() {
    }

    public void creditarMontante() {
    }

    protected void consultarDatos() {
    }

    /**
     * @stereotype constructor
     */
    public void abrirConta() {
    }

    private double saldo;
    private String dataAbertura;
    private String dataUltimaOp;
}
```

Mais uma vez se verifica que a ferramenta efectuou o processo de *Forward Engineering* com sucesso, sendo que todas as visibilidades se encontram de acordo com o que foi definido no modelo. Em relação ao construtor definido no modelo de teste verifica-se que a ferramenta o colocou como sendo uma operação (uma vez que existe um tipo de retorno), mas colocou sobre a forma de comentário da linguagem Java a indicação de que tem o estereotipo “*constructor*”. Não foi gerado um construtor por parte da ferramenta.

IV.4.2 – Reverse Engineering

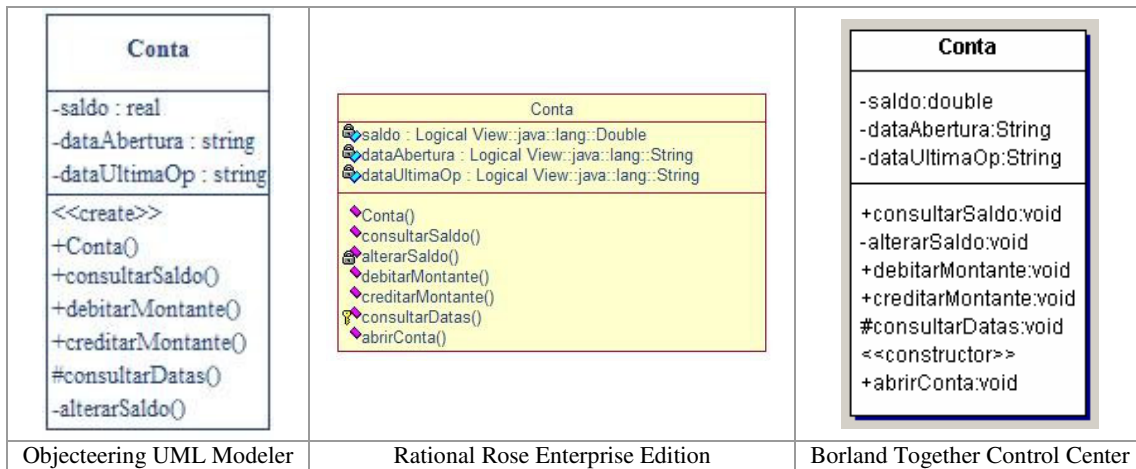


Figura 18 – Modelos resultantes para visibilidade de atributos e operações

Objectteering UML Modeler

Verifica-se que todas as visibilidades se encontram correctas e de acordo com as definidas no modelo inicial. A classe é em tudo semelhante à presente no modelo definido inicialmente, à excepção do construtor que tem o nome da classe em vez do nome que tinha sido definido no modelo de teste. O modelo resultante não apresenta perda nem alteração de informação em relação ao código fonte. Considera-se que o processo de *Round-Trip Engineering* foi concluído com sucesso.

Rational Rose Enterprise Edition

O modelo está de acordo com o código a partir do qual foi gerado e apresenta-se semelhante ao modelo original. Em termos de visibilidade todos os atributos e operações foram definidos como no modelo inicial. Perdeu-se informação acerca do construtor, uma vez que este existe no código fonte (com o nome da classe) e após o processo de *Reverse Engineering* não existe a indicação de que é um construtor. Verifica-se que a ferramenta colocou o *path* completo da classe correspondente para representar os tipos dos atributos, uma vez que o projecto tinha sido definido como sendo orientado para desenvolvimento de código Java.

Borland Together Control Center

Verifica-se que as visibilidades dos vários elementos existentes no modelo se encontram de acordo com a informação contida no código fonte. O modelo é em tudo semelhante ao modelo utilizado originalmente, pelo que se conclui que o processo de *Reverse Engineering* foi efectuado com sucesso. Em relação ao construtor, existe a indicação no modelo resultante de que existe com o nome “abrirConta”, apesar de na realidade no código não representar um construtor (uma vez que tem um tipo de retorno). A ferramenta guardou a informação de que esta operação tinha esse estereotipo (através da utilização de um comentário da linguagem Java no nível de abstracção de código fonte) durante todo o processo de *Round-Trip Engineering*.

IV.4.3 – Resumo dos resultados e conclusões
















		Objecteering UML Modeler	Rational Rose Enterprise Edition	Borland Together Control Center
Forward Engineering	Indicação da visibilidade dos atributos			
	Indicação da visibilidade das operações			
	Geração correcta do construtor			
Reverse Engineering	Indicação da visibilidade dos atributos			
	Indicação da visibilidade das operações			
TOTAL		5/5	4/5	4/5

Tabela 4 – Tabela comparativa para visibilidade de atributos e operações

Como se observa, todas as ferramentas efectuaram com sucesso o processo de *Round-Trip Engineering* sobre o modelo de teste em termos de visibilidade de atributos e operações. Em relação à geração correcta do construtor, apenas a ferramenta *Objecteering UML Modeler* se comportou como esperado, uma vez que substituiu o construtor incorrectamente criado no modelo original por outro com o nome da classe no código fonte. A ferramenta *Rational Rose Enterprise Edition* criou um construtor com o nome da classe, apesar de ter mantido o que tinha sido definido no modelo original como sendo uma operação. O *Borland Together Control Center* não gerou nenhum construtor no código fonte, tendo mantido o que tinha sido criado no modelo com uma indicação em comentário de que este é um construtor.

IV.5 – Associações Simples

Uma relação de associação pretende especificar quais os objectos de uma classe que podem estar ligados a objectos de outra classe.

Uma associação é representada em UML por uma linha a cheio com um conjunto de informações associado que especifica:

- ☞ Nome da associação
- ☞ Papel de cada participante (classe)
- ☞ Multiplicidade de cada participante na associação

A multiplicidade indica o número de instâncias de uma classe que se podem relacionar através da associação em questão com uma instância da outra classe participante. A multiplicidade pode ser referida definindo um valor específico ou uma gama de valores.

Existem três tipos de associações:

- ☞ Associação simples
- ☞ Agregação
- ☞ Composição

Esta secção do relatório pretende analisar as ferramentas em estudo para o caso das associações simples. Considere-se o seguinte exemplo de relação simples entre duas classes, representado na *Figura 19*.

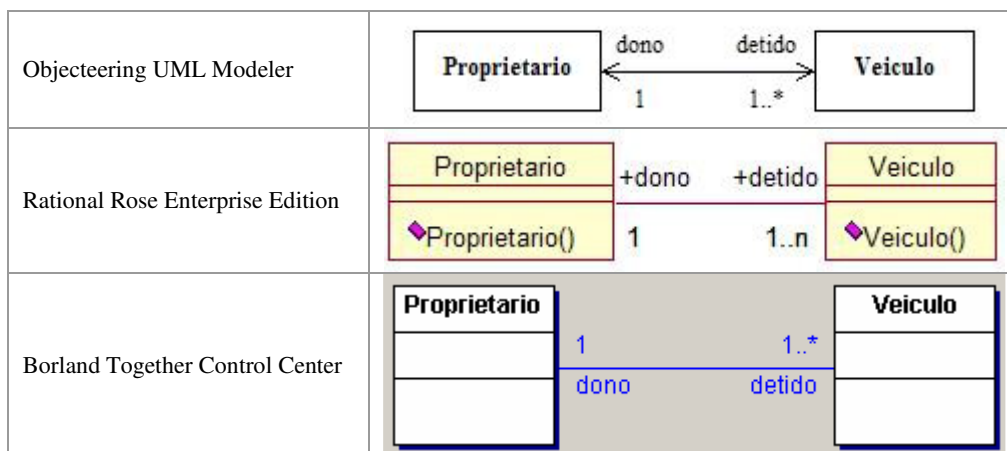


Figura 19 – Modelo de teste para associações simples

É importante referir que o modelo criado na ferramenta *Objecteering UML Modeler* não é exactamente igual (apenas em termos de representação gráfica) aos modelos criados nas outras ferramentas. As setas que surgem nas extremidades da associação indicam que esta é navegável em ambos os sentidos. As restantes ferramentas representam a navegabilidade em ambos os sentidos por uma linha simples, sem setas.

Mais uma vez se verifica que a ferramenta *Rational Rose Enterprise Edition* criou um construtor por omissão em ambas as classes.

(Esta página foi intencionalmente deixada em branco)

IV.5.1 – Forward Engineering

Objecteering UML Modeler

```
import java.util.*;

public class Proprietario
{
    protected Vector detido = new Vector();
    public Veiculo getDetido (int i) {
        return (Veiculo)this.detido.elementAt(i);
    }
    public int cardDetido () {
        return this.detido.size();
    }
}
```

```
public class Veiculo
{
    Proprietario dono;
    Proprietario getDono () {
        return this.dono;
    }
    int cardDono () {
        if ( this.dono == null ) return 0;
        else return 1;
    }
}
```

O código gerado apresenta-se correcto e de acordo com o modelo inicial. Na classe “Proprietario” verifica-se a existência de um objecto da classe “Vector”, destinado a guardar referências para objectos da classe “Veiculo”. Existem ainda operações geradas automaticamente pela ferramenta que permitem utilizar o dito vector com facilidade, existindo um método para obter um elemento do vector dada a sua posição no mesmo e outro que permite obter o seu tamanho, ou seja, o número de elementos que contem. Verifica-se que na classe “Veiculo” existe uma referência para um objecto da classe “Proprietario”, o que está de acordo com o que havia sido definido no modelo. Foram geradas automaticamente operações para retornar e verificar a existência da referência para o objecto da classe “Proprietario”. Os nomes dos papéis dos participantes na associação foram mantidos no código gerado.

Rational Rose Enterprise Edition

```
public class Proprietario
{
    public Veiculo detido[];

    /** @roseuid 3EBBBB040391 */
    public Proprietario()
    {
    }
}
```

```
public class Veiculo
{
    public Proprietario dono;

    /** @roseuid 3EBBBB050389 */
    public Veiculo()
    {

    }
}
```

Como se observa no código acima apresentado a classe “Proprietario” declara um vector de referências para objectos do tipo “Veiculo”. Ao contrário da ferramenta estudada anteriormente, esta não criou um objecto do tipo “Vector”, tendo apenas declarado a existência de um vector. Na classe “Veiculo” existe uma referência para um objecto do tipo “Proprietario”, de acordo com a especificação. O código fonte apresenta os nomes dos papéis dos participantes na associação tal como definido no modelo original.

Borland Together Control Center

```
public class Proprietario {
    /**
     * @clientCardinality 1
     * @supplierCardinality 1..*
     * @clientRole dono
     * @supplierRole detido
     */
    private Veiculo lnkVeiculo;
}
```

```
public class Veiculo {}
```

O código gerado por esta ferramenta não pode ser comparado directamente com o das restantes uma vez que toda a informação relevante existente no modelo foi guardada como um comentário da linguagem Java. Não foi assim gerado código fonte de acordo com o modelo, tendo apenas sido criada uma referência na classe “Proprietario” para um objecto do tipo “Veiculo”, o que não está correcto de acordo com as cardinalidades definidas no modelo original. No comentário criado pela ferramenta verifica-se a indicação das cardinalidades e dos papéis dos intervenientes na associação tal como definidos no modelo inicial. Esta solução de guardar a informação na forma de comentário poderia ser considerada correcta se tivesse sido criada uma referência na classe “Veiculo” para um objecto da classe “Proprietario” e se nesta existisse apenas um comentário a indicar a utilização de um vector. Assim ficaria ao cargo do implementador escolher a melhor solução de programar esta associação.

IV.5.2 – Reverse Engineering

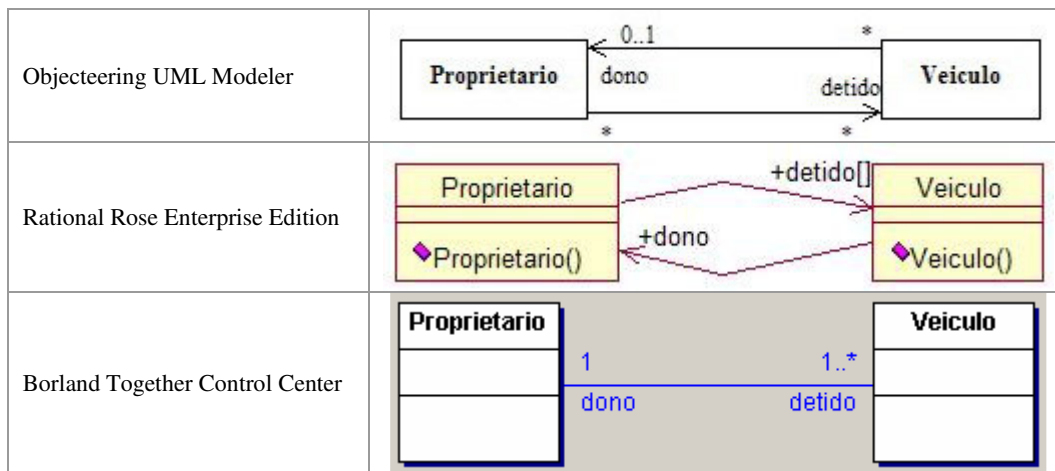


Figura 20 – Modelos resultantes para associações simples

Objectteering UML Modeler

O modelo apresentado apresenta duas associações, em vez de apenas uma de acordo com o modelo inicial. Os nomes dos papéis das classes envolvidas na associação foram mantidos no processo de *Reverse Engineering*. As operações que tinham sido criadas por omissão durante o processo de *Forward Engineering* foram filtradas no processo de *Reverse Engineering* por omissão, existindo a possibilidade de configurar essa funcionalidade na ferramenta. Em termos de cardinalidades verifica-se que estão de acordo com o definido no modelo original, embora possa não parecer. Tendo sido criadas duas associações, cada uma com apenas um sentido de navegabilidade, é importante analisar as cardinalidades individualmente. Na associação definida no sentido da classe “Proprietario” para a classe “Veiculo” verifica-se que a cardinalidade é “vários” (*), o que não está de acordo com o existente no modelo, por se ter perdido a informação de que tem que existir pelo menos um (“1..*”). No sentido inverso a associação indica que a cardinalidade é “0..1”, tendo sido perdida a informação de que existe uma e uma só referência para um objecto do tipo “Proprietario” dentro da classe “Veiculo”.

Rational Rose Enterprise Edition

Foram criadas duas associações por esta ferramenta, com os nomes correctos dos papéis que tinham sido definidos no modelo inicial. Perdeu-se completamente a

informação referente às cardinalidades, uma vez que estas nem se encontram definidas no modelo resultante do processo de *Reverse Engineering*.

Borland Together Control Center

O modelo resultante para esta ferramenta encontra-se bem definido e de acordo com o modelo original, sendo que não existiu perda de informação em todo o processo de *Round-Trip Engineering*. É importante repetir que a informação guardada no nível de abstracção de código fonte não é guardada como código mas sim na forma de um comentário na linguagem Java (específico da ferramenta). Isto implica que não se possa realizar o processo de *Reverse Engineering* por outra ferramenta, o que limita a abordagem “*best-of-breed*” já anteriormente mencionada.

IV.5.3 – Resumo dos resultados e conclusões

		Objecteering UML Modeler	Rational Rose Enterprise Edition	Borland Together Control Center
Forward Engineering	Geração das cardinalidades			
	Manutenção dos nomes dos papéis dos participantes na relação			
	Geração de selectores para navegação nas associações			
	Geração de selectores para a cardinalidade das associações			
Reverse Engineering	Indicação das cardinalidades			
	Manutenção dos nomes dos papéis dos participantes na relação			
	Não desdobramento das associações			
TOTAL		5/7	3/7	?/7

Tabela 5 – Tabela comparativa para associações simples

Tanto a ferramenta *Objecteering UML Modeler* como a *Rational Rose Enterprise Edition* realizaram com sucesso o processo de *Forward Engineering*. A outra ferramenta - *Borland Together Control Center* – guardou toda a informação relevante na forma de comentário, em vez de gerar código fonte. Não se pode comparar assim com as restantes ferramentas de forma justa. Apenas a ferramenta *Objecteering UML Modeler* gerou selectores de navegação e de cardinalidade. Em termos de *Reverse Engineering*, apenas a ferramenta *Borland Together Control Center* foi capaz de guardar a informação correcta acerca das cardinalidades. Esta foi também a única que não desdobrou a associação. Todas as ferramentas, guardaram a informação acerca dos papéis dos participantes na relação tal como definido no modelo original.

IV.6 – Agregações

No caso das associações ditas “normais”, ambas as classes participantes se encontram num mesmo nível conceptual. Uma agregação (também referida na literatura como agregação simples) indica que uma instância de uma classe possui uma ligação mais forte, do tipo “do todo para as partes”.

Em termos de representação, uma agregação é representada usando um losango não preenchido, colocado junto à classe que representa o agregado. O modelo utilizado nesta fase da análise foi o representado na *Figura 21*.

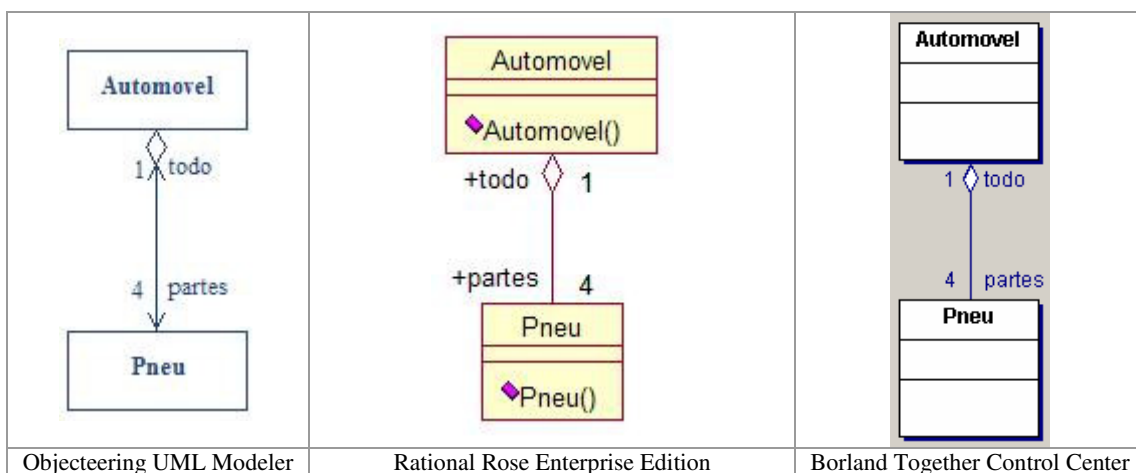


Figura 21 – Modelo de teste para agregações

O exemplo anteriormente apresentado deve ser interpretado da seguinte forma: um automóvel tem quatro pneus (estes fazem parte do automóvel) e cada pneu está montado num só automóvel (é um componente ou “parte” de um automóvel).

No caso do modelo apresentado para a ferramenta *Objectteering UML Modeler* verifica-se que existem duas setas nas extremidades da linha que representa a associação (agregação). É apenas uma indicação que a agregação é navegável em ambos os sentidos, tal como foi definido nas outras ferramentas.

Verifica-se mais uma vez que a ferramenta *Rational Rose Enterprise Edition* criou um construtor por omissão em cada uma das classes.

IV.6.1 – Forward Engineering

Objecteering UML Modeler

```
import java.util.*;

public class Automovel
{
    protected Vector partes = new Vector(4);
    public Pneu getPartes (int i) {
        return (Pneu)this.partes.elementAt(i);
    }
    public int cardPartes () {
        return this.partes.size();
    }
}
```

```
public class Pneu
{
    Automovel todo;
    Automovel getTodo () {
        return this.todo;
    }
    int cardTodo () {
        if ( this.todo == null ) return 0;
        else return 1;
    }
}
```

O código gerado pela ferramenta é semelhante ao código que seria gerado se a associação existente entre as classes fosse uma associação simples. Não existe qualquer indicação de que se trata de uma agregação. Foi criado um objecto do tipo “Vector” para guardar as referências para objectos do tipo “Pneu” dentro da classe “Automovel”. Este vector foi correctamente dimensionado na sua criação, como a indicação de que existem quatro referências para objectos do tipo “Pneu”. Como na análise anteriormente efectuada para o caso das associações simples, a ferramenta criou uma operação para obter uma determinada referência dada a sua posição no vector e outra operação para obter a dimensão do vector. A classe “Pneu” tem a referência de um objecto do tipo “Automovel”, o que está de acordo com a especificação. Foram geradas operações para obter a referência para o objecto do tipo “Automovel” e para verificar a existência dessa referência.

Rational Rose Enterprise Edition

```
public class Automovel
{
    public Pneu partes[];

    /** @roseuid 3EBBE60601CF */
    public Automovel()
    {
    }
}
```

```
public class Pneu
{
    public Automovel todo;

    /** @roseuid 3EBBE60602A2 */
    public Pneu()
    {
    }
}
```

Tal como no caso da ferramenta estudada anteriormente não existe a indicação de que a associação em questão é uma agregação. Como tal, a ferramenta lidou com o modelo em estudo como se se tratasse de uma associação simples. Gerou a indicação de que a classe “Automovel” usa um vector para guardar referências para objectos da classe “Pneu”, embora tenha perdido a informação da dimensão existente no modelo. A classe “Pneu” tem a indicação de uma referência para um objecto do tipo “Automovel”. Ambas as classes têm o construtor que foi criado pela ferramenta antes do processo de *Forward Engineering*.

Borland Together Control Center

```
public class Automovel {
    /**
     * @link aggregation
     * @clientCardinality 1
     * @supplierCardinality 4
     * @clientRole todo
     * @supplierRole partes
     */
    private Pneu lnkPneu;
}
```

```
public class Pneu {
}
```

A ferramenta gerou código usando a técnica de colocar a informação relevante para realizar posteriormente o processo de *Reverse Engineering* dentro de um comentário Java. O código apenas tem a indicação de que a classe “Automovel” tem uma ligação com a classe “Pneu”, mas a cardinalidade não se encontra correcta de acordo com o modelo. Em termos do comentário gerado, verifica-se que toda a informação relevante se encontra presente, tendo o tipo de associação (agregação), as cardinalidades e os nomes dos papéis dos participantes na associação.

IV.6.2 – Reverse Engineering

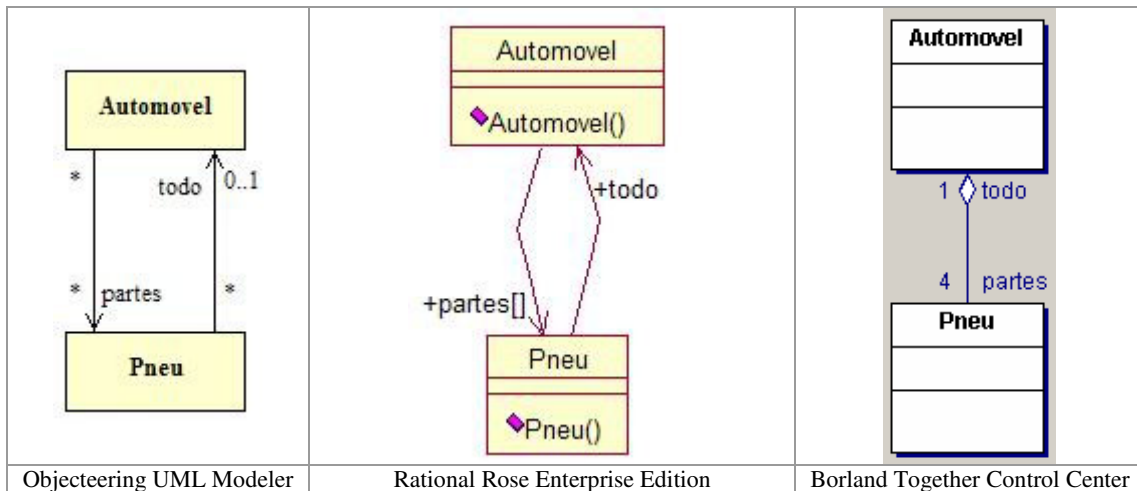


Figura 22 – Modelos resultantes para agregações

Objectteering UML Modeler

O modelo acima indicado apresenta duas associações, em vez de uma agregação como no modelo inicial. Este facto não surpreende uma vez que essa perda de informação já havia sido detectada no código gerado pelo processo de *Forward Engineering*. Verifica-se a existência de informação correcta acerca dos nomes dos papéis dos intervenientes na relação. Em termos de cardinalidades, na associação que tem o sentido da classe “Pneu” para a classe “Automovel”, verifica-se que se perdeu informação referente a que deve existir pelo menos uma referência, passando assim de “1” para “0..1”. Não se pode considerar como uma falha, pois essa informação deveria ser gerida pelo construtor. Na associação com o sentido de navegabilidade inverso verifica-se que se perdeu completamente a informação existente no código gerado, que indicava que um objecto da classe “Automovel” guarda quatro referências para objectos da classe “Pneu”.

Rational Rose Enterprise Edition

Verifica-se que a ferramenta perdeu completamente a informação relativa às cardinalidades, uma vez que estas nem se encontram definidas no modelo resultante do processo de *Reverse Engineering*. Perdeu-se a informação relativa ao tipo de associação presente, mas esse facto já era esperado uma vez que a dita informação não se encontrava presente no código fonte. Os nomes dos papéis dos

intervenientes na relação foram mantidos. Verifica-se ainda a existência dos construtores por omissão que a ferramenta criou no modelo inicial.

Borland Together Control Center

O modelo apresentado é igual ao modelo inicial, apesar do facto de que a ferramenta não guardou a informação da maneira desejada, ou seja, gerando código fonte. Ao ter guardado a informação na forma de um comentário Java de formato próprio da ferramenta, não se torna possível efectuar o processo de *Reverse Engineering* por outra ferramenta, para além do facto de não ter facilitado o desenvolvimento de código fonte por não ter gerado código fonte a partir do modelo.

IV.6.3 – Resumo dos resultados e conclusões




		Objecteering UML Modeler	Rational Rose Enterprise Edition	Borland Together Control Center
Forward Engineering	Geração das cardinalidades			
	Manutenção dos nomes dos papéis dos participantes na relação			
	Geração de selectores para navegação nas associações			
	Geração de selectores para a cardinalidade das associações			
Reverse Engineering	Não desdobramento das associações (Geração de uma agregação)			
	Indicação das cardinalidades			
	Manutenção dos nomes dos papéis dos participantes na relação			
TOTAL		5/7	2/7	?/7

Tabela 6 – Tabela comparativa para agregações

Como é visível pela tabela acima apresentada, apenas a ferramenta *Objecteering UML Modeler* conseguiu realizar o processo de *Forward Engineering* com sucesso, sem perder informação. Esta foi a única ferramenta que gerou selectores de navegação e de cardinalidade. A ferramenta *Rational Rose Enterprise Edition* perdeu informação acerca das cardinalidades. Em relação à ferramenta *Borland Together Control Center* verifica-se que a informação foi armazenada mas na forma de comentário. Esta última foi a única que conseguiu efectuar com sucesso o processo de *Reverse Engineering*, uma vez que as restantes não geraram o modelo final com uma agregação nem guardaram informação correcta acerca das cardinalidades. Todas as ferramentas mantiveram a informação correcta acerca dos nomes dos papéis dos participantes na relação.

IV.7 – Composições

Uma composição (ou também referida na literatura como agregação composta) acrescenta à definição de agregação a noção de que os “componentes” (partes do todo) têm uma forte pertença ao “agregado” de tal forma que as partes não podem existir sem o todo. Têm assim um tempo de vida delimitado, e o “todo” é responsável pela criação e destruição das suas “partes”. Graficamente, uma composição é representada com um losango a cheio que é colocado junto da classe que representa o agregador (o “todo”). O modelo utilizado para testar as capacidades das ferramentas em estudo em realizar o processo de *Round-Trip Engineering* sobre modelos com composições encontra-se representado na *Figura 23*.

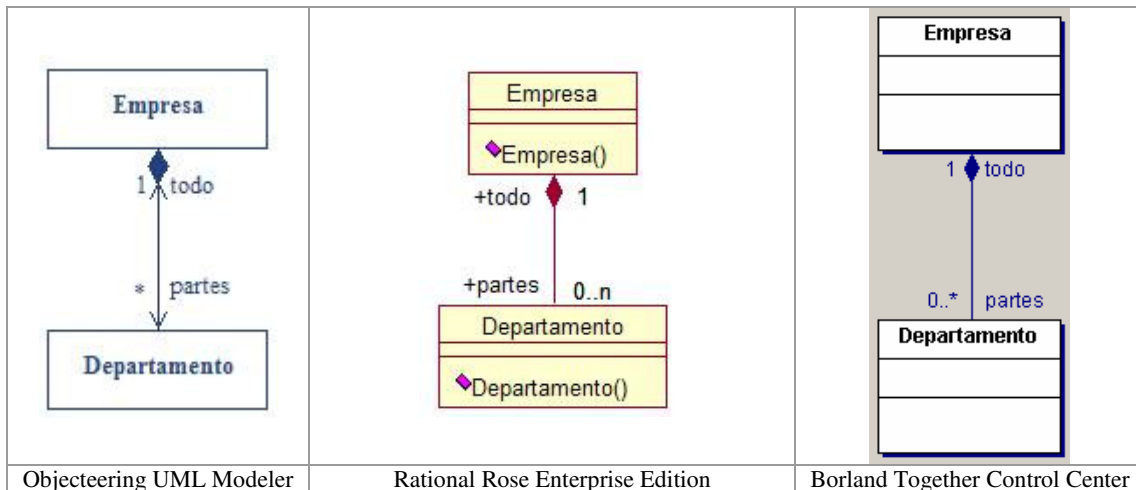


Figura 23 – Modelo de teste para composições

O exemplo anteriormente apresentado deve ser interpretado da seguinte forma: uma empresa é composta por vários departamentos, os quais não fazem sentido se não existir a empresa.

O modelo apresentado pela ferramenta *Objecteering UML Modeler* tem a associação representada graficamente com duas setas pelo simples facto de ser esta a representação de uma associação navegável em ambos os sentidos. O modelo é igual aos apresentados pelas outras ferramentas.

Tal como nos outros modelos apresentados anteriormente neste relatório, verifica-se que a ferramenta *Rational Rose Enterprise Edition* coloca um construtor por omissão em todas as classes.

IV.7.1 – Forward Engineering

Objecteering UML Modeler

```
public class Empresa
{
    protected Vector partes = new Vector();
    public Departamento getPartes (int i) {
        return (Departamento)this.partes.elementAt(i);
    }
    public int cardPartes () {
        return this.partes.size();
    }
}
```

```
public class Departamento
{
    Empresa todo;
    Empresa getTodo () {
        return this.todo;
    }
    int cardTodo () {
        if ( this.todo == null ) return 0;
        else return 1;
    }
}
```

À semelhança do que aconteceu no estudo das agregações, verifica-se que a ferramenta gerou código para o modelo em questão como se a associação presente fosse uma associação simples, enquanto na realidade é uma composição. Assim, verifica-se que existe um objecto do tipo “Vector” para guardar referências para objectos do tipo “Departamento” dentro da classe “Empresa”. Tal como anteriormente, a ferramenta gerou operações que facilitam o acesso ao dito vector. A classe “Departamento” tem uma referência para um objecto do tipo “Empresa”, o que está de acordo com o modelo original. A informação de que teria sempre que ter uma referência para um objecto do tipo “Empresa” deveria ser gerido pelo construtor. Tem ainda as habituais operações que facilitam o manuseamento da referência anteriormente mencionada. É importante ainda referir que o código gerado tem a indicação dos nomes dos papéis dos participantes na relação, tal como definidos no modelo.

Rational Rose Enterprise Edition

```
public class Empresa
{
    public Departamento partes[];

    /** @roseuid 3EBBF78C018A */
    public Empresa()
    {
    }
}
```

```
public class Departamento
{
    public Empresa todo;

    /** @roseuid 3EBBF78C0266 */
    public Departamento()
    {
    }
}
```

O código apresentado não possui qualquer indicação que a relação existente é uma composição. A composição existente no modelo inicial é assim tratada como uma associação simples. Assim sendo existe a indicação de que é utilizado um vector de referências para objectos do tipo “Departamento” dentro da classe “Empresa”. Na classe “Departamento” existe a referência para um objecto do tipo “Empresa”, tal como presente na especificação. Verifica-se ainda a existência dos construtores automaticamente gerados pela ferramenta. Os nomes dos papéis dos participantes na relação de composição foram mantidos.

Borland Together Control Center

```
public class Empresa {
    /**
     * @link aggregationByValue
     * @clientCardinality 1
     * @supplierCardinality 0..*
     * @clientRole todo
     * @supplierRole partes
     */
    private Departamento lnkDepartamento;
}
```

```
public class Departamento {
}
```

Tal como anteriormente, a ferramenta utilizou a estratégia de guardar informação na forma de comentário. Para além deste apenas existe a indicação de que a classe “Empresa” tem uma referência para um objecto da classe “Departamento”, sendo que as cardinalidades se encontram incorrectas. Em relação ao comentário, este contém toda a informação relevante para a ferramenta poder efectuar o processo de *Reverse Engineering*, uma vez que inclui o tipo de associação, as cardinalidades da mesma e o nome dos papéis de cada um dos participantes na composição.

IV.7.2 – Reverse Engineering

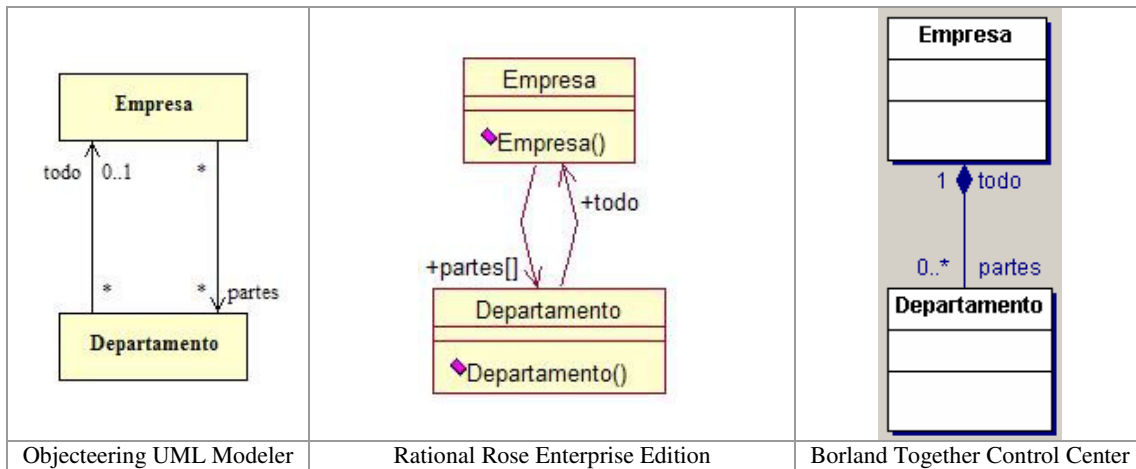


Figura 24 – Modelos resultantes para composições

Objectteering UML Modeler

Verifica-se que o modelo resultante do processo de *Reverse Engineering* apresenta duas associações simples, em vez de uma composição como no modelo inicial. Este facto não surpreende, uma vez que não existia qualquer indicação no código fonte gerado pela ferramenta acerca do tipo da associação. Como se observa os nomes dos papéis dos participantes na relação foram mantidos. Em relação às cardinalidades a análise terá que ser feita de acordo com o sentido de navegabilidade de cada associação. Assim, para a associação que tem o sentido da classe “Empresa” para a classe “Departamento” verifica-se que a cardinalidade é “várias” (“*”), tal como especificado no modelo inicial. No outro sentido a cardinalidade não está correcta uma vez que perdeu a informação de que um objecto da classe “Departamento” terá sempre que ter uma referência para um objecto da classe “Empresa”. Este facto não se deve apenas à cardinalidade definida no modelo original, mas sim à definição de composição (as partes não podem existir sem o todo). Não se pode considerar como uma falha, uma vez que esta informação teria que ser gerida por um construtor.

Rational Rose Enterprise Edition

A ferramenta criou um modelo com duas associações simples em vez de uma composição, tal como se previa uma vez que o código fonte não tinha indicação do tipo de associação. Em termos de cardinalidades verifica-se que a ferramenta

perdeu a informação existente no código fonte, uma vez que estas não se encontram definidas no modelo resultante. Os nomes dos papéis dos participantes na relação foram mantidos, assim como o construtor definido por omissão pela ferramenta.

Borland Together Control Center

Como se verifica o modelo resultante após o processo de *Reverse Engineering* é igual ao modelo inicial. Estes resultados não podem, porém, ser comparados com os obtidos pelas restantes ferramentas uma vez que no nível de abstracção de código fonte a informação relativa ao modelo é guardada na forma de um comentário Java com formato específico da ferramenta, em vez de código fonte “verdadeiro”.

IV.7.3 – Resumo dos resultados e conclusões

		Objecteering UML Modeler	Rational Rose Enterprise Edition	Borland Together Control Center
Forward Engineering	Geração das cardinalidades			
	Manutenção dos nomes dos papéis dos participantes na relação			
	Geração de selectores para navegação nas associações			
	Geração de selectores para a cardinalidade das associações			
Reverse Engineering	Geração de uma composição (Não desdobramento da relação)			
	Indicação das cardinalidades			
	Manutenção dos nomes dos papéis dos participantes na relação			
TOTAL		5/7	3/7	?/7

Tabela 7 – Tabela comparativa para composições

Na fase de *Forward Engineering* todas as ferramentas se comportaram como previsto, à excepção da *Borland Together Control Center* que utilizou comentários como forma de armazenar informação, pelo que não é passível se ser comparada com as restantes ferramentas nesta fase. Apenas a ferramenta *Objecteering UML Modeler* gerou selectores de navegação e de cardinalidade. Em relação à fase de *Reverse Engineering*, apenas uma - *Borland Together Control Center* – foi capaz de criar uma composição no modelo final. Em relação às cardinalidades, esta foi a única ferramenta que foi capaz de guardar essa informação no modelo. Todas as ferramentas geraram os nomes dos papéis dos participantes na relação tal como no modelo original.

IV.8 – Classes Associativas

As relações estudadas até esta secção do relatório fazem uma ligação entre duas classes. Essa relação pode ela própria ter atributos e operações, caso em que se denomina classe associativa. A informação acerca da relação é modelizada como uma classe.

Em termos gráficos, uma classe associativa é representada como uma simples associação (com as devidas cardinalidades) com uma classe ligada por um tracejado. O modelo de teste utilizado nesta fase da análise foi o representado na *Figura 25*.

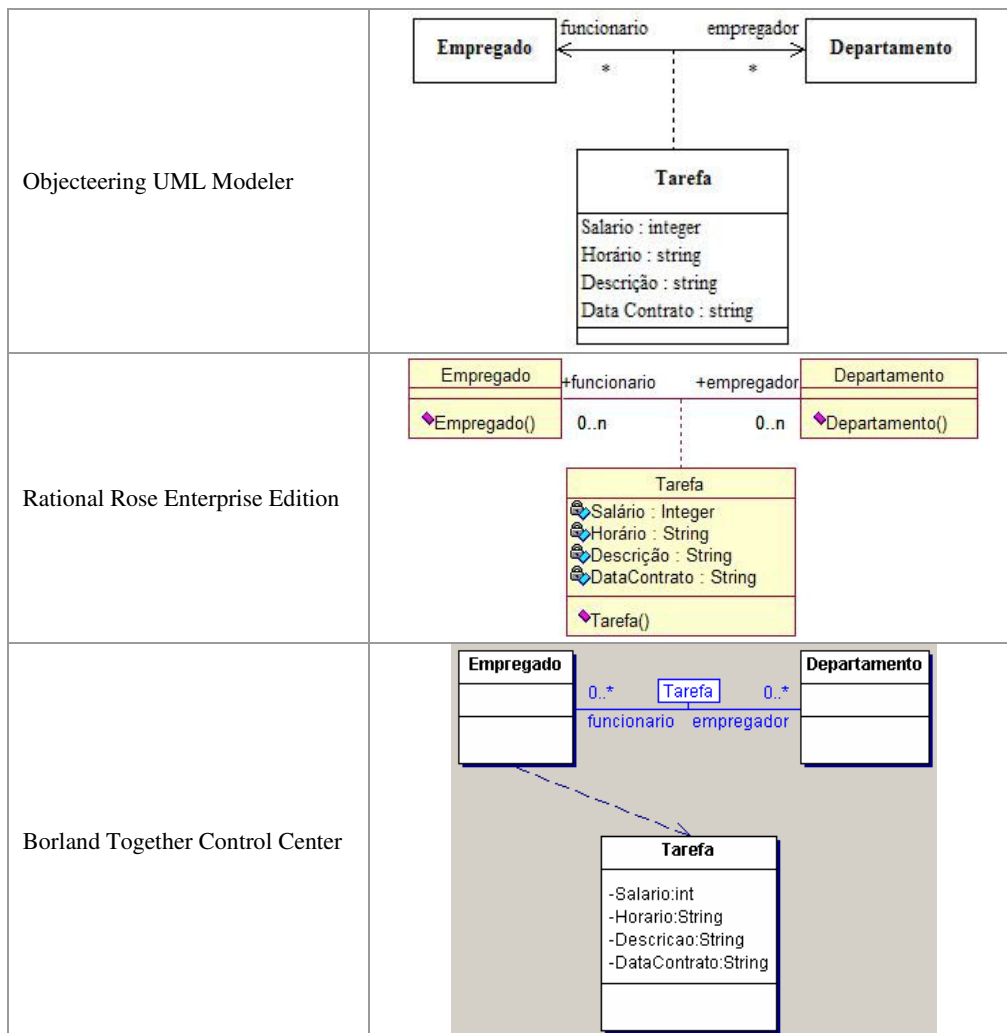


Figura 25 – Modelo de teste para classes associativas

No modelo acima apresentado são visíveis duas classes – “Empregado” e “Departamento” – e uma classe associativa – “Tarefa”. A associação entre as

classes tem as cardinalidades definidas como é visível no modelo. A classe associativa guarda informação acerca da relação, neste caso dados acerca da tarefa exercida por um empregado que trabalha num departamento.

A ferramenta *Rational Rose Enterprise Edition* gerou construtores por omissão, tal como fez nos modelos anteriormente apresentados neste relatório. Também como nos casos anteriores apresentados para a ferramenta *Objectteering UML Modeler* verifica-se que a associação se encontra definida como sendo em ambos os sentidos da navegação (apesar de o representar de forma diferente das outras ferramentas em análise).

No caso da ferramenta *Borland Together Control Center* a análise foi mais complicada devido ao facto de esta não suportar classes associativas na forma gráfica prevista no documento de normalização do UML. Após alguns contactos com a empresa produtora foi dada uma lista de instruções de como conseguir utilizar esse recurso na ferramenta em questão. Embora em termos gráficos não pareça uma classe associativa, foi esta a solução apresentada.

IV.8.1 – Forward Engineering

Objecteering UML Modeler

```
import java.util.*;

public class Empregado
{
    protected Vector empregador = new Vector();
    public Departamento getEmpregador (int i) {
        return (Departamento)this.empregador.elementAt(i);
    }
    public int cardEmpregador () {
        return this.empregador.size();
    }
}
```

```
import java.util.*;

public class Departamento
{
    protected Vector funcionario = new Vector();
    public Empregado getFuncionario (int i) {
        return (Empregado)this.funcionario.elementAt(i);
    }
    public int cardFuncionario () {
        return this.funcionario.size();
    }
}
```

```
public class Tarefa
{
    protected int Salario;
    public int getSalario () {
        return this.Salario;
    }

    protected String Horário;
    public String getHorário () {
        return this.Horário;
    }

    protected String Descrição;
    public String getDescrição () {
        return this.Descrição;
    }

    protected String Data_Contrato;
    public String getData_Contrato () {
        return this.Data_Contrato;
    }
}
```

Em relação à classe “Empregado” verifica-se que esta foi definida como tendo um vector (objecto do tipo “Vector”) de referências para objectos do tipo “Departamento”. Foram ainda definidas funções auxiliares para utilização do vector – uma para obter uma determinada referência a partir da sua posição no vector e uma outra para obter

o tamanho do vector. A classe “Departamento” é idêntica à classe “Empregado” descrita anteriormente. A classe “Tarefa” foi definida com todos os atributos incluídos no modelo, tendo os respectivos tipos sido bem definidos. Foram ainda definidas operações de retorno dos respectivos valores. Não existe, porém, nenhum tipo de ligação desta classe com as outras. A classe associativa deveria ter referência para objectos correspondentes às classes que participam na relação.

Rational Rose Enterprise Edition

```
public class Empregado
{
    public Departamento empregador[];

    /** @roseuid 3ED4C8560184 */
    public Empregado()
    {
    }
}
```

```
public class Departamento
{
    public Empregado funcionario[];

    /** @roseuid 3ED4C8560224 */
    public Departamento()
    {
    }
}
```

```
public class Tarefa
{
    private Integer Salário;
    private String Horário;
    private String Descrição;
    private String DataContrato;

    /** @roseuid 3ED4C856010C */
    public Tarefa()
    {
    }
}
```

Nas classes “Empregado” e “Departamento” verifica-se a existência de um vector de referências, para além do construtor gerado pela ferramenta. A classe “Tarefa” foi definida com todos os atributos existentes no modelo, assim como os respectivos tipos. À semelhança do verificado na análise da ferramenta anterior, verifica-se que não existe nenhuma ligação entre a classe “Tarefa” – classe associativa – e qualquer uma das outras classes.

Borland Together Control Center

```
public class Empregado {
    /**
     * @clientCardinality 0..*
     * @supplierCardinality 0..*
     * @associationAsClass Tarefa
     * @clientRole funcionario
     * @supplierRole empregador
     */
    private Departamento lnkDepartamento;
}
```

```
public class Departamento {
}
```

```
public class Tarefa {
    private int Salario;
    private String Horario;
    private String Descricao;
    private String DataContrato;
}
```

Como se observa toda a informação está incluída na classe “Empregado” que guarda as cardinalidades da relação que tem com a classe “Departamento” e a informação que a classe “Tarefa” é uma classe associativa (na forma de um comentário da linguagem Java específico da ferramenta). Apresenta ainda uma referência para um objecto da classe “Departamento”. A classe “Tarefa” não apresenta qualquer ligação com as restantes classes. A solução apresentada em termos de código fonte não é igual (nem semelhante) às apresentadas nas ferramentas anteriormente estudadas, pelo facto de não ter sido gerado código fonte “verdadeiro” uma vez que a ferramenta utilizou a técnica já vista anteriormente de guardar a informação acerca das associações na forma de comentário.

IV.8.2 – Reverse Engineering

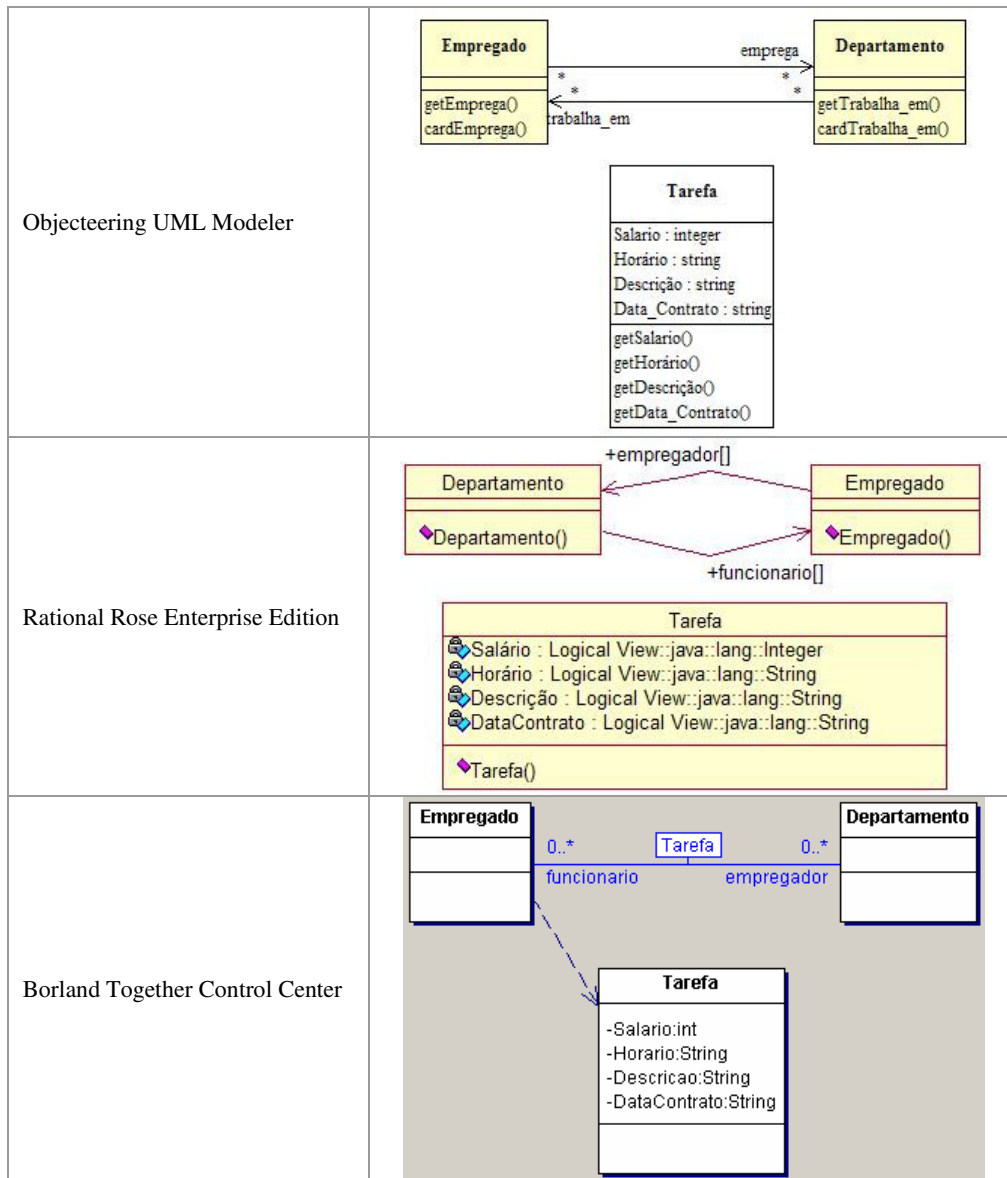


Figura 26 – Modelos resultantes para classes associativas

Objectteering UML Modeler

Como era de esperar as classes foram criadas correctamente, verificando-se que a classe associativa “Tarefa” inclui todos os atributos definidos no código fonte, assim como os respectivos tipos. Verifica-se que se encontram presentes as operações definidas no modelo original, assim como as que a ferramenta gerou automaticamente (selectores). É importante referir que os selectores apenas surgem no modelo resultante porque foi anteriormente definida essa opção, sendo que por omissão a ferramenta os filtra. Se se tiver em conta os sentidos de navegabilidade,

verifica-se que as cardinalidades estão definidas como no modelo original. Como também era de esperar não foi criada correctamente a classe associativa, uma vez que não existe nenhuma indicação no código fonte que a ligue a qualquer uma das classes.

Rational Rose Enterprise Edition

Como era de esperar as classes foram criadas correctamente, verificando-se que a classe associativa “Tarefa” inclui todos os atributos definidos no código fonte, assim como os respectivos tipos. As cardinalidades da relação não se encontram definidas no modelo. Não foi gerada correctamente a classe associativa no modelo resultante, uma vez que não existe nenhuma indicação no código fonte que a ligue às classes participantes na relação de associação presente no modelo.

Borland Together Control Center

Verifica-se que o modelo é igual ao utilizado originalmente. Todas as classes estão definidas como no modelo original, assim como a associação e respectivas cardinalidades. Tal como no modelo original existe uma ligação entre a classe associativa e a classe “Empregado”. O modelo não é passível de comparação directa com os obtidos com as restantes ferramentas uma vez que utiliza uma estratégia de armazenar a informação relevante ao processo de *Reverse Engineering* na forma de um comentário, em vez de gerar código fonte como seria esperado.

IV.8.3 – Resumo dos resultados e conclusões

























		Objecteering UML Modeler	Rational Rose Enterprise Edition	Borland Together Control Center
Forward Engineering	Geração das cardinalidades			
	Manutenção dos nomes dos papéis dos participantes na relação			
	Geração da classe associativa			
	Geração de selectores para navegação nas associações			
	Geração de selectores para a cardinalidade das associações			
Reverse Engineering	Indicação das cardinalidades			
	Manutenção dos nomes dos papéis dos participantes na relação			
	Geração da classe associativa			
	Não desdobramento da relação			
TOTAL		5/8	3/8	?/8

Tabela 8 – Tabela comparativa para classes associativas

Em relação ao processo de *Forward Engineering*, nenhuma das ferramentas foi capaz de gerar a classe associativa como esperado. A ferramenta *Borland Together Control Center* usou comentários como forma de armazenar informação. As restantes mantiveram a informação correcta acerca dos nomes dos papéis dos participantes na relação. Apenas a ferramenta *Objecteering UML Modeler* gerou selectores de navegação e de cardinalidade. Em termos de *Reverse Engineering*, apenas a ferramenta *Borland Together Control Center* guardou a informação correcta acerca das cardinalidades da relação no modelo resultante. A única ferramenta que gerou o modelo resultante com uma classe associativa foi a *Borland Together Control Center*, tendo também sido a única que não desdobrou a relação. Todas as ferramentas mantiveram os nomes dos papéis dos participantes na relação.

IV.9 – Interfaces

Uma interface é um elemento que providencia um meio de separar a vista interna e externa de um dado elemento. Permite assim dar a conhecer a(s) maneira(s) de usar um dado elemento sem no entanto mostrar os detalhes de implementação. Uma interface é implementada por classes que se comprometem a implementar os métodos que esta define. Permite ainda o uso de um objecto com diferentes tipos, conforme o objectivo da sua utilização. Em Java não existe o conceito de herança múltipla, mas como uma classe pode implementar várias interfaces, um objecto pode deste modo providenciar vários tipos.

Em termos gráficos, uma interface tem o mesmo aspecto de uma classe normal. O que a diferencia é o uso de um estereótipo “interface”, como se exemplifica na *Figura 27*.

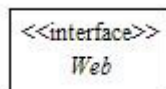


Figura 27 – Representação gráfica normal de uma interface

Em alternativa pode ser usada a representação compacta que consiste num círculo, como se exemplifica na *Figura 28*.



Figura 28 - Representação gráfica compacta de uma interface

Uma interface pode participar nas mesmas relações que as classes normais. Adicionalmente pode participar ainda numa relação do tipo realização, que se estabelece entre uma interface e uma classe. Em termos gráficos esta relação é diferente conforme se use a forma normal ou compacta de representar a interface (*Figuras 29 e 30*, respectivamente). Na forma normal utiliza-se um triângulo do lado da interface, ligado à classe de implementação por uma linha a tracejado. Na forma compacta usa-se uma linha a cheio.



Figura 29 – Representação gráfica normal de uma relação de realização

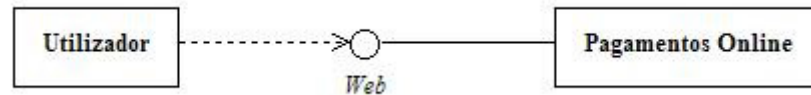


Figura 30 - Representação gráfica compacta de uma relação de realização

O modelo utilizado pelas ferramentas para testar as capacidades em termos de realizar o processo de *Round-Trip Engineering* sobre modelos com interfaces foi o representado na *Figura 31*.

Objecteering UML Modeler	
Rational Rose Enterprise Edition	
Borland Together Control Center	

Figura 31 – Modelo de teste para interfaces

O modelo encontra-se definido em dois pacotes – “PacoteUtilizador” e “PacoteBD” – de modo a verificar se as ferramentas em análise colocam a necessária indicação de importação de modo a permitir o acesso à interface “Interface_BD” por parte da classe “Utilizador”. A ferramenta *Objecteering UML Modeler* não apresenta graficamente os pacotes anteriormente referidos, apesar do modelo ser igual aos apresentados para as outras ferramentas.

O modelo correspondente à ferramenta *Borland Together Control Center* apresenta a representação gráfica normal de uma relação de realização, apesar da interface estar representada na forma compacta. Sendo apenas uma questão de grafismo, verifica-se que o modelo é igual aos utilizados pelas restantes ferramentas.

(Esta página foi intencionalmente deixada em branco)

IV.9.1 – Forward Engineering

Objectteering UML Modeler

```
package PacoteBD;  
  
public interface Interface_BD  
{  
}
```

```
package PacoteBD;  
  
public class BD  
    implements PacoteBD.Interface_BD  
{  
}
```

```
package PacoteUtilizador;  
  
import PacoteBD.Interface_BD;  
  
public class Utilizador  
{  
}
```

Como é visível no código fonte acima apresentado, as classes foram correctamente distribuídas pelos respectivos pacotes. Verifica-se a correcta geração da classe “BD” e da interface “Interface_BD”. A primeira tem a indicação de que implementa a segunda, pelo que se considera correcto e de acordo com o modelo original. Em relação à classe “Utilizador”, verifica-se que indica a importação da interface que pretende utilizar. Contudo, como não foi gerado nenhum código no corpo da classe “Utilizador”, perdeu-se a informação de que esta classe utiliza de alguma forma a interface “Interface_BD”.

Rational Rose Enterprise Edition

```
package PacoteBD;  
  
public interface Interface_BD  
{  
}
```

```
package PacoteBD;  
  
public class BD implements Interface_BD  
{  
  
    /** @roseuid 3EBEFDAF01E5 */  
    public BD()  
    {  
    }  
}
```

```
package PacoteUtilizador;

import PacoteBD.Interface_BD;

public class Utilizador
{
    /** @roseuid 3EBEFDAF03D0 */
    public Utilizador()
    {
    }
}
```

O código gerado encontra-se correcto e de acordo com a especificação, à semelhança do que aconteceu com a ferramenta analisada anteriormente. A distribuição das classes pelos pacotes encontra-se correcta, assim como a indicação de importação por parte da classe “Utilizador” da interface “Interface_BD”. Tanto a interface como a classe que a implementa (“BD”) foram correctamente geradas pela ferramenta. Contudo, tal como no caso da ferramenta anterior, perdeu-se a informação de que a classe “Utilizador” utiliza de alguma forma a interface “Interface_BD”.

Borland Together Control Center

```
package PacoteBD;

public interface Interface_BD {
}
```

```
package PacoteBD;

public class BD implements Interface_BD {
}
```

```
package PacoteUtilizador;

public class Utilizador {
    /** @link dependency */
    /**# PacoteBD.Interface_BD lnkInterface_BD; */
}
```

Verifica-se que a ferramenta gerou correctamente código fonte para a interface e para a classe que a implementa (“Interface_BD” e “BD”, respectivamente). Em relação à classe “Utilizador”, mais uma vez a ferramenta armazenou a informação relativa à relação de dependência na forma de comentário.

IV.9.2 – Reverse Engineering

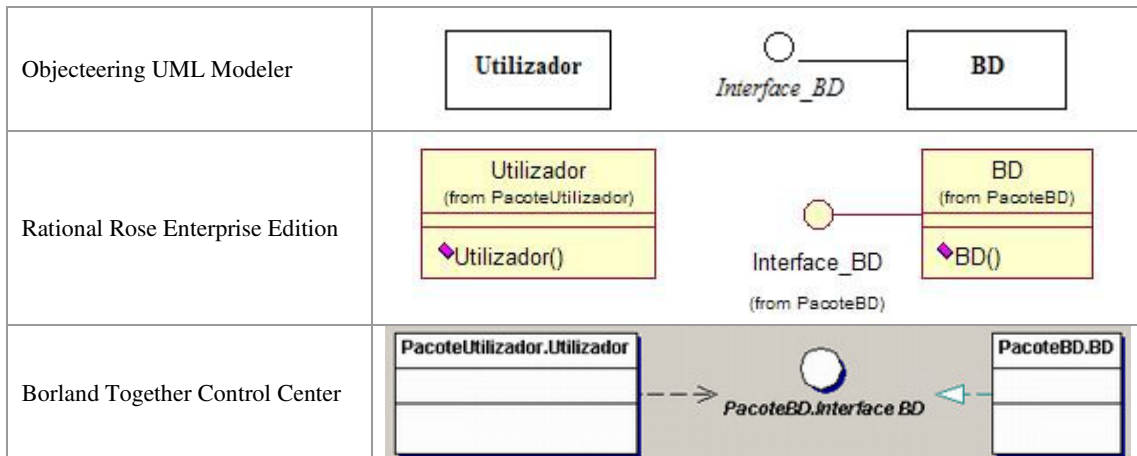


Figura 32 – Modelos resultantes para interfaces

Objectteering UML Modeler

O modelo resultante após o processo de *Reverse Engineering* inclui ambas as classes e a interface tal como no modelo original (e como definido no código fonte). Verifica-se ainda a correcta geração dos pacotes. O modelo apresenta uma relação de realização entre a classe “BD” e a interface “Interface_BD”, tal como no modelo original. Como era espectável, dado que a perda de informação já tinha ocorrido na fase anterior, não foi recuperada a dependência da classe “Utilizador” na interface “Interface_BD”.

Rational Rose Enterprise Edition

O modelo acima indicado apresenta as classes e a interface existentes no modelo original de forma correcta, distribuídas pelos respectivos pacotes. Verifica-se ainda a correcta inclusão da relação de realização. Tal como no caso da ferramenta anterior, a dependência entre a classe “Utilizador” e a interface “Interface_BD” não foi recuperada, uma vez que essa informação foi perdida na fase anterior.

Borland Together Control Center

O modelo resultante é igual ao modelo original. Todas as classes e a interface estão bem definidas, assim como a relação de realização entre a interface e a classe que a implementa (“BD”). Encontra-se ainda definida uma dependência entre a classe “Utilizador” e a interface, tal como definido na fase de desenho. A informação acerca desta dependência, no nível de abstracção de código fonte, foi armazenada na forma de comentário em vez de código fonte, pelo que não se pode comparar esta característica com as restantes ferramentas.

IV.9.3 – Resumo dos resultados e conclusões


		Objecteering UML Modeler	Rational Rose Enterprise Edition	Borland Together Control Center
Forward Engineering	Geração da interface			
	Indicação da importação pretendida entre as classes			
	Indicação da implementação pretendida entre as classes			
	Geração da dependência da interface			
Reverse Engineering	Geração da interface			
	Geração da relação de dependência			
	Geração da relação de realização			
TOTAL		5/7	5/7	?/7

Tabela 9 – Tabela comparativa para interfaces

Na fase de *Forward Engineering* todas as ferramentas se apresentam com resultados semelhantes, à excepção da *Borland Together Control Center* que não foi capaz de gerar a indicação de importação no código fonte (gerou um comentário com essa informação). Todas as ferramentas geraram a implementação pretendida entre as classes, embora nenhuma tenha gerado correctamente a dependência da interface. Durante a fase de *Reverse Engineering* também os resultados são semelhantes para as ferramentas em análise. Em relação à relação de dependência, verifica-se que apenas a ferramenta *Borland Together Control Center* a gerou no modelo resultante.

V - Análise Quantitativa das ferramentas de RTE

V.1 – Modelo de teste utilizado

Common Warehouse Metamodel (CWM)

O objectivo da iniciativa *Common Warehouse Metadata* da OMG é disponibilizar a fácil troca de metadados entre ferramentas de *datawarehouse* e repositórios de metadados em sistemas heterogéneos distribuídos.

O *Common Warehouse Metamodel* é um metamodelo que reúne metamodelos de diversas tecnologias como bases de dados relacionais, XML, OLAP (*Online Analytical Processing*), *data mining* e outras.

É interessante analisar a relação existente entre o CWM e o UML. O metamodelo do CWM define um pacote “*ObjectModel*” que é baseado na especificação do UML exceptuando todos os aspectos irrelevantes para um cenário de *data warehouse*. Qualquer metaclasses do CWM herda directa ou indirectamente de alguma metaclasses do *ObjectModel*. Considerando como exemplo o pacote relacional do CWM, verifica-se que este define uma metaclasses “*Table*” (tabela) que deriva da metaclasses “*Class*” (classe) do *ObjectModel*. De modo semelhante, a metaclasses relacional do CWM “*column*” (coluna) deriva da metaclasses “*attribute*” (atributo) do *ObjectModel*. A notação do UML é utilizada na representação de diagramas do metamodelo do CWM. As restrições adicionais do CWM são representadas em OCL (*Object Constraint Language*), tal como definido na especificação do UML.

V.2 – Resultados Obtidos

V.2.1 – Apresentação dos resultados

Apresenta-se de seguida uma tabela com os resultados obtidos após o processo descrito na metodologia deste relatório. São representadas as métricas para o modelo original e para cada um dos modelos gerados pelas ferramentas em relação a cada um das metaclasses. É apresentada ainda uma coluna para cada um dos modelos gerados pelas ferramentas com a indicação do erro relativo. Este foi

calculado tendo em conta os valores do modelo original. Apresenta-se ainda a média simples e ponderada (de acordo com o número de metaobjectos do modelo original) para cada uma das ferramentas, de modo a permitir uma melhor comparação.

	Modelo Original	Objecteering UML Modeler	ER	Rational Rose Enterprise Edition	ER	Borland Together Control Centre	ER
Association	214	101	-53%	450	110%	47	-78%
AssociationEnd	413	208	-50%	872	111%	93	-77%
Attribute	438	0	-100%	268	-39%	503	15%
Generalization	333	323	-3%	288	-14%	102	-69%
MMClass	382	383	0%	363	-5%	325	-15%
Operation + Method	0 + 382	381 + 0	0%	351 + 0	-8%	0 + 0	-100%
Package	35	37	6%	35	0%	35	0%
Médias simples			-33%		27%		-37%
Médias ponderadas			-35%		21%		-32%

Tabela 10 – Tabela de métricas da análise quantitativa

Se o cenário fosse aquele em que o processo de Round-Trip Engineering se efectua sem qualquer perda de informação, então a tabela acima apresentada deveria apresentar valores nulos para o erro relativo. Uma vez que tal não é o caso, torna-se necessário discutir os resultados obtidos, o que é efectuado na secção seguinte.

V.2.2 – Discussão dos resultados

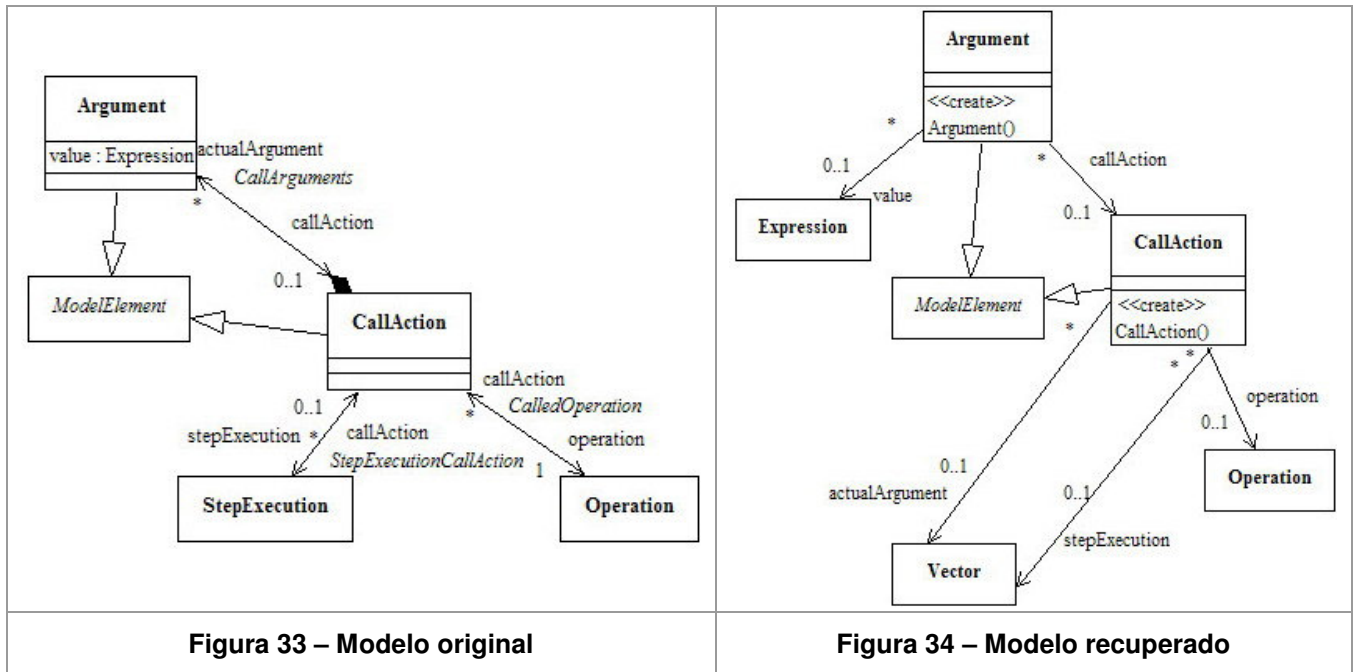
Atributos

A ferramenta *Objecteering UML Modeler* perdeu a informação acerca de todos os atributos durante o processo. Verifica-se que tal é devido ao facto de a ferramenta durante o processo de *Reverse Engineering* gerar uma associação entre a classe que declara o atributo e a classe desse mesmo atributo (*Figuras 33 e 34*).

Associações

Ainda em relação à ferramenta *Objecteering UML Modeler* e tendo em conta o que foi apresentado anteriormente em relação aos atributos, seria de esperar que o número de associações no modelo aumentasse consideravelmente, o que não se verifica pelos valores da tabela em que o número de associações diminuiu em 53%. Verifica-se que o modelo após *Reverse Engineering* efectuado por esta ferramenta tem a indicação de todas as associações, embora o mesmo modelo após

exportação e importação por parte de outra ferramenta de modo a ser possível a extracção de métricas não tenha indicação de algumas associações. Apenas manteve a informação relativa a associações entre classes do mesmo pacote, tendo as restantes sido perdidas (Figura 35).



Ainda em relação às associações verifica-se que a ferramenta *Rational Rose Enterprise Edition* as gerou em maior número (110%) do que definido no modelo original. Este facto não surpreende, uma vez que como se verificou na análise qualitativa, a ferramenta desdobra as associações em duas.

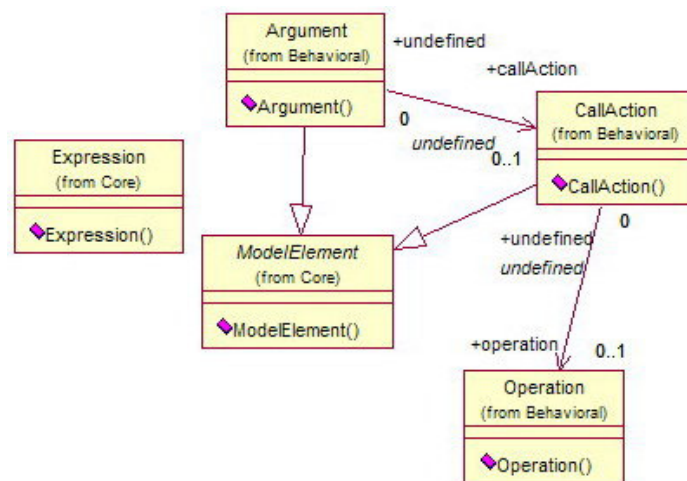


Figura 35 – Modelo recuperado importado para efeitos de extracção de métricas

Verifica-se que a ferramenta *Borland Together Control Centre* perdeu cerca de 15% das classes existentes no modelo original, pelo que poderá ser uma possível explicação para os maus resultados em termos de número de associações e de generalizações.

Em todas as ferramentas se verifica que o número de *AssociationEnd's* é aproximadamente o dobro do valor de associações, tal como se esperava.

Operações e métodos

O modelo original tem definidos construtores nas classes como sendo métodos, e não operações. Nenhuma das ferramentas teve estes construtores em consideração, tendo duas delas gerado construtores durante o processo: *Objectteering UML Modeler* e *Rational Rose Enterprise Edition*.

V.3 - Conclusões

Verificou-se pela análise quantitativa que se perdeu uma grande quantidade de informação por parte das ferramentas em estudo. Alguma dessa informação perdida é facilmente explicada simplesmente cruzando os resultados com a informação obtida na fase da análise qualitativa, como referido na secção anterior. Outros resultados, porém, indicam que alguma informação que se perdeu não é justificada pela análise qualitativa, pelo que foram apresentadas algumas justificações possíveis para a perda de informação.

Se, por um lado, alguma informação que se perdeu não é significativa em termos quantitativos (pequena percentagem), é importante ter em linha de conta que num sistema de dimensão média/grande a perda é considerável e deverá ser considerado o esforço dispensado de modo a corrigir algumas das perdas.

Considerando o tipo de informação que se perde em vez de simplesmente as quantidades em que se perde, verifica-se que algumas perdas apesar de existirem em número reduzido, são graves. É o caso da informação relativa às classes existentes e dos respectivos atributos.

Em termos de comparação directa entre as ferramentas em estudo, e se se tiver em conta as médias simples em valor absoluto, verifica-se que a ordenação decrescente

de perda de informação é a seguinte: *Rational Rose Enterprise Edition* (27%); *Objectteering UML Modeler* (33%) e *Borland Together Control Centre* (37%). Porém, se se tiver em conta as médias ponderadas a ordenação altera-se, mantendo a mesma ferramenta em primeiro lugar (21%) mas invertendo as posições das outras (32% e 35%). Apesar de uma das ferramentas se ter destacado um pouco das restantes, considera-se que a variação dos resultados não é muito significativa.

VI - Case-Study Cinema

VI.1 - Apresentação do Case-Study

Apresenta-se nesta secção um *case-study* relativo a um sistema de venda de bilhetes para um cinema. Foi utilizado um modelo relativamente simples de modo a poder ser facilmente observável a perda ou modificação de informação ao longo do processo de *Round-Trip Engineering* por parte das ferramentas em estudo. A metodologia aplicada foi a mesma que no capítulo anterior deste relatório, com a única diferença de não existir o passo de geração de métricas. Pretende-se comparar os modelos resultantes por observação directa. O modelo utilizado foi o representado na *Figura 36*.

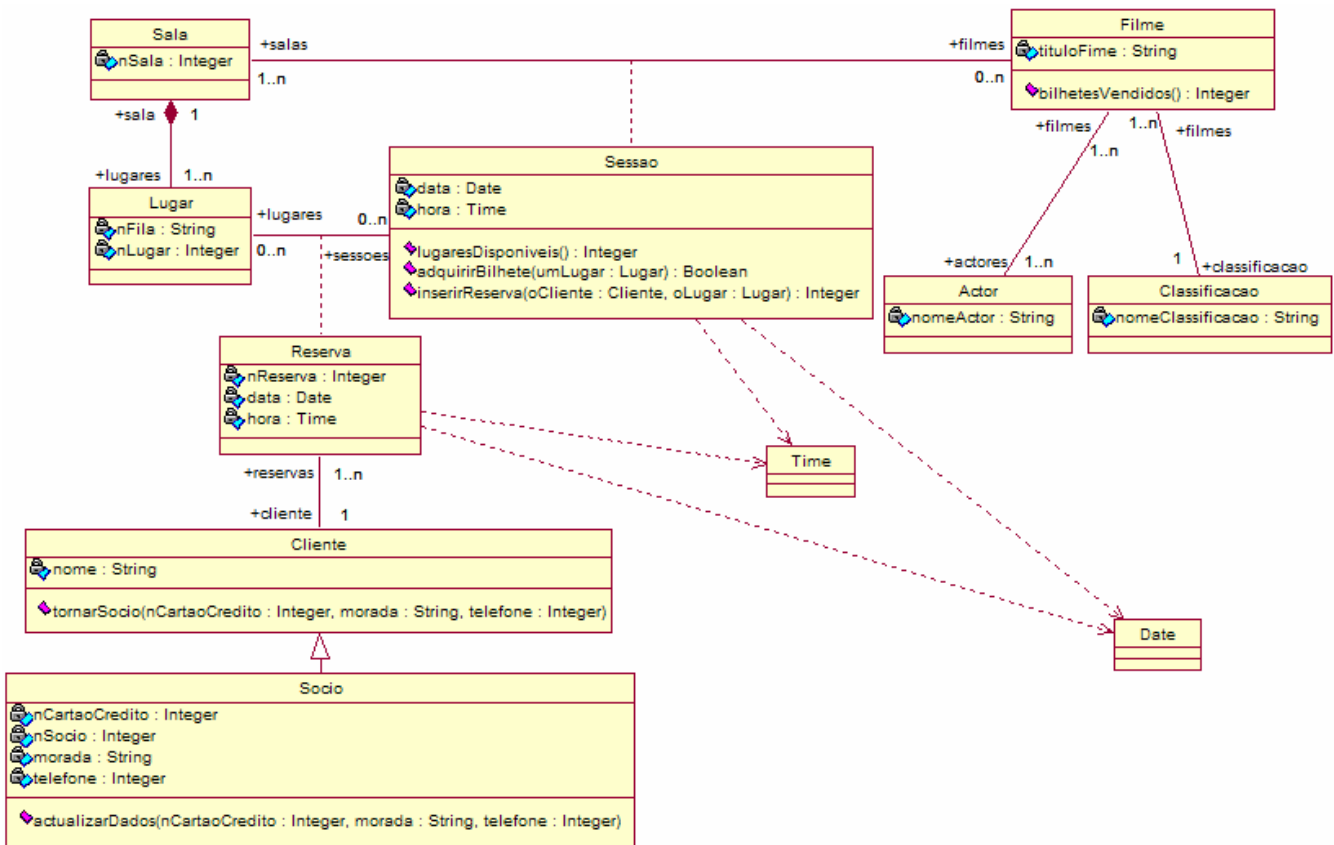


Figura 36 – Modelo de teste utilizado no case-study

Segundo a metodologia anteriormente referida, o modelo acima apresentado foi importado, no formato XML, pelas ferramentas em teste e foi realizado o processo de

Round-Trip Engineering em cada uma delas. Os modelos resultantes deste processo são expostos na secção seguinte.

VI.2 - Modelos resultantes e observações

Seguem-se os modelos resultantes para cada uma das ferramentas em estudo e observações retiradas dos mesmos. Sempre que possível é efectuada uma referência para partes anteriores deste relatório, de modo a demonstrar a consistência dos resultados.

Objecteering UML Modeler

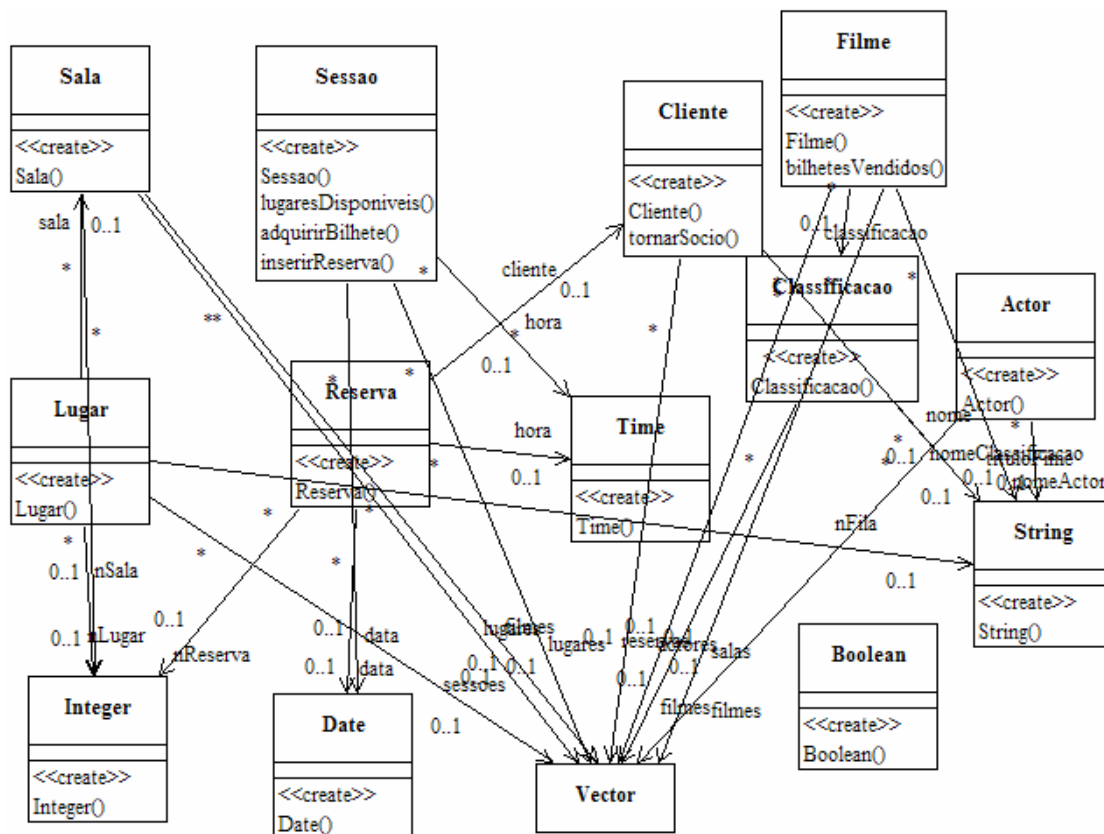


Figura 37 – Modelo resultante para a ferramenta Objecteering UML Modeler

O modelo resultante apresentado na Figura 37 não é facilmente legível pelo facto de ter sido gerada a classe Vector por parte da ferramenta e todas as associações com cardinalidade “*” terem sido transformadas numa associação para esta classe. Foram ainda criadas as classes Boolean, String e Integer, que apresentam

associações sempre que algum atributo de uma das classes do modelo tem um destes tipos. Note-se que os tipos de retorno das operações não aparecem. Numa tentativa de tornar o modelo mais legível e mais facilmente comparável com o modelo original, foram apagadas as classes anteriormente referidas e o modelo é novamente apresentado na *Figura 38*.

Como se verifica pelo modelo resultante apresentado na *Figura 38* a ferramenta perdeu informação acerca das classes associativas presentes no modelo inicial. Este facto não surpreende uma vez que essa perda de informação já tinha sido detectada na análise qualitativa referente a esta abstracção (*Capítulo IV.8*).

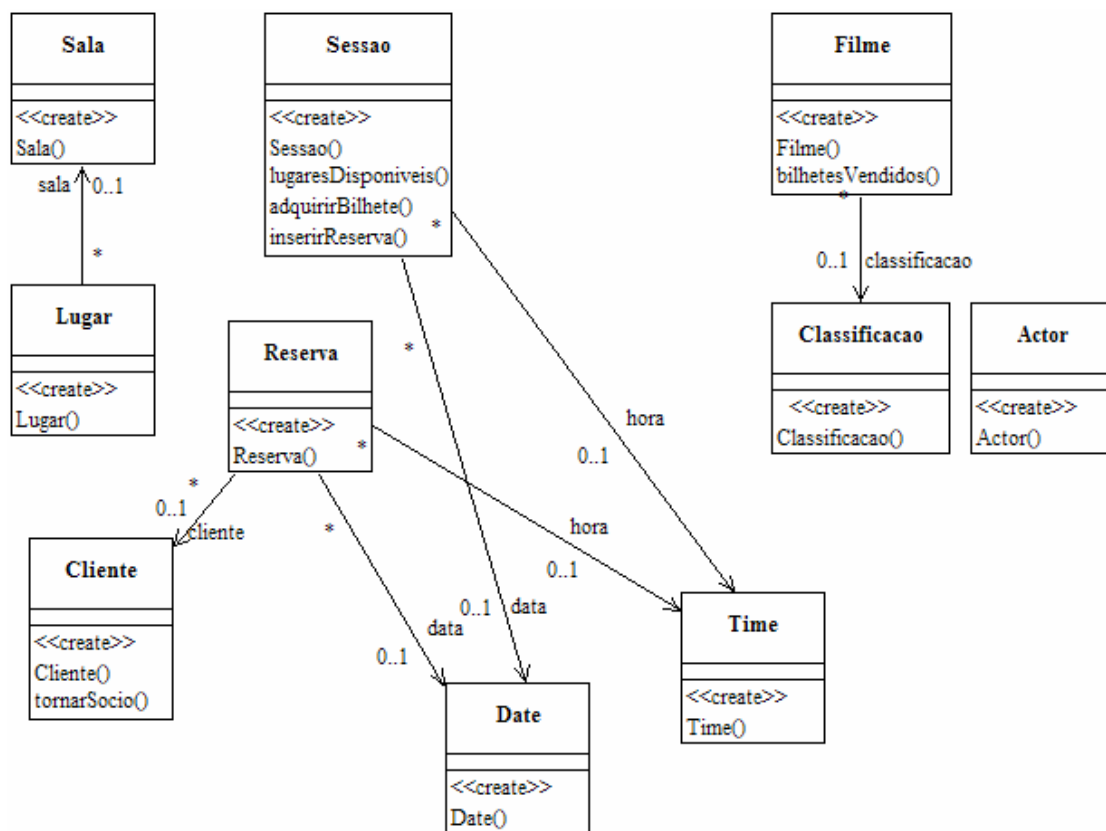


Figura 38 – Modelo resultante para a ferramenta Objecteering UML Modeler

Em termos de cardinalidade verifica-se que a ferramenta transformou a cardinalidade “1” em “0..1”, tal como se esperava pelos resultados apresentados na análise qualitativa (*Capítulo IV*). Tal como foi justificado anteriormente, as cardinalidades “*” foram transformadas em associações para a classe “Vector” (*Figura 37*).

Verifica-se a presença de um construtor em cada uma das classes, sendo que o modelo original não os tinha definidos. A ferramenta não gerou uma composição no modelo resultante, tendo sido substituída por uma associação simples. Este facto era esperado de acordo com os resultados obtidos na análise qualitativa (*Capítulo IV.7*).

As operações foram mantidas, assim como os parâmetros definidos no modelo original. Foram mantidos os tipos de retorno das operações. Esta informação relativa a parâmetros e tipo de retorno não se encontra visível nas figuras apresentadas por questões de legibilidade.

É importante referir que o modelo resultante apresentado para esta ferramenta não tem representada a classe sócio, tal como definido no modelo inicial, o que é explicado pelo facto de a ferramenta não conseguir efectuar *Reverse Engineering* sobre uma classe que herde de outra. Este facto não surpreende, uma vez que já tinha sido detectado na análise qualitativa deste relatório (*Capítulo IV.3*).

Rational Rose Enterprise Edition

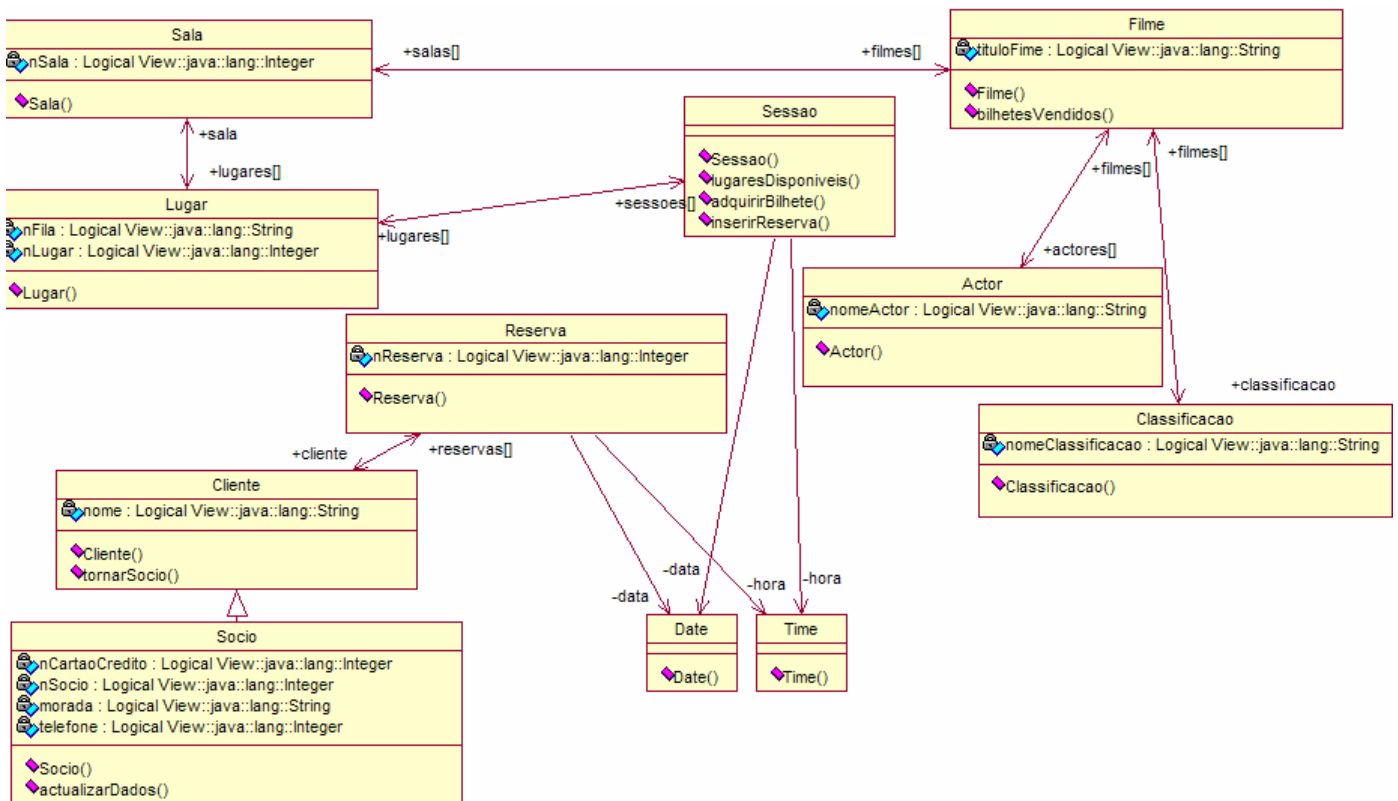


Figura 39 – Modelo resultante para a ferramenta Rational Rose Enterprise Edition

Verifica-se, por observação da *Figura 39*, que a ferramenta perdeu a informação acerca das classes associativas, tal como verificado no *Capítulo IV.8* deste relatório. Em relação às cardinalidades, verifica-se que estas foram perdidas totalmente durante o processo, confirmando o que havia sido verificado na análise qualitativa. Situação semelhante aconteceu com a composição presente no modelo inicial que, mais uma vez de acordo com o verificado no *Capítulo IV.7*, foi transformada numa associação simples. A relação de herança foi mantida, tal como era esperado de acordo com o *Capítulo IV.3*.

As classes apresentam um construtor, o que não acontecia no modelo inicial. Refira-se que, na realidade, o processo de *Forward Engineering* efectuado por esta ferramenta sobre o modelo inicial já tinha a informação dos construtores em cada uma das classes, uma vez que estes foram automaticamente adicionados antes do processo de geração de código.

Os atributos e respectivos tipos foram mantidos, excepto os que tinham o tipo referente às classes *Time* e *Date* que, por estarem definidas no modelo, foram substituídos por associações para estas classes. Em relação às operações verifica-se que estas mantêm o tipo de retorno, assim como os parâmetros e respectivos tipos. A informação relativa às operações não se encontra visível por questões de legibilidade.

Borland Together Control Center

Verifica-se pela *Figura 40* que o modelo resultante para esta ferramenta inclui a informação acerca das classes associativas, de acordo com o que havia sido verificado no *Capítulo IV.8* deste relatório. A relação de herança e a composição presentes no modelo original também se encontram presentes, de acordo com os *Capítulos IV.3* e *IV.7*, respectivamente. As cardinalidades encontram-se definidas de acordo com o modelo original.

Os atributos e respectivos tipos foram mantidos, tal como esperado pela análise qualitativa deste relatório. O mesmo se passou com as operações, parâmetros e respectivos tipos e tipos de retorno. Em relação aos atributos do tipo *Time* e *Date*, verifica-se que, apesar de terem sido mantidos no modelo resultante, foram

adicionadas associações para as respectivas classes, uma vez que estas se encontram presentes no modelo. Existe assim uma redundância de informação.

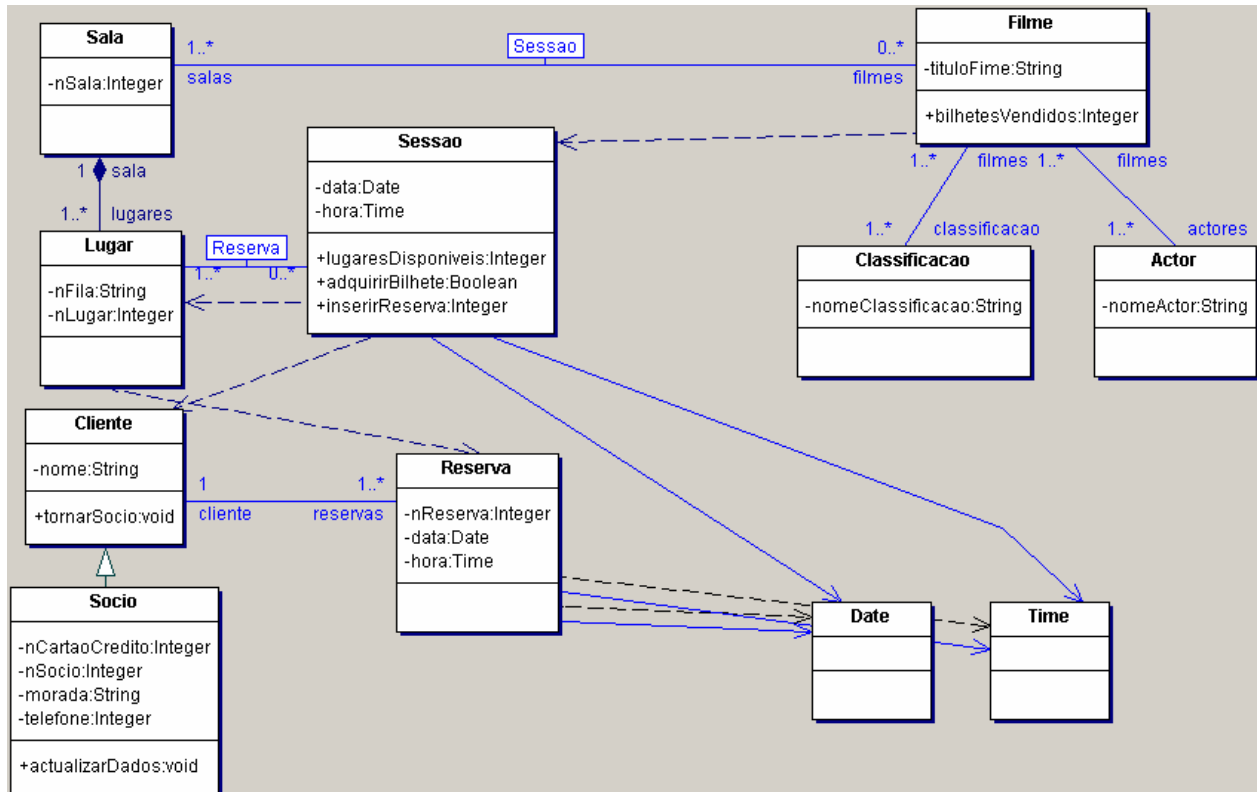


Figura 40 – Modelo resultante para a ferramenta Borland Together Control Center

VII – *Apreciação Crítica e Conclusões*

Tendo em conta as conclusões retiradas ao longo das várias fases deste relatório podem-se agora retirar conclusões gerais acerca do estado da arte do processo de *Round-Trip Engineering*.

Actualmente a maioria das ferramentas apenas suporta o processo de *Round-Trip Engineering* sobre diagramas de classes, o que é insuficiente para conseguir manter a mesma informação nos dois níveis de abstracção – modelo e código fonte. Em relação ao diagrama de classes, que é o efectivamente abordado neste projecto final de curso, considera-se que ainda é necessário alguma evolução por parte das ferramentas. Com efeito teriam que evoluir de modo a reduzir a quantidade de informação perdida e a conseguir realizar todo o processo mantendo o modelo original e o modelo recuperado idênticos.

Enquanto o processo de *Round-Trip Engineering* efectuado pelas ferramentas não atingir esse estado de maturidade, torna-se difícil a sua utilização como método de desenvolvimento de software. Ambos os processos que compõem o *Round-Trip Engineering* têm as suas vantagens, tal como indicado no capítulo dedicado aos conceitos teóricos (capítulo II). O que não se pode esperar para já é que as ferramentas consigam manter o modelo actualizado de acordo com o código em desenvolvimento, chegando-se a uma altura em que o modelo pouco ou nada tem a ver com o que de facto está programado.

Foram levantadas algumas questões neste relatório, pelo que no capítulo seguinte são referidas algumas linhas de orientação para possíveis trabalhos futuros nesta área.

VIII – Evolução Futura

Seguindo a metodologia aplicada neste trabalho, possíveis trabalhos futuros poderão tentar corrigir o código fonte gerado pelas ferramentas de modo a testar o processo de *Reverse Engineering* sobre código que se sabe estar correcto à partida, em vez de usar o código gerado automaticamente. Uma outra hipótese será alterar a metodologia da análise de modo a realizar o processo da seguinte forma: a partir de código fonte gerar o modelo correspondente (*Reverse Engineering*) e, de seguida, gerar código fonte (*Forward Engineering*). O método de análise teria assim que ser baseado em comparação de código fonte, em vez de comparação de modelos.

Obviamente que trabalhos futuros se poderão basear em novas ferramentas que possam surgir e mesmo novas versões das ferramentas e *plugins* usados neste trabalho. Esta é uma área em desenvolvimento, pelo que se espera que sejam introduzidas melhorias nas ferramentas a curto/médio prazo.

Outras possíveis análises poderão ser baseadas em diagramas que não o diagrama de classes utilizado neste trabalho. Seria interessante utilizar o metamodelo do UML 2.0 de modo a realizar a análise com base em diagramas de sequência.

Bibliografia

Silva, Alberto e Carlos Videira. UML Metodologias e Ferramentas CASE. 1ª Edição. Edições Centro Atlântico. Portugal, 2001.

Iyengar, Sridhar. XML Metadata Interchange (OMG XMI); Distributed Metadata Interchange for the Web Generation. Meta-Data Conference. Atlantic City, 2001.

Supporting Quotes for XMI Specifications. XMI Specification Press Kit, 1999.

<http://www.omg.org/news/pr99/supporting_quotes.html>

Rhodes, Chris Vander. OMG Members Unite in Support of XMI Technology. OMG's. Press Releases, 1999.

<http://www.omg.org/news/pr99/2_6.html>

Breuer, P. T. e K. Lano. Creating Specifications from Code: Reverse-Engineering Techniques. Programing Research Group. Oxford University Computer Laboratory. Oxford, 1995.

Brodsky, Stephen A. XMI: The Value of Interchange. OMG XMI Briefing. EUA,1999.

Systä, Tarja. Dynamic Modeling in Forward and Reverse Engineering of Object-Oriented Software Systems. Department of Computer Science. University of Tampere. Filândia, 1999.

Baxter, Ira D. e Michael Mehlich. Reverse Engineering is Reverse Forward Engineering. Semantic Designs, Inc. Austin, 1997.

OMG. XMI Audio Conference. XMI Specification Press Kit, 1999.

<http://www.omg.org/news/pr99/xmi_overview.html>

Keller, Shawn. Forward and Reverse Engineering. WaveBend Solutions, 2001.

<http://www.therationaledge.com/content/oct_01/t_javaEngineeringWithRose_sk.html>

Wolicki, Carol e Stephen Brodsky. Object Management Group. 1999.

OMG. Information related to OMG's XMI Technology Briefing. XML Metadata Interchange (XMI[™]). XMI[™] Specification Press Kit. February 5, 1999.

<<http://www.omg.org/mda/specks.html>>

Philispart, Laurent. Combining Modelig and Application Mining for better software. Cast, 2001.

Objects by Design: Choosing a UML Modeling Tool. Round - Trip Engineering. 2003.

<http://www.objectsbydesign.com/tools/modeling_tools.html>

Keller, Shawn. Forward and Reverse Engineering of Code With Rational Rose and Visual Age for Java. Rational Edge, 2001.

<http://www.therationaledge.com/content/oct_01/t_javaEngineeringWithRose_sk.html>

Knublauch, Holger e Thomas Rose. Round trip Engineering of Ontologies for Knowledge-Based Systems. Research Institute for Applied Knowledge Processing, Ulm, Germany, 2000.

Medvidovic, Nenad e David S. Rosenblum. Round-Trip Software Engineering Using UML: From Architecture to Design and Back. 1999.

Reverse Engineering – A Whatis definition.

<[http://www.whatistechtarget.com/definition/\(\),sid9_gci507015,00.html](http://www.whatistechtarget.com/definition/(),sid9_gci507015,00.html)>

Müller, Hausi A. et. al. Reverse Engineering: A Road Map. 2000.

Improving Developer Productivity Through The Unified Modeling Language, Embarcadero Application Development Tools, Embarcadero Technologies, Inc, 2000.

Bontemps, Yves. A Round-Trip Approach to Requirements Engineering. FUNDP Institute d'Informatique Aspirant FNRS, 2000.

Tsai, W. T., H. J. Lee e S. Bhattacharya. Software Automation by Combining Reverse Engineering with Forward Engineering Tools. 1995

Consistency between Design and Running Application as Condition for Modernization and Further Development. MicroTOOL GmbH, Berlim, 2001.

Round trip engineering, UML Bridge, Object Venture, 2002

<<http://objectventure.com/umlbridge.html>>

Holub, Alen. Round Trip Engineering is a fundamentally flawed process, When it comes to good keep it simple. 2002.

<http://www.javaworld.com/javaworld/jw-01-2002/jw-0111-ootools_p.html>

OMG. XML Metadata Interchange (XMI) Specification. Versão 1.2. 2002

Iyengar, Sridhar e Steve Brodsky. XML Metadata Interchange. Proposal to OA & DTF RFP-3. 1998.

Systä, Tarja. Dynamic reverse engineering of Java software. 1998.

Gannod, Gerald C. e Betty H. C. Cheng. A two-phase approach to reverse engineering using formal methods. Computer Science Department, Michigan State University, USA, 1993.

Layzell, P.J., M.J. Freeman e P. Benedus. Improving Reverse Engineering through the use of multiple knowledge sources. University of Manchester, Institute of Science and Technology, UK, 2000.

Mancoridis, Mitchell, Rorres et. al. Tools for reverse engineering: What we are doing while waiting for the holy grail of data exchange?. Drexel University.

Ward, M.P. Reverse Engineering through Formal Transformation. Computer Science Department, Durham, 1994.

Klint, P., M.G.J. van den Brand, C. Verhoef. Reverse Engineering and System Renovation. Programming Research Group, University of Amsterdam, 2002.

Iyengar, Sridhar e Steve Brodsky. Metadata Integration using UML, MOF and XMI. 2000.

Mitchell, Brian S., Spiros Mancoridis e Martin Traverso. Search based Reverse Engineering. Department of Mathematics & Computer Science, Drexel University, 2002.

Rugaber, Spence e Richard Clayton. The Representation Problem in Reverse Engineering. College of Computing and Software Research Center, Georgia Institute of Technology, Atlanta.

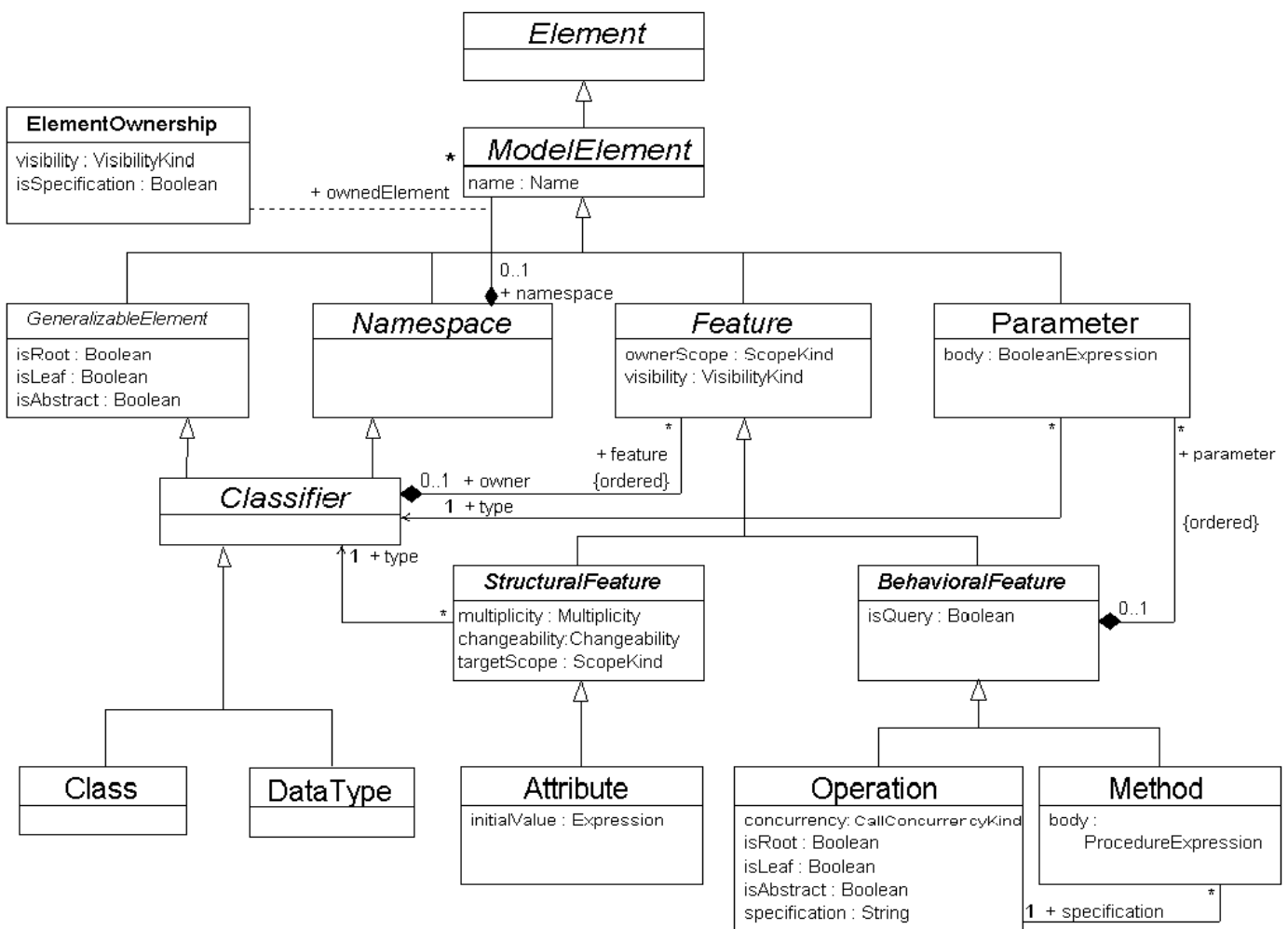
OMG. Unified Modeling Language Specification. Versão 1.4. 2001.

OMG. UML for Systems Engineering. 2002.

Anexos

Anexo A - Metamodelo UML 1.*

Na análise quantitativa anteriormente apresentada neste relatório é utilizado o metamodelo do UML, versão 1.*. Neste capítulo é apresentada uma partes desse metamodelo. Segue-se o diagrama de classes do pacote “core”, com uma breve descrição das respectivas metaclasses.



A metaclasses de topo **Element** é a parte atómica de um modelo. **ModelElement** é uma entidade do sistema a ser modelado. **GeneralizableElement** é um elemento do modelo que pode participar numa relação de generalização e que pode ser uma generalização de outros elementos do mesmo tipo. **Namespace** é um **ModelElement** que contem um conjunto de outros **ModelElements**, como **Associations** e **Classifiers**. Cada **ModelElement** terá que ter um nome único dentro de um só **Namespace**.

Outras partes do modelo como as características de um *Classifier* não são modeladas como elementos de um *Namespace*. O *Namespace* é utilizado para conteúdos não estruturados como o conteúdo de um pacote ou uma classe declarada dentro do escopo de outra classe. A metaclasses ***ElementOwnership*** permite definir a visibilidade de um *ModelElement* fora do *Namespace* onde este está incluído. Um ***Classifier*** contém um conjunto de *Features* como *Attributes*, *Methods* e *Operations* e tem um nome único dentro do *Namespace*. Um *Classifier* pode ser especializado em *Class*, *Datatype*, *Interface*, *Component* e outros definidos em outros pacotes do metamodelo. Um *Classifier* é um *GeneralizableElement* e, como tal, pode herdar *Features* e participar em associações. Como um *Classifier* é também um *Namespace*, pode declarar outros *Classifiers* dentro do escopo. Uma ***Feature*** é uma característica comportamental ou estrutural de um *Classifier*. ***StructuralFeature*** é parte estrutural de uma instância de um *Classifier*, tal como um *Attribute*. Um ***Attribute*** é uma parte nomeada de um *Classifier* que descreve uma gama de valores que uma instância de *Classifier* pode ter. Uma *BehavioralFeature* refere-se à parte dinâmica de um elemento do modelo, como uma operação (*Operation*) ou um método (*Method*). No metamodelo, uma ***BehavioralFeature*** especifica a parte comportamental de um *Classifier*. ***Parameter*** é um *ModelElement* que representa uma variável (argumento) que pode ser alterada, passada ou retornada. Um *Parameter* pode incluir um nome, um tipo e a direcção da comunicação. *Parameters* são usados na especificação de operações, mensagens e eventos, *templates*, etc. Uma ***Operation*** é uma *BehavioralFeature* que representa um serviço que pode ser requisitado por um objecto. Tem uma assinatura, um nome (herdado de *ModelElement*) e uma lista de parâmetros (de *BehavioralFeature*), incluindo possíveis valores de retorno. Um ***Method*** é a implementação de uma *Operation* e especifica o algoritmo ou procedimento que afecta os resultados de uma *Operation*. Um *Method* é a declaração de uma parte comportamental nomeada de um *Classifier* e realiza uma (directamente) ou um conjunto (indirectamente) de *Operations* do *Classifier*.