Departamento
de Engenharia
Informática

Relatório de
TRABALHO FINAL DE CURSO

do Curso de
LICENCIATURA EM ENGENHARIA
INFORMÁTICA E DE COMPUTADORES (LEIC)

Ano Lectivo 2005 / 2006

N.º da Proposta: 13

Título: ProjectIT – Produção Automática de Software

Professor Orientador:

Alberto Manuel Rodrigues da Silva _____(assinatura)_____

Co-Orientador:

Carlos Alberto Escaleira Videira _____(assinatura)_____

Aluno:

51088, Rui Miguel Tavares da Silva _____(assinatura)_____

Agradecimentos

Ao Prof. Alberto Silva.

Ao Prof. Carlos Videira.

A toda a equipa do ProjectIT-Studio, nomeadamente ao João Saraiva, David Ferreira e Carlos Martins.

Ao INESC-ID, em particular ao Grupo de Sistemas de Informação (GSI).

Aos meus pais, irmão e cunhada.

Resumo

As abordagens de desenvolvimento de *software* baseadas em modelos utilizam o nível de abstracção do modelo para a construção das aplicações, de modo a que estas funcionem em qualquer plataforma. Para tal, os modelos são transformados em artefactos da plataforma de destino através de técnicas de geração automática.

O ProjecIT é um projecto de investigação do Grupo de Sistemas de Informação do INESC-ID que tem em conta este conceito na sua abordagem de desenvolvimento de *software*. Por isso, considera um conjunto de ferramentas de *software* que ajudam os intervenientes técnicos da sua abordagem a gerir e executar as tarefas da mesma.

Neste trabalho, continuou-se a desenvolver e evoluir os mecanismos de gestão e execução do processo de geração da abordagem ProjectIT, que vêm no seguimento de três TFCs anteriores. O que incluiu alterar: (1) a linguagem de modelação, (2) a forma de definir as transformações modelo-para-código, e finalmente (3) as interfaces de utilizador para configurar e gerir os processos de geração.

Finalmente, aplicou-se a abordagem ProjectIT no desenvolvimento da aplicação *MyOrders2* como forma de a validar, o que incluiu definir: (1) o modelo da aplicação com a linguagem de modelação; e (2) as transformações modelo-para-código para as plataformas WinForms.NET e ASP.NET.

Palavras-Chave

Desenvolvimento de *software*, *Model-Driven Development*, modelação, geração, *template*.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Siglas

ASP	*Active Server Pages*
CASE	*Computer Aided Systems Engineering*
CGI	*Common Gateway Interface*
CIM	*Computation Independent Model*
DSL	*Domain Specific Languages*
GPL	*General Purpose Language*
GSI	*Grupo de Sistemas de Informação*
JSP	*Java Server Pages*
MDA	*Model-Driven Architecture*
MDD	*Model-Driven Development*
OCL	*Object Constraint Language*
OMG	*Object Management Group*
PIM	*Platform Independent Model*
PSM	*Platform Specific Model*
SI	*Sistema de Informação*
UML	*Unified Modeling Language*
XIS	*eXtreme modeling Interactive Systems*
XMI	*XML Metadata Interchange*
XML	*Extensible Markup Language*
XSLT	*XSL Transformations*

1 Introdução

Ao longo da história da Informática, têm aparecido novas abordagens de desenvolvimento de *software* para reduzir a dificuldade deste processo. Estas abordagens pretendem elevar o nível de abstracção das plataformas onde se desenvolve o *software*, e assim reduzir a dificuldade do processo. As abordagens baseadas em modelos pretendem que a abstracção seja praticada ao nível dos modelos. Os modelos permitem pensar nos sistemas de informação numa plataforma mais abstracta, o que permite a melhor compreensão dos sistemas e a comunicação mais eficiente entre os interessados. Este facto verifica-se também em muitas outras disciplinas para além da Informática [1]. No entanto, na Informática existe a possibilidade de tirar maior partido desta representação, tal acontece porque os modelos são normalmente guardados em formatos que os computadores conseguem interpretar, e que aplicando as técnicas correctas podem tirar vantagem disso e gerar automaticamente a aplicação, ou pelo menos uma parte significativa da mesma [2], para plataformas de mais baixo nível.

Assim, as abordagens de desenvolvimento de *software* baseadas em modelos pretendem separar a linguagem de modelação da plataforma de implementação, isto através da definição de transformações que recebem como entrada a definição do modelo e que fornecem como saída os artefactos da plataforma [3].

Actualmente, existem vários projectos que exploram as abordagens de desenvolvimento de *software* baseadas em modelos, nomeadamente o ProjectIT [2], o openArchitectureWare (oAW) [w1], o OptimalJ [w3], etc.

Este trabalho pretende contribuir para o ProjectIT com o estudo e desenvolvimento de técnicas relacionadas com a geração automática de aplicações, que apoie os intervenientes técnicos da abordagem ProjectIT no desenvolvimento de *software*.

1.1 ProjectIT

O ProjectIT [2] é um programa de investigação do Grupo de Sistemas de Informação (GSI) do INESC-ID [w4]. O objectivo deste programa é o estudo de mecanismos de apoio à gestão de projectos de sistemas na área das tecnologias de informação, com a finalidade de melhorar a qualidade e a produtividade dos mesmos. Os princípios orientadores são baseados nos processos de desenvolvimento de sistemas de informação, destacando-se os seguintes [2]: (1) alinhar o projecto com o negócio, (2) envolver clientes e utilizadores, (3) planear o projecto adequadamente, (4) facilitar a comunicação e partilhar a visão com base

em modelos, (5) reusar sempre que possível, (6) fazer o desenvolvimento com base em modelos, e (7) manter as coisas tão simples quanto possível.

A nível aplicacional, o ProjectIT propõe o desenvolvimento de duas ferramentas, o **ProjectIT-Enterprise** e o **ProjectIT-Studio**. O **ProjectIT-Enterprise** tem como objectivo providenciar mecanismos de colaboração entre os intervenientes no desenvolvimento do sistema, numa interface *web-cliente*. O **ProjectIT-Studio** tem como objectivo providenciar mecanismos de produtividade, numa interface *rich-client*. A gestão dos artefactos produzidos, é mantida num repositório de informação comum, o ProjectIT-CommonDB, que tem como objectivo a sincronização e a integração da informação produzida pelas diferentes ferramentas. O ProjectIT-LocalDB é o repositório de informação que o ProjectIT-Studio gere localmente, que é posteriormente sincronizada com a existente no repositório comum. A Figura 1.1 mostra a arquitectura aplicacional do ProjectIT.

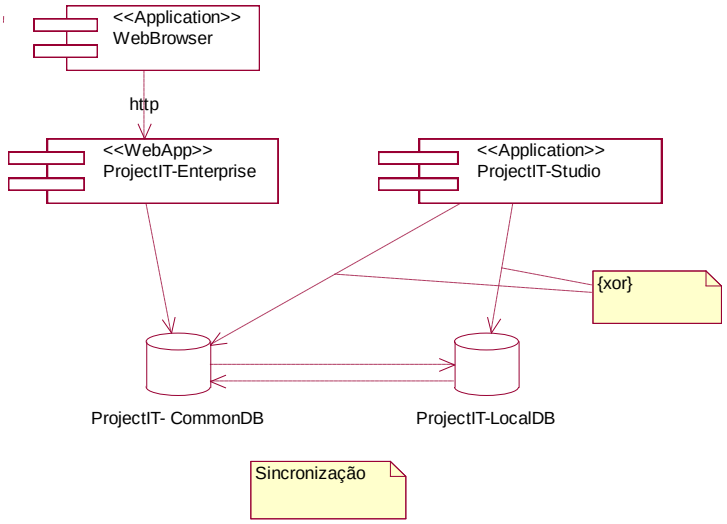


Figura 1.1 – Visão geral da arquitectura aplicacional do ProjectIT (extraído de [2]).

A nível de funcionalidade o ProjectIT propõe os seguintes módulos funcionais (ver Figura 1.2): ProjectIT-Workbench, ProjectIT-Time, ProjectIT-Requirements, ProjectIT-Tests, e ProjectIT-MDD. Destes módulos, destaca-se o módulo ProjectIT-MDD pela importância que tem para este trabalho.

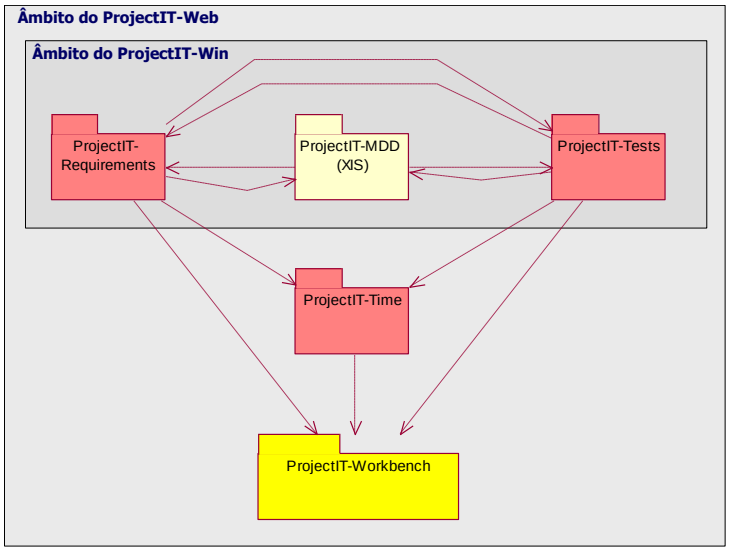


Figura 1.2 – Visão dos principais módulos funcionais do ProjectIT (extraído de [2]).

O módulo ProjectIT-MDD enquadra-se na área de modelação de sistemas de informação e na área de desenvolvimento de *software* baseado em modelos. Este módulo tem os aspectos relacionados com: (1) a linguagem de modelação e perfil UML; e (2) as transformações modelo-para-modelo e modelo-para-código. A primeira concretização deste módulo foi a ferramenta XisTool, uma aplicação desenvolvida ao longo de três TFCs anteriores [4][5][6], e que serviu de ponto de partida para a realização deste trabalho.

A finalidade destas aplicações do ProjectIT é fornecer um conjunto de ferramentas, que apoiem os intervenientes técnicos da abordagem ProjectIT na gestão e execução das tarefas de desenvolvimento.

1.1.1 Abordagem ProjectIT

A abordagem ProjectIT [7] assenta nos princípios do desenvolvimento de *software* baseado em modelos. Pretende “privilegiar as actividades de projecto (como sejam a engenharia de requisitos, a análise e o desenho) em detrimento das actividades de produção (como sejam a programação, os testes e integração de componentes e de sistema), de modo a que estas sejam realizadas tanto quanto possível de forma automática” [8].

Genericamente, a abordagem ProjectIT [7] recebe os requisitos da aplicação (e.g., requisitos funcionais, não funcionais, e de desenvolvimento) e produz os artefactos digitais (e.g., ficheiros de código fonte, *scripts* de configuração) da aplicação. A Figura 1.3 mostra as principais actividades, artefactos e intervenientes da abordagem ProjectIT.

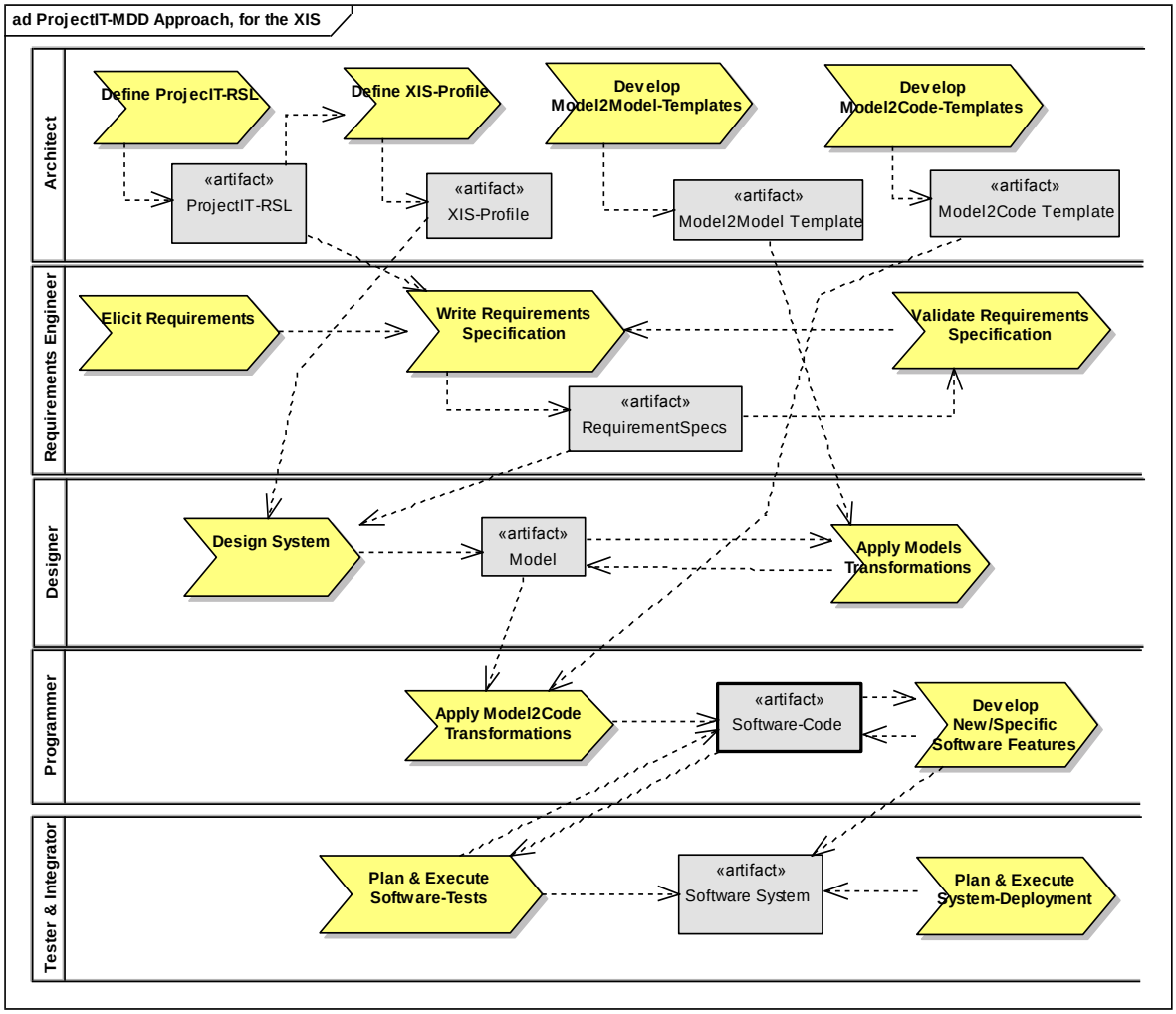


Figura 1.3 – Visão geral da abordagem ProjectIT (extraído de [7]).

O desenvolvimento de aplicações, segundo a abordagem ProjectIT, ocorre da seguinte forma: (1) o arquitecto define a linguagem de especificação dos requisitos, a linguagem de modelação num perfil UML e desenvolve os *templates* para as transformações modelo-para-modelo e modelo-para-código; (2) o engenheiro de requisitos faz o levantamento dos requisitos; (3) o designer modela a aplicação com base nos requisitos, utilizando uma ferramenta CASE para modelação de UML com transformações modelo-para-modelo; (4) o programador executa as transformações modelo-para-código e implementa os requisitos que não são suportados ao nível da modelação; e finalmente (5) o engenheiro de testes verifica se o sistema criado corresponde ao sistema esperado, com base nos requisitos definidos inicialmente, e o integrador faz a instalação da aplicação.

1.2 Objectivos

Este trabalho tem como principal objectivo continuar a desenvolver e integrar no ProjectIT a componente de geração automática de artefactos, com o objectivo de suportar as actividades relacionadas com a geração, da abordagem ProjectIT.

Desta forma, pretende-se neste trabalho atingir os seguintes objectivos individuais: (1) a redefinição e suporte da linguagem de modelação; (2) o estudo e aplicação de técnicas de transformação modelo-para-código; (3) a criação de interfaces de utilizador para gestão e execução de transformações modelo-para-código; e (4) o desenvolvimento de aplicações segundo a abordagem ProjectIT.

A justificação para o ponto (1) prende-se com a necessidade de se alterar significativamente a forma de modelação das interfaces de utilizador, para permitir ao *designer* modelar com maior detalhe as interfaces de utilizador.

No ponto (2), pretende-se acompanhar a evolução tecnológica desta área, dado que muitos projectos relacionados com este, tais como o openArchitectureWare (oAW) [w1] e o OptimalJ [w3], utilizam a tecnologia de definição de transformações modelo-para-código através de *templates* com sucesso.

Para justificar o ponto (3), contribuiu significativamente o trabalho desenvolvido recentemente no TFC anterior [6], que consistiu no desenvolvimento de uma plataforma – o Eclipse.NET [9] – para integrar os vários componentes do ProjectIT. Desta forma, pretende-se refazer as interfaces de utilizador para que este componente fique integrado com os restantes, através dos mecanismos providenciados pela nova plataforma.

Finalmente, o ponto (4) justifica-se para validar o trabalho desenvolvido nos pontos anteriores. Pretende-se utilizar a aplicação *MyOrders* para este efeito, porque tem sido a aplicação de exemplo utilizada nos TFCs anteriores.

1.3 Estrutura

Este relatório apresenta o estado final deste trabalho e está dividido em seis capítulos. No capítulo seguinte, apresentam-se os conceitos e as ferramentas usadas no trabalho. No capítulo 3, descreve-se a linguagem de modelação. No capítulo 4, apresenta-se a ferramenta desenvolvida. No capítulo 5, descrevem-se as transformações modelo-para-código. Finalmente, no capítulo 6 enunciam-se as conclusões e sugere-se trabalho futuro.

2 Estado da Arte

Este trabalho integra aspectos provenientes de várias áreas de conhecimento. Neste capítulo, descrevem-se os conceitos que mais impactos têm na solução proposta deste trabalho para o problema do desenvolvimento de *software*.

2.1 Model-Driven Development

O *Model-Driven Development* (MDD) [1] é um paradigma de desenvolvimento de *software* que pretende utilizar modelos para representar as aplicações, e utilizar os modelos criados em ferramentas que apliquem técnicas de geração automática [10] para produzirem os artefactos da aplicação. Os modelos podem ser representados textualmente ou graficamente [2].

O *Model-Drivel Architecture* (MDA) [w5] é uma abordagem da *Object Management Group* (OMG) [w6] para o desenvolvimento de *software*, baseada no paradigma MDD. A base do MDA é separar a especificação das funcionalidades da aplicação, dos detalhes de implementação das funcionalidades na plataforma, para que as funcionalidades e a plataforma da aplicação possam evoluir independentemente. Para tal, as aplicações são especificadas por modelos que são interpretados por máquinas computacionais para produzirem os artefactos da aplicação para a plataforma escolhida [3], se a plataforma evoluir, utilizam-se os mesmos modelos para gerar a aplicação para a nova plataforma. Com esta abordagem o MDA pretende alcançar a portabilidade, interoperabilidade e reusabilidade dos modelos através da separação arquitectural dos problemas. Segundo a abordagem MDA, existem três tipos de modelos [3]: *Computation Independent Model* (CIM), *Platform Independent Model* (PIM) e *Platform Specific Model* (PSM). O CIM foca a modelação no ambiente e nos requisitos da aplicação. O PIM modela as funcionalidades da aplicação. O PSM modela os detalhes da plataforma usada pela aplicação. O MDA tem um conjunto de padrões para transformar os modelos de uma representação para outra.

Recentemente a Microsoft lançou um conjunto de metodologias para o MDD, mas com uma visão ligeiramente diferente da OMG. Para a Microsoft a linguagem de modelação deve ser criada especificamente para resolver um conjunto restrito de problemas relativos a um domínio de aplicação, isto é deve ser desenvolvida uma *Domain Specific Languages* (DSL) [w7]. Uma DSL é uma linguagem de computação que serve para resolver um tipo específico de problema, ao contrário de uma *General Purpose Language* (GPL) que pretende resolver qualquer tipo de problema. A estratégia da Microsoft nesta área é fornecer ferramentas que

ajudam o desenvolvimento de DSLs e das ferramentas que utilizam a DSL desenvolvida [w7] para modelar as aplicações.

2.1.1 openArchitectureWare

O openArchitectureWare [w1] é uma plataforma baseada nos princípios do MDA/MDD. Suporta linguagens para definição de modelos, bem como de transformações modelo-para-modelo (*Xtend*) e modelo-para-código (*Xpand2*). Está desenvolvida na plataforma Java e os editores suportados são desenvolvidos na plataforma Eclipse. Os processos de geração são controlados por um motor de *workflows*, que é configurado através de ficheiros XML.

A linguagem *Xpand2* tem uma abordagem para a definição de transformações modelo-para-código do tipo *template.* A Figura 2.1 mostra um exemplo de um *template Xpand2..*

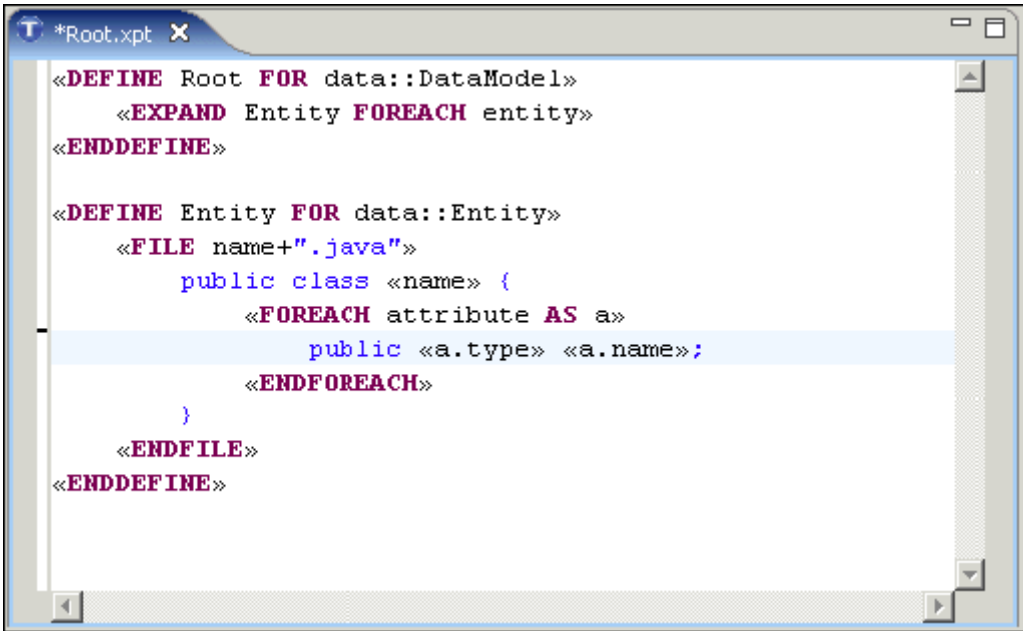


Figura 2.1 – Exemplo de um *template Xpand2* (extraído de [w2]).

2.2 Unified Modeling Language

O *Unified Modeling Language* (UML) [w13] é uma linguagem de modelação que funciona essencialmente através de representações gráficas. É um standard da OMG [w6] e encontra-se actualmente na versão 2.0. Os modelos criados em UML estão divididos em diagramas que correspondem às várias perspectivas de análise do sistema em estudo. Os diagramas modelam essencialmente duas características dos sistemas: o comportamento e a estrutura.

O uso mais frequente do UML é como uma linguagem para a comunicação dos sistemas entre os participantes dos projectos de tecnologias de informação. Caso não satisfaça todos os requisitos de comunicação dos interessados, o UML pode ser estendido através de um mecanismo de criação de perfis [11].

Recentemente, têm-se vindo a desenvolver técnicas de reutilização dos modelos UML para se aplicar técnicas de geração automática de artefactos (e.g. código fonte, documentação, testes, etc.). O MDA [w5] é um exemplo concreto que usa o UML como linguagem de modelação das aplicações.

Os modelos UML podem ser criados usando simplesmente "lápis e papel"; existem também ferramentas CASE especializadas na manipulação de modelos UML que oferecem algumas funcionalidades adicionais, tais como validação e importação/exportação de modelos.

2.3 Templates

Os *templates* são definições de transformações modelo-para.código, usam uma abordagem diferente da abordagem que se baseia na escrita de programas. Um *template* [12] é um exemplar do artefacto (e.g. página HTML, ficheiro de código fonte, etc.) com acções embutidas que um motor avalia quando faz o processamento do *template*. Enquanto que um programa executa instruções de escrita que produzem o artefacto. Um JSP [w9] é um exemplo de *template*.

Na definição dos *templates*, existem motivações para adoptarmos regras que separem as instruções de interpretação do *input* (modelo) das de especificação do *output* (vista) [12], e.g. encapsulamento, clareza, reutilização, manutenção, segurança, performance, etc. As regras que permitem fazer esta separação são [12]: (1) a vista não pode modificar directamente os objectos de dados do modelo, ou modificar indirectamente por invocação de métodos no modelo que causem efeitos colaterais; (2) a vista não pode fazer computações de valores que dependem do modelo; (3) a vista não pode comparar valores que dependem do modelo; (4) a vista não pode fazer suposições acerca dos tipos dos dados; e (5) os dados do modelo não devem ter informação de apresentação.

2.4 Plataforma .NET

A plataforma .NET [w8] é um ambiente para o desenvolvimento e execução de aplicações informáticas. É constituída por dois componentes: (1) o *Common Language Runtime* (CLR), que é o ambiente de desenvolvimento e execução das aplicações; e (2) a *Framework Class*

Library (FCL), que é a biblioteca com as funcionalidades da plataforma que serve para desenvolver as aplicações.

Figura 2.2 – Plataforma .NET (extraído de [w8])

O CLR disponibiliza um conjunto de ferramentas para ajudar no desenvolvimento e execução das aplicações.

A FCL disponibiliza, entre outras coisas, funcionalidades para a construção de aplicações *WindowsForms* e *WebForms*. As *WindowsForms* são aplicações *rich-client*, i.e. que se executam num ambiente com um comportamento semelhante ao do sistema operativo. As WebForms são aplicações *web-client*, i.e. que se executam num ambiente do tipo cliente-servidor.

2.5 Eclipse.NET

O Ecipse.NET [9] é uma plataforma que fornece um ambiente para o desenvolvimento e execução de aplicações informáticas, que funcionem de forma integrada. As aplicações são desenvolvidas na forma de *plugins* que estendem as funcionalidades fornecidas pela plataforma base. Os *plugins* comunicam entre si através de mecanismos providenciados pela plataforma, os pontos de extensão, onde cada *plugin* indica os pontos de extensão que fornece aos restantes *plugins* da plataforma para comunicarem com ele. As interfaces de utilizador dos *plugins* são desenvolvidas através das bibliotecas JFace/SWT.

Esta plataforma foi desenvolvida no âmbito de um TFC anterior [6] que consistiu na conversão de uma versão do Eclipse da plataforma Java para a plataforma .NET.

3 Linguagem de Modelação

No contexto deste trabalho, o objectivo da linguagem de modelação é permitir ao *designer* criar o modelo da aplicação, de modo a que este seja mais abstracto e independente da plataforma. Por “mais abstracto” entende-se a possibilidade de ser modelado com padrões mais ricos do que os de controlo de fluxo, de excepções, etc., providenciados pelas linguagens de programação, tanto para a definição do modelo de domínio como das interfaces de utilizador. Ser “independente da plataforma” significa que o modelo é válido para qualquer plataforma de um nível de abstracção inferior que suporte esta linguagem, no sentido em que a aplicação resultante fique com a mesma funcionalidade em qualquer das plataformas onde seja instalada.

Um primeiro aspecto a considerar nas linguagens de modelação é o que se pretende modelar do sistema. No interesse deste trabalho, pretende-se modelar aplicações informáticas interactivas de gestão de informação, com suporte para definição dos mecanismos de controlo de acesso e da definição dos detalhes do aspecto das interfaces de utilizador.

Outro aspecto a considerar nas linguagens de modelação é se a representação dos modelos é textual ou gráfica [13]. As **linguagens textuais** são baseadas na definição de gramáticas, permitem definir rigorosamente e formalmente os modelos, dado que existem técnicas já bem estudadas para definição de gramáticas, onde é possível caracterizar logo à partida a expressividade da mesma. No entanto, as **linguagens gráficas**, baseadas na definição de diagramas (e.g. através de metamodelos), permitem comunicar mais facilmente os modelos complexos, mas em contrapartida são mais difíceis de definir e caracterizar ao nível da expressividade. Neste trabalho, pretende-se que a linguagem de modelação seja gráfica.

As linguagens gráficas são normalmente definidas em perfis UML, para serem utilizadas em ferramentas *Computer Aided Systems Engineering* (CASE) de UML, tais como o Rational Rose, Enterprise Architect, ou o ProjectIT-Studio/UMLModeler, esta última ainda em fase de desenvolvimento no GSI do INESC-ID.

Numa primeira tentativa de satisfação destes requisitos, desenvolveu-se a linguagem XIS *(eXtreme modeling Interactive Systems)*.

3.1 Linguagem XIS – Versão 1

A linguagem XIS é uma linguagem de modelação gráfica desenvolvida no GSI do INESC no âmbito de 3 TFCs anteriores [4][5][6]. A característica mais visível desta linguagem é a de ser fortemente inspirada no padrão *MODEL-VIEW-CONTROLLER* (MVC) [14]. O *designer* modela o domínio da aplicação, e sobre o domínio define entidades de negócio, controlos e vistas. Esta forma de modelar as aplicações é bastante modular; no entanto, a forma como se definem as vistas nesta linguagem apresenta algumas limitações no que respeita à modelação das interfaces, pois as vistas são apenas uma combinação de uma entidade de negócio com um controlador, ficando por definir os aspectos relativos ao conteúdo e disposição dos elementos. A especificação completa da linguagem XIS encontra-se no relatório do TFC de Lemos e Matias [4].

Tendo em conta estes problemas, desenvolveu-se no âmbito deste trabalho, juntamente com uma equipa do GSI do INESC-ID, a linguagem XIS2. Nesta nova versão, existe uma melhoria relativamente à modelação das interfaces de utilizador, visto que as interfaces passaram a ser definidas segundo o padrão *COMPOSITE* [15], i.e. através da composição de elementos complexos (e.g. formulários, tabelas, etc.) com elementos simples (e.g. caixas de texto, botões, colunas, etc.), o que permite definir interfaces com maior detalhe, tanto ao nível do conteúdo como da disposição dos elementos. Além do mais, associado a cada elemento da interface é também suportado o controlo de acesso que os utilizadores têm, dependendo dos respectivos papéis.

3.2 Linguagem XIS – Versão 2 (ou XIS2)

Na linguagem XIS2 [16], o modelo é organizado em torno de três conjuntos de vistas, como sugere a Figura 3.1: (1) a vista das entidades, (2) a vista dos casos de utilização e (3) a vista das interfaces de utilizador. A vista das entidades permite definir a estrutura da informação que a aplicação deve gerir, e as acções de gestão que se podem realizar sobre cada entidade. A vista dos casos de utilização permite definir os actores e as acções que cada actor pode executar, tipicamente de modo relativo, na gestão das entidades. Finalmente, a vista das interfaces de utilizador permite definir os espaços de interacção para gestão das entidades e a respectiva navegabilidade entre esses espaços de interacção.

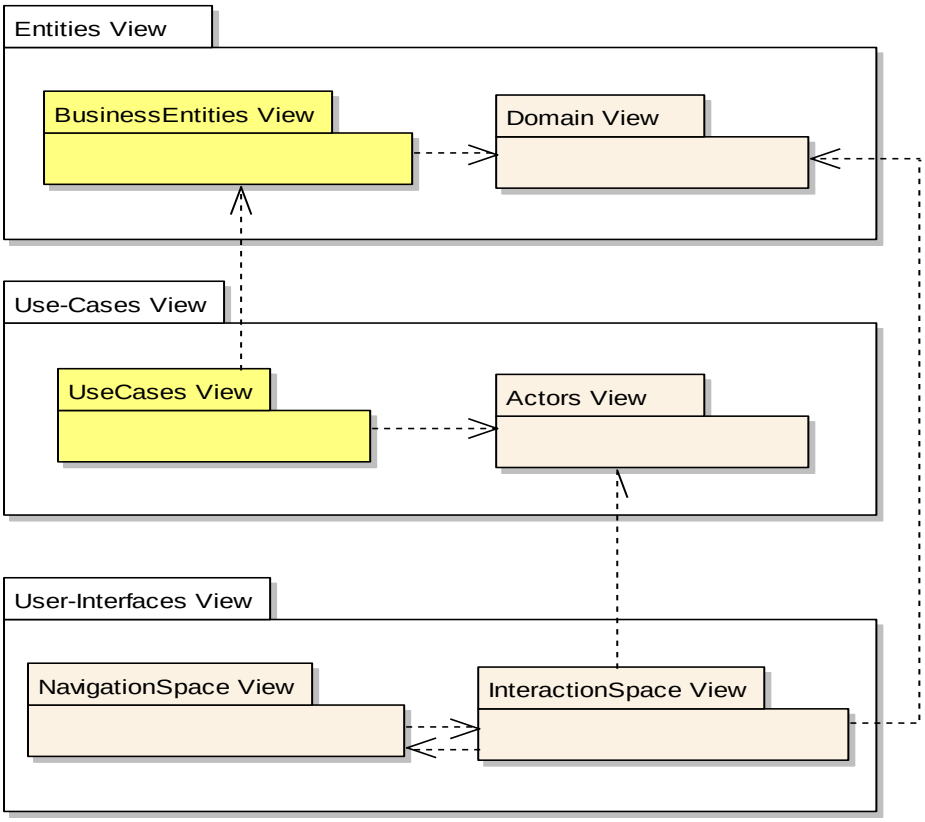


Figura 3.1 – Vistas da linguagem XIS2 (extraído de [7]).

Para definir completamente o modelo da aplicação, o *designer* tem de definir as vistas necessárias para as transformações modelo-para-código (ver Figura 3.2), que são: a vista do domínio, a vista dos actores e as vistas das interfaces de utilizador. Uma vez que a tarefa de modelar as interfaces de utilizador, segundo o padrão *COMPOSITE*, pode ser dispendiosa ao nível do esforço humano requerido, a linguagem XIS2 permite definir vistas auxiliares de entidades de negócio e casos de utilização que, quando usadas em transformações modelo-para-modelo, produzem automaticamente as interfaces de utilizador, permitindo, no entanto, a posterior intervenção do *designer* na configuração e refinamento das interfaces.

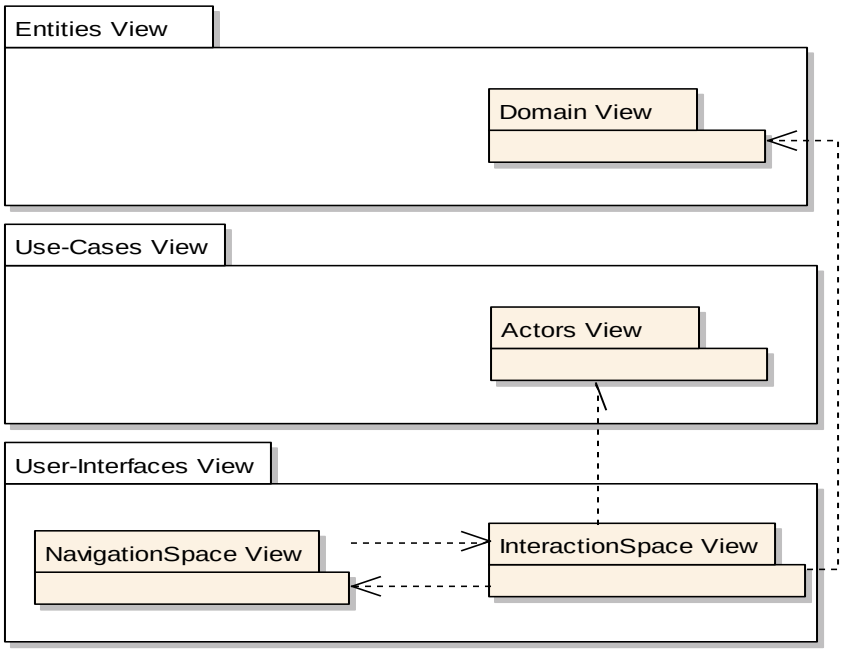


Figura 3.2 – Conjunto mínimo de modelos necessários para a geração de código (extraído de [7]).

Desta forma, o *designer* tem duas abordagens na produção dos seus modelos [7]: a "*dummy*" e a "*smart*". Segundo a primeira abordagem, o *designer* define no modelo as vistas necessárias para as transformações modelo-para-código (ver Figura 3.2), o que inclui definir as vistas das interfaces de utilizador, tarefa essa que consome bastante tempo. Assim sendo, recomenda-se a segunda abordagem (ver Figura 3.3), que corresponde a definir vistas auxiliares para as transformações modelo-para-modelo, vistas essas que permitem a geração das vistas das interfaces de utilizador, bastando depois a intervenção do designer em alguns aspectos mais específicos. As vistas opcionais podem ser criadas mesmo quando se usa a abordagem "*dummy*", pois são também úteis para efeitos de documentação.

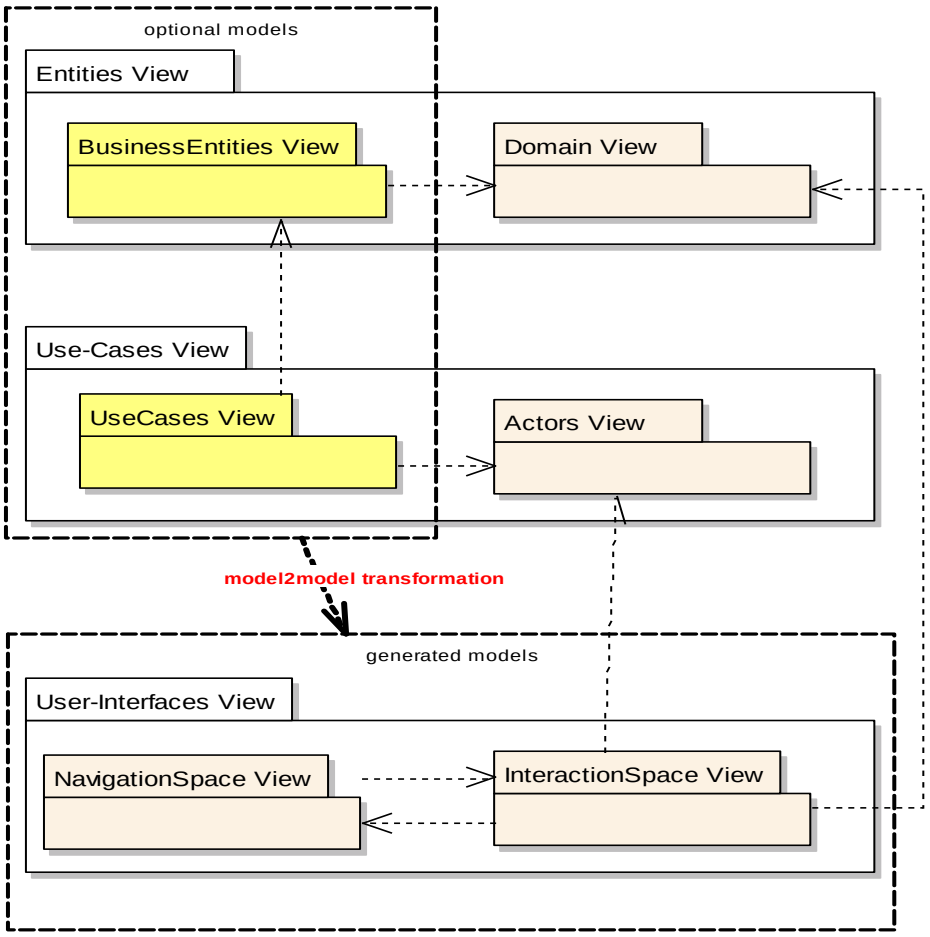


Figura 3.3 – Vistas auxiliares das transformações modelo-para-modelo (extraído de [7]).

Nas subsecções seguintes, descrevem-se as vistas da linguagem XIS2, acompanhadas de exemplos retirados da aplicação *MyOrders2* (ver Anexo C). No Anexo B, está o manual de referência do perfil UML da linguagem XIS2.

3.2.1 Vista de Domínio

A vista de domínio *(DomainView)*, ilustrada na Figura 3.4, define as entidades (*XisEntity*) da aplicação, com os respectivos atributos (*XisEntityAttribute*), associações (*XisEntityAssociation*) e relações de herança (*XisInheritance*).

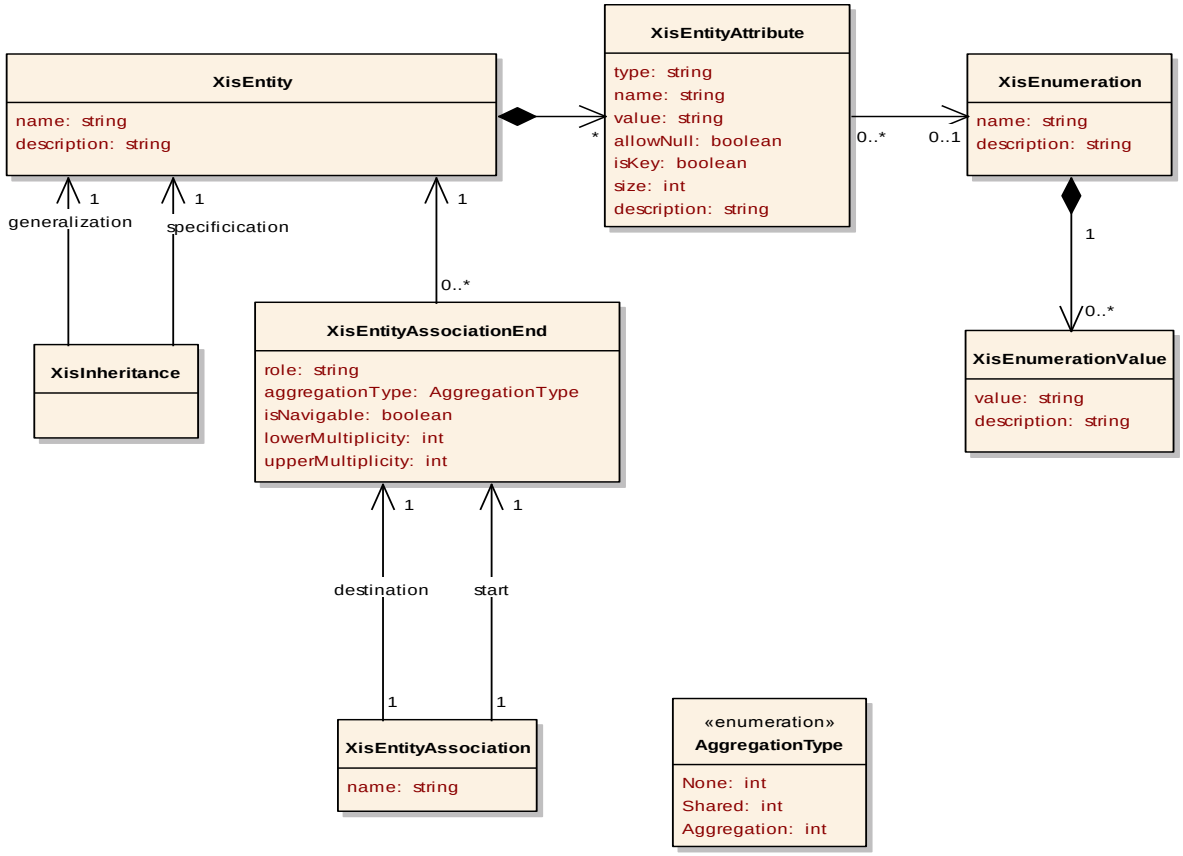


Figura 3.4 – Vista de domínio (extraído de [16]).

Os atributos das entidades podem ser de tipos primitivos de UML (e.g. *String*, *Boolean*, etc.) ou de tipos enumerados, definidos nesta vista através de *XisEnumeration* e *XisEnumerationValue*.

As associações entre as entidades são do tipo binário. Para se simular uma associação do tipo n-ária, é preciso ter uma entidade que represente a associação, e depois definir associações binárias entre esta entidade representante da associação e as entidades participantes na associação n-ária.

A Figura 3.5 mostra o modelo de domínio da aplicação *MyOrders2* definido com o perfil UML da linguagem XIS2. O modelo tem os estereótipos do perfil UML aplicados aos elementos do diagrama, mas as marcas com valor não são visíveis directamente nos diagramas.

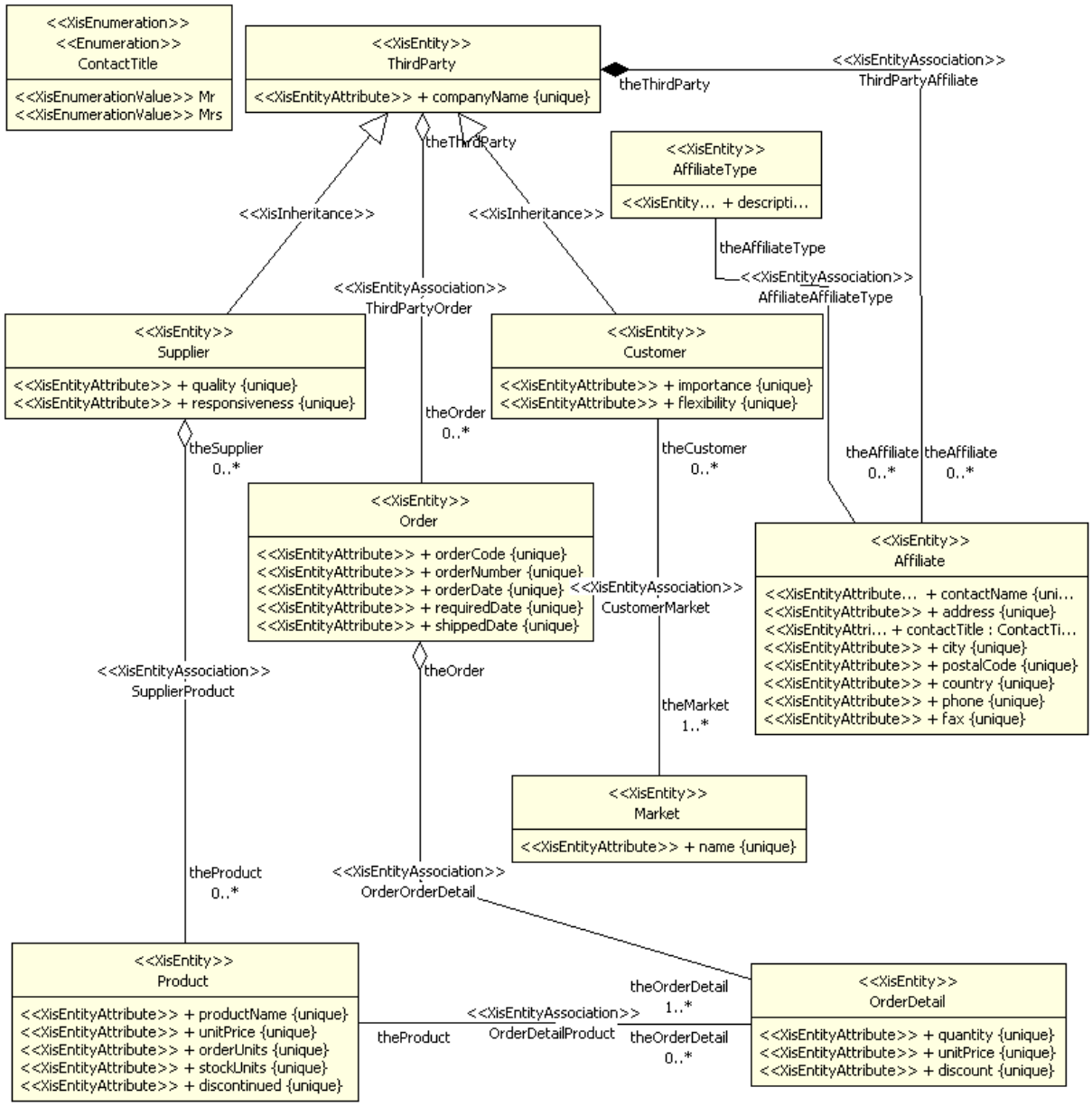


Figura 3.5 – Modelo de domínio do *MyOrders2*.

3.2.2 Vista de Entidades de Negócio

A vista de entidades de negócio *(BusinessEntities View)* define entidades mais abstractas *(XisBusinessEntity)* no âmbito do negócio, que relacionam as entidades de domínio *(XisEntity)* com as operações *(XisStandardActions)* que se podem efectuar. As entidades desta vista são apresentadas na Figura 3.6.

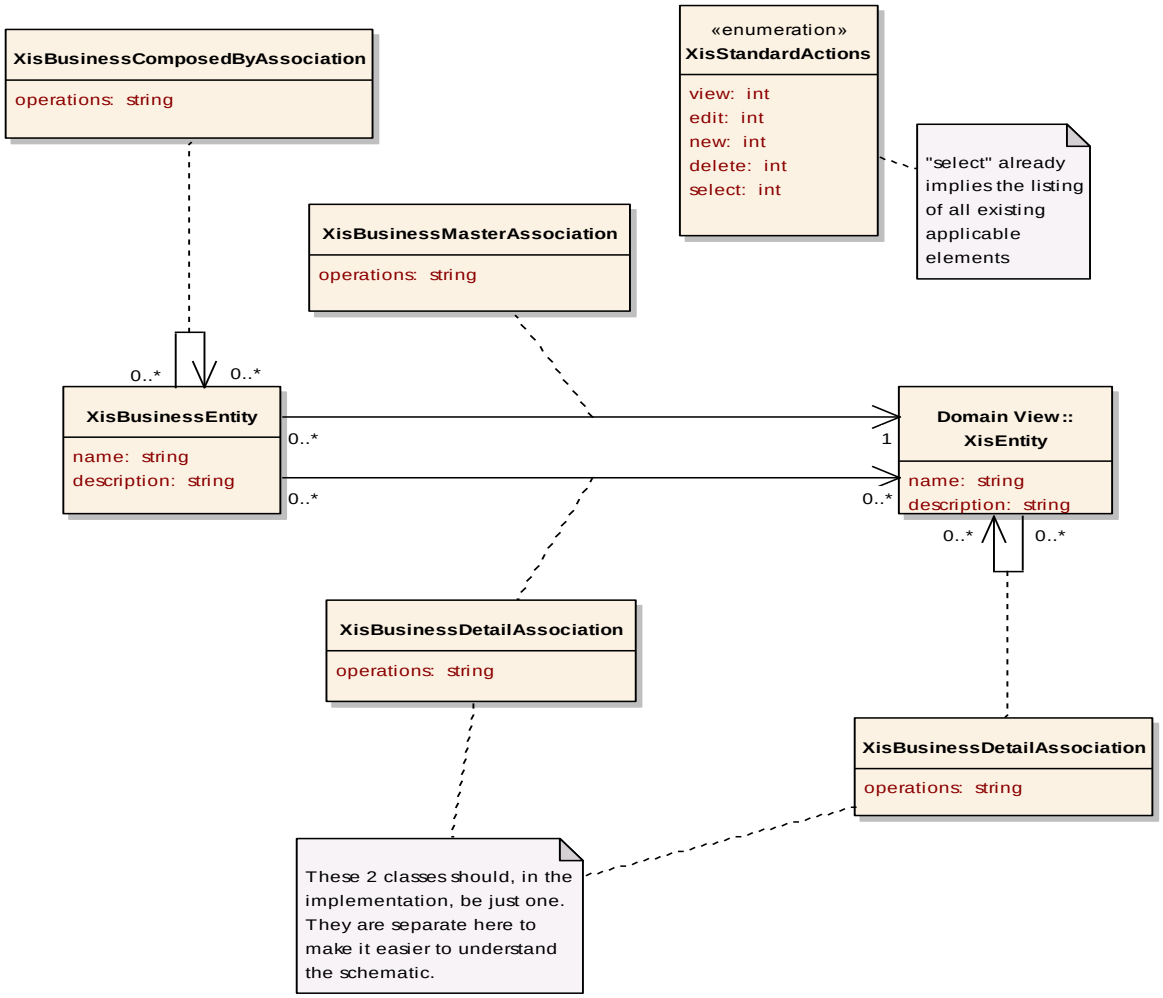


Figura 3.6 - Vistas de entidades de negócio (extraído de [16]).

3.2.3 Vista de Actores

A vista de actores *(Actors View)*, ilustrada na Figura 3.7, define os actores (*XisActor*) que intervêm na aplicação, que servem para modelar os aspectos relativos ao controlo de acesso, na vista de espaços de interacção. As relações de herança (*XisInheritanceAssociation*) entre os actores permitem a reutilização das permissões de acesso, o que significa que o actor-filho tem acesso aos mesmos locais que o actor-pai mais os que forem definidos especificamente para ele.

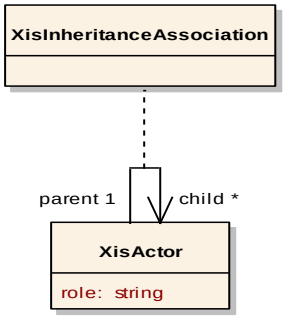


Figura 3.7 - Vista dos actores (extraído de [16]).

3.2.4 Vista de Casos de Utilização

A Figura 3.8 mostrar a vista dos casos de utilização, que define as operações (*XisOperatesOnAssociation*) que os actores podem realizar (*XisPerformsAssociation*) nas entidades de negócio. O *XisUseCase* agrupa conjuntos de operações no contexto de operações mais complexas.

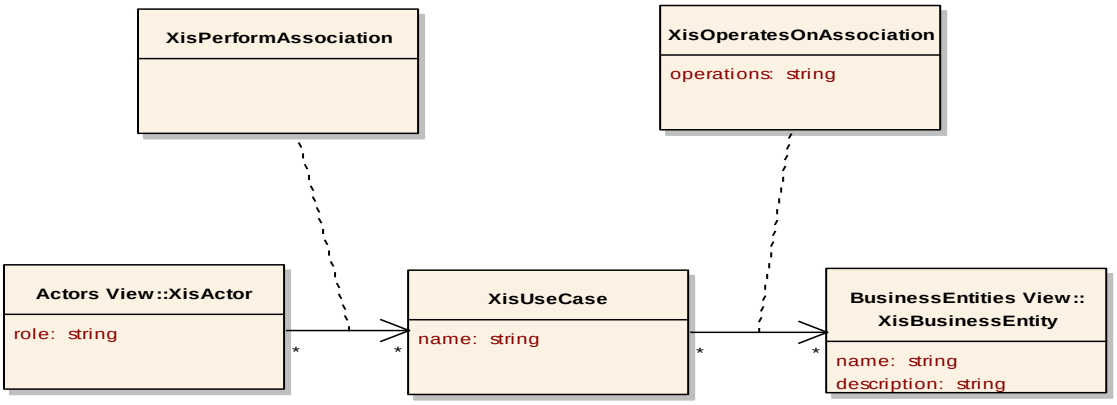


Figura 3.8 - Vista dos casos de utilização (extraído de [16]).

3.2.5 Vista de Espaços de Interacção

A vista de espaços de interacção *(InteractionSpace View)* define as interfaces *(XisInteractionSpace)* que gerem as entidades de domínio, bem como as permissões de acesso *(XisElementRight)* aos elementos do espaço de interacção (ver Figura 3.9).

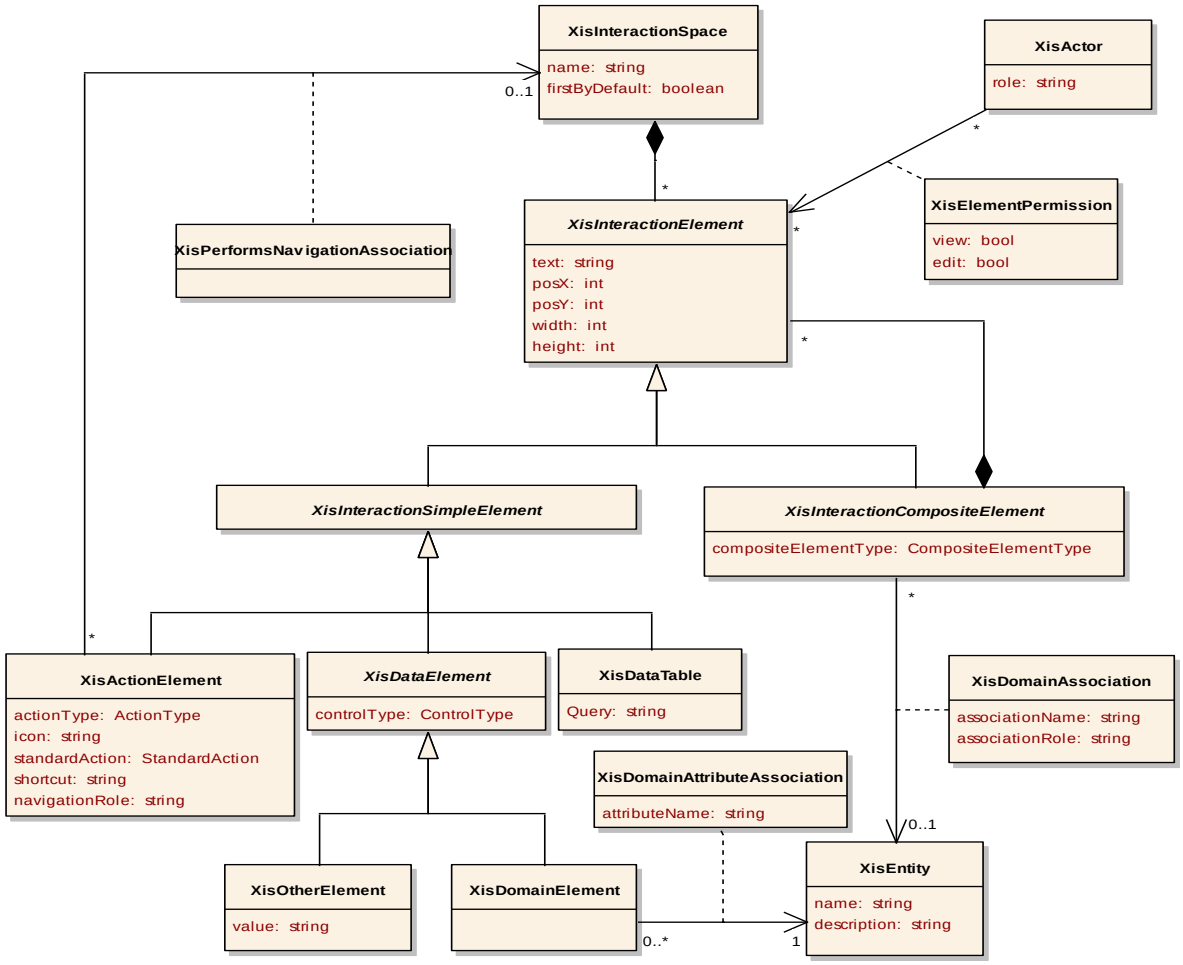


Figura 3.9 - Vista de espaços de interacção (extraído de [16]).

Esta vista baseia-se essencialmente no padrão *COMPOSITE* [15], tem uma classe que desempenha o papel de *Component*, a *XisInteractionElement*, que define para todos os elementos de interacção: (1) um texto que o caracteriza, (2) a posição e (3) o tamanho. O papel *Composite* é desempenhado pela classe *XisInteractionCompositeElement*, que contém outros objectos da mesma classe ou *Leafs* que herdam da classe *XisInteractionSimpleElement*. Do conjunto de classes que desempenham o papel de *Leaf*, as mais importantes são a *XisActionElement* e a *XisDomaniElement*.

Os elementos são caracterizados por um conjunto de atributos de tipo enumeração (ver Figura 3.10), de modo a se evitar a explosão de classes do metamodelo.

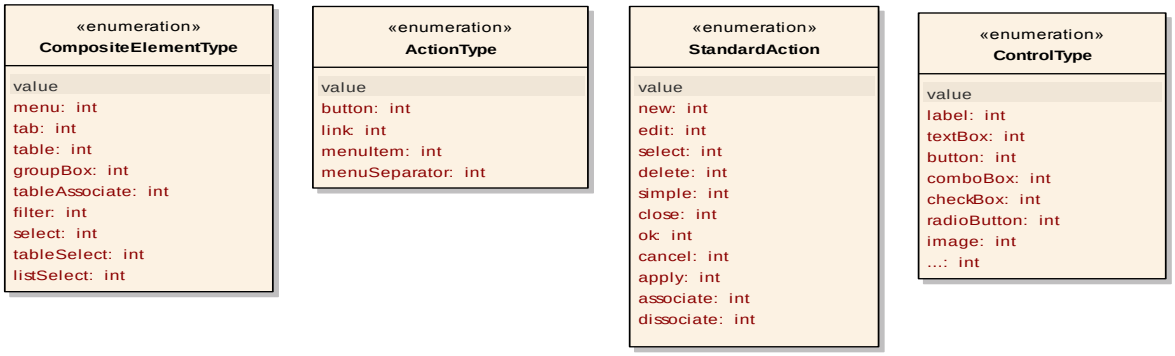


Figura 3.10 - Tipos enumerados da vista de espaços de interacção (extraído de [16]).

A Figura 3.11 mostra um extracto do modelo da interface de utilizador para o formulário da entidade fornecedor da aplicação *MyOrders2*. A composição dos elementos da interface é determinada visualmente através da seguinte regra: um elemento está dentro de outro elemento se todos os seus vértices estão dentro do outro.

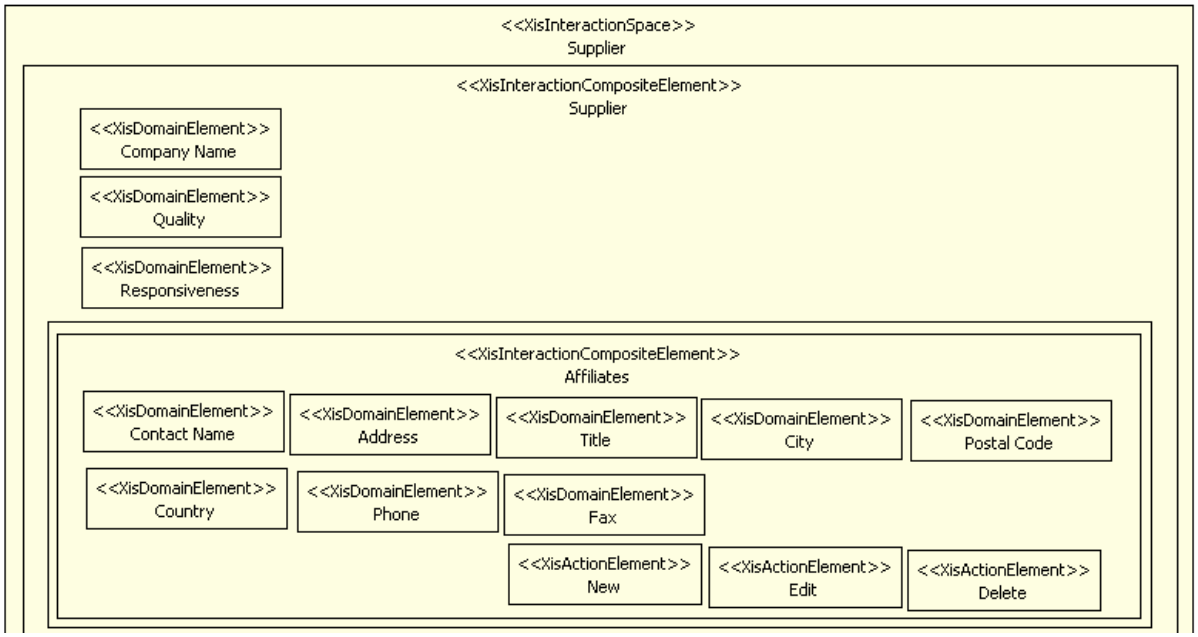


Figura 3.11 – Extracto do modelo da interface de utilizador para a entidade fornecedor.

Os elementos podem ser *mapeados* para entidades da vista de domínio, através de *XisDomainAssociations* e de *XisDomainAttributeAssociations*. Este *mapeamento* é importante para se saber de onde se carregam os dados da interface e onde se devem guardar no repositório. A Figura 3.12 mostra um extracto do *mapeamento* da interface de utilizador para a entidade fornecedor da aplicação *MyOrders2*.

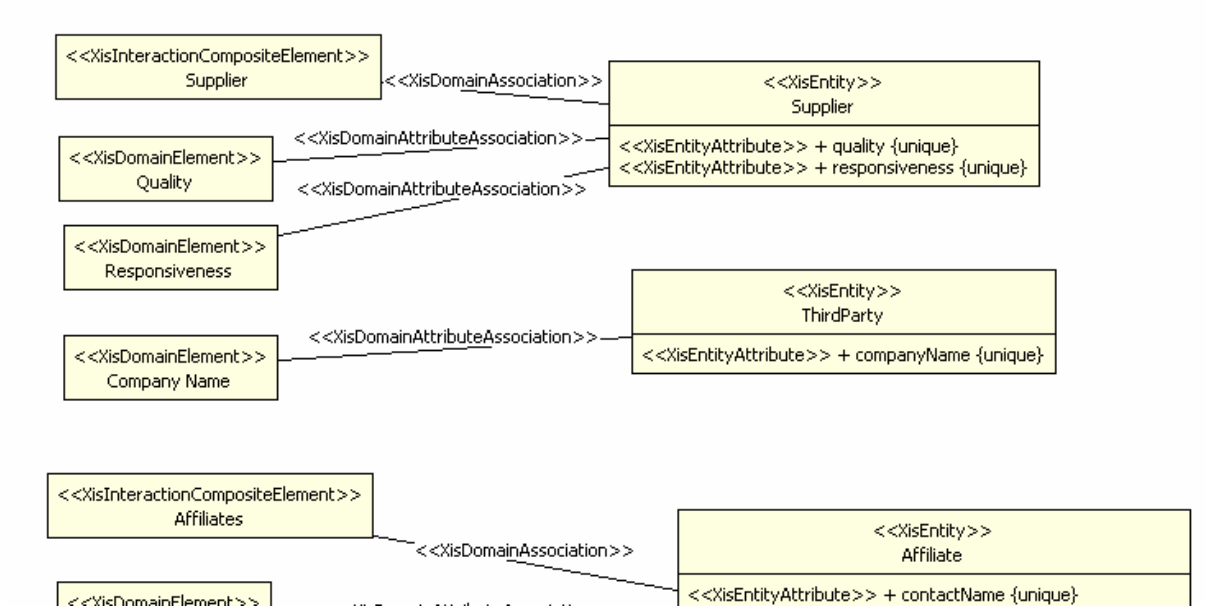


Figura 3.12 – Extracto do *mapeamento* da interface de utilizador para a entidade fornecedor.

As acções podem ter uma *XisPerformsNavigationAssociation*, para causarem uma transição de um espaço de interacção para outro.

3.2.6 Vista de Navegação de Espaços

A vista de navegação de espaços *(NavigationSpace View)* permite definir as transições *(XisNavigationAssociation)* entre os espaços de interacção, através de *XisNavigationAssociation*. Cada associação de navegação tem o espaço de interacção de origem e o de destino, ilustrado na figura Figura 3.13 através da navegabilidade da associação.

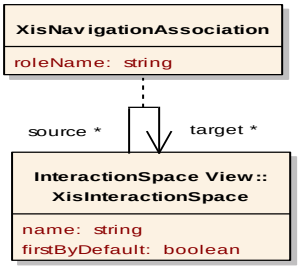


Figura 3.13 – Vista de espaços de navegação (extraído de [16]).

As navegações definidas nesta vista são usadas na definição das acções (*XisActionElement*) na vista de espaços de interacção.

4 ProjectIT-Studio/MDDGenerator

O ProjectIT-Studio/MDDGenerator é o componente do ProjectIT-Studio [7] responsável pela configuração e execução dos processos de geração das aplicações. O ProjectIT-Studio integra actualmente mais dois componentes, desenvolvidos no âmbito de outros trabalhos: (1) o ProjectIT-Studio/Requirements – componente responsável pela definição e gestão de requisitos; e (2) o ProjectIT-Studio/UMLModeler – componente responsável pela definição e gestão dos modelos.

Os componentes do ProjectIT-Studio são desenvolvidos em *plugins* para a plataforma Eclipse.NET, que como mostra a Figura 4.1, corre sobre a plataforma .NET 2.0. A decisão desta arquitectura de instalação partiu do grupo de investigação onde este trabalho estava inserido, e deveu-se sobretudo ao problema da integração dos componentes do ProjectIT-Studio [6].

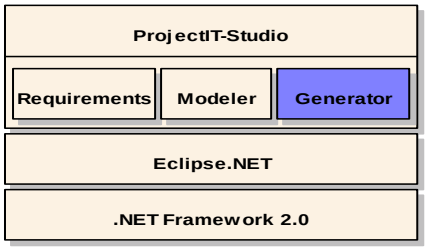


Figura 4.1 - Visão de instalação (adaptado de [7]).

O componente ProjectIT-Studio/MDDGenerator tem quatro módulos, ilustrados na Figura 4.2: (1) o XisMetamodel2 tem o metamodelo das entidades da linguagem XIS2, que é usado para definição dos *templates*; (2) o TemplateEngine tem o motor responsável pela interpretação dos *templates* que criam os artefactos da aplicação; (2) o Generator.Plugin é responsável pelo *plugin* para a plataforma Eclipse.NET, e fornece as interfaces de utilizador que gerem as arquitecturas de *software*, processos de geração e editam os *templates*; e (4) o Generator.Core é responsável pela execução dos processos de geração.

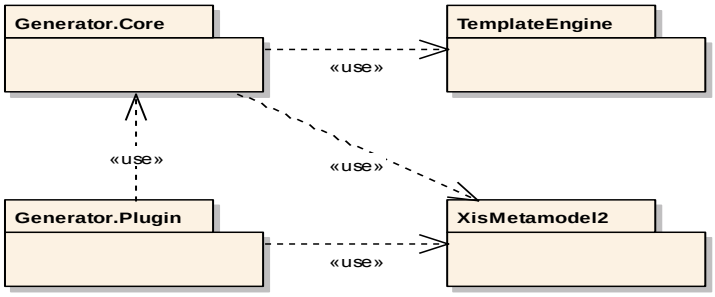


Figura 4.2 - Visão dos módulos do ProjectIT-Studio/MDDGenerator.

4.1 XisMetamodel2

O módulo XisMetamodel2 tem a implementação do metamodelo da linguagem de modelação XIS2, i.e. uma representação para os modelos numa rede de objectos, que é usada, entre outras coisas, na definição das transformações do tipo modelo-para-modelo ou modelo-para-código. A estrutura do metamodelo está de acordo com a definição da linguagem XIS2, que está na secção 3.2.

Este módulo tem também dois componentes que permitem: (1) ler/gravar o metamodelo de/para o formato XML, e (2) converter o modelo do metamodelo UML da ferramenta CASE para o metamodelo da linguagem XIS2. O primeiro componente é necessário para configurar os processos de geração. O segundo componente é necessário, porque os modelos criados na ferramenta CASE de UML com o perfil da linguagem XIS2 ficam guardados no metamodelo UML dessa ferramenta, mas as transformações, independentemente da ferramenta usada, são definidas com o metamodelo da linguagem de modelação, neste caso o da linguagem XIS2.

4.2 TemplateEngine

O TemplateEngine é o módulo do ProjectIT-Studio/MDDGenerator responsável pela gestão dos *templates* utilizados na geração das aplicações.

Um ***template*** é a definição de uma transformação modelo-para-código, escrita numa linguagem própria, semelhante à linguagem ASP (*Active Server Pages*) [w14]. A linguagem tem um conjunto de directivas para definir os *templates* (descritas no manual de utilizador que está no Anexo A), que são utilizadas juntamente com instruções de código na linguagem C#. Para ser executado, o *template* é previamente convertido num programa de

C# e compilado para uma biblioteca, da qual se podem instanciar objectos da classe que tem o programa com a transformação modelo-para-código.

Por si só o *template* é suficiente para definir estas transformações, mas de forma a satisfazer requisitos de reutilização e de separação de responsabilidades existe também o **filtro**, que é um programa escrito na linguagem C#, que pode ser usado (1) para definir operações comuns a vários *templates* e filtros, e (2) para criar vistas de entrada dos *templates* mais simplificadas e apropriadas, de modo a facilitar a definição dos mesmos. De forma semelhante aos *templates*, o filtro só é usado depois de previamente compilado, ficando também associado a uma biblioteca, que é importada pelos *templates* ou filtros que o utilizam.

A Figura 4.3 mostra as dependências que existem entre os *templates* e os filtros: o *template* depende de outros *templates* e de filtros; o filtro depende apenas de outros filtros. Estas dependências são reflectidas na definição dos artefactos, através do uso das directivas apropriadas: nos *templates*, as directivas *"Call"*, *"Include"* e *"Filter"*; nos filtros, a directiva *"Filter"*. O tipo da dependência entre os artefactos difere conforme sejam usadas as directivas *"Call"* e *"Include"*, ou a directiva *"Filter"*. A primeira é resolvida através do mecanismo de reflexão da plataforma .NET, enquanto que a segunda é resolvida na compilação das bibliotecas através da configuração de importações.

Figura 4.3 - Artefactos utilizados no módulo TemplateEngine.

As entidades que constituem este módulo são: (1) o *ITemplate*, que é a interface que representa um *template* depois de convertido em programa; (2) o *TemplateEngine*, que é responsável pela conversão dos *templates* em programas, e subsequente compilação e instanciação dos mesmos; (3) o *TemplateRegistry*, que é responsável pela gestão do registo de *templates* compilados; (4) o *IFilter*, que é a interface que os filtros implementam para poderem ser executados; (5) o *FilterEngine*, que é responsável pela compilação de filtros; e (6) o *FilterRegistry*, que é responsável pela gestão do registo de filtros compilados. A Figura 4.4 mostra as relações entre as entidades deste módulo; de notar que as interfaces

ITemplate e *IFilter* são apenas usadas para manipular as instâncias das classes definidas, respectivamente, nas bibliotecas (*assemblies)* do *TemplateEngine* e do *FilterEngine*.

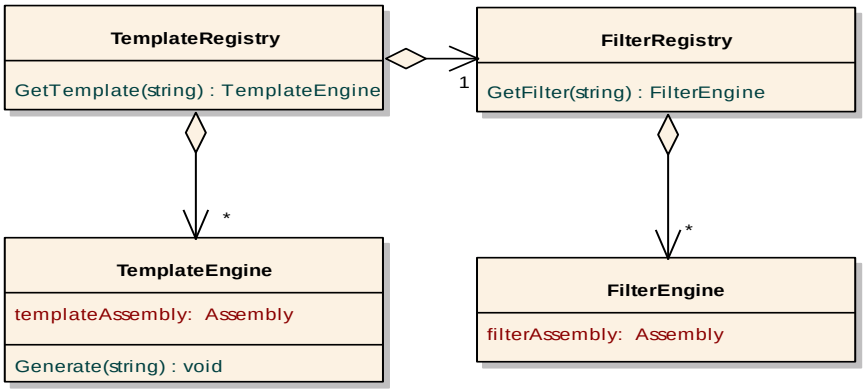


Figura 4.4 - Relações entre as entidades do módulo *TemplateEngine*.

O processo de execução de um *template* inicia-se quando uma entidade pede um *template* ao *TemplateRegistry* (ver Figura 4.5). O *TemplateRegistry* verifica então se tem no registo o *TemplateEngine* responsável pelo *template* pedido, se for esse o caso, então devolve-o, senão cria um *TemplateEngine* para adicionar ao registo, que passa a ser responsável pela conversão desse *template* num programa e pela subsequente compilação. Sempre que é necessário executar o *template*, o *TemplateEngine* cria uma instância da classe que implementa a interface *ITemplate*, que está na biblioteca gerida por ele.

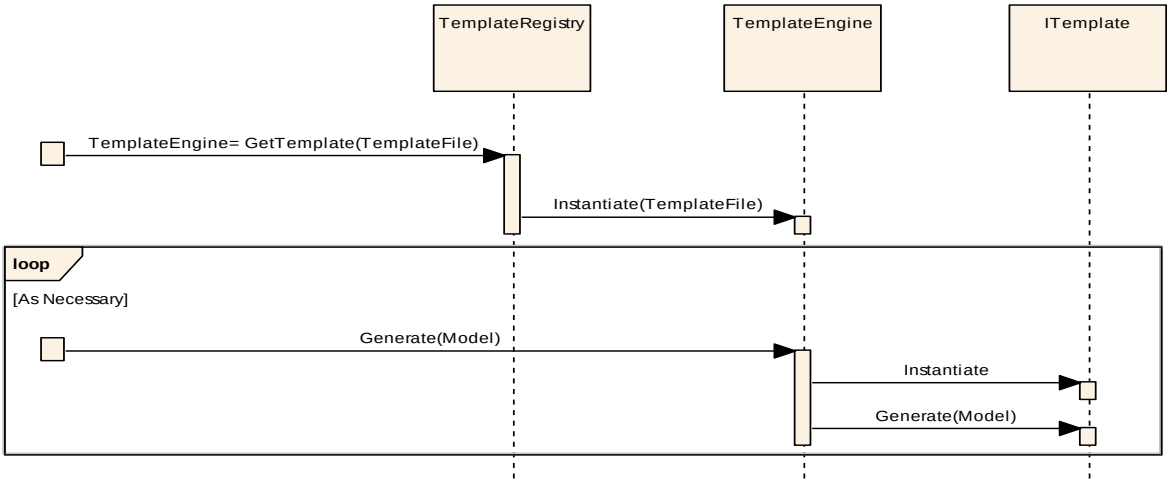


Figura 4.5 - Execução de *tempates*.

A entidade que inicia a execução de um *template*, também pode ser ela própria um *template*; tal acontece quando na definição do *template* se utiliza as directivas *"Call"* ou

"Include", que servem, respectivamente, para iniciar a execução de um *template*, ou para iniciar a execução de um *template* e incluir o seu resultado no local da chamada.

A dependência do tipo *template-template*, criada pela utilização das directivas *"Call"* e *"Include"*, é resolvida através dos mecanismos de reflexão da plataforma .NET. Na conversão destas directivas, o *TemplateEngine* insere as instruções do mecanismo de reflexão no código do programa. Uma vez que é utilizado este mecanismo, os *templates* podem executar-se mutuamente, com a restrição de não poderem partilhar os tipos de dados definidos internamente. Para contornar esta restrição, podem depender ambos de um filtro com a definição dos tipos de dados partilhados.

Por outro lado, a execução de um *template* também pode desencadear a compilação e execução de filtros (ver Figura 4.6), tal acontece quando na sua definição se usa a directiva *"Filter"*. Neste caso, o *TemplateEngine* responsável por este *template*, pede ao *FilterRegistry* o *FilterEngine* responsável pela compilação do filtro. Depois, no início da execução, o *template* cria uma instância do filtro e executa o método *Apply()* da interface *IFilter*.

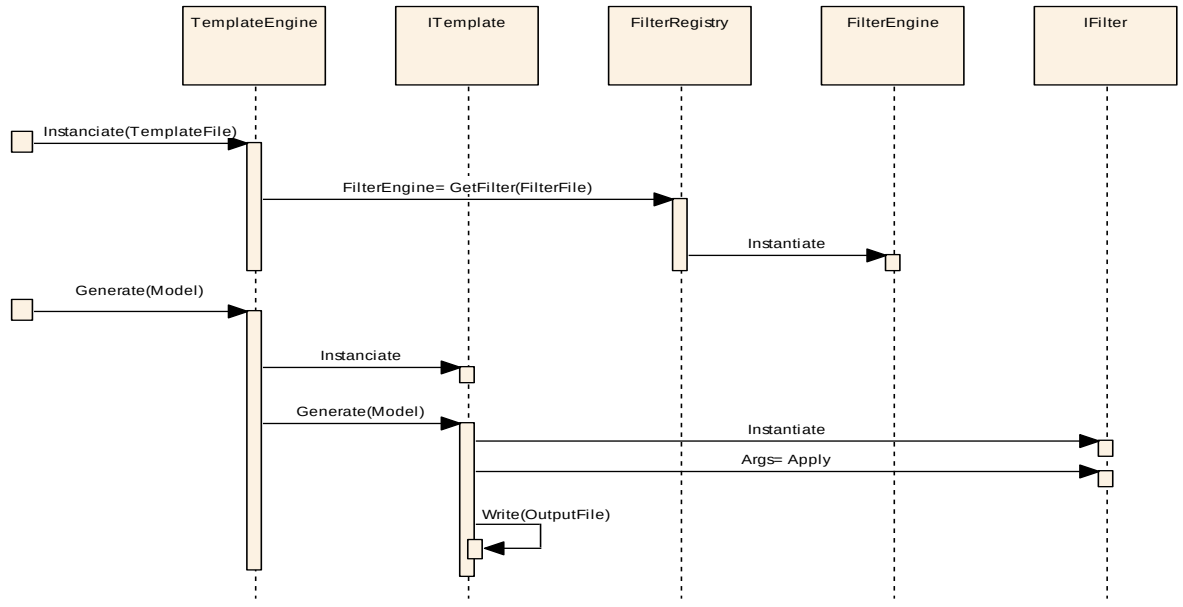


Figura 4.6 – Execução de filtros.

Esta dependência, do tipo *template-filtro*, é resolvida pelo *TemplateEngine* e pelo *FilterEngine* da seguinte forma: no momento da compilação da biblioteca do *TemplateEngine*, é importada a biblioteca que é gerida pelo FilterEngine, o que obriga a compilar os filtros antes dos *templates*. Desta forma, os *templates* podem usar directamente

os tipos de dados definidos nos filtros, sem necessidade de utilizarem mecanismos de reflexão.

A dependência do tipo *filtro-filtro* é resolvida utilizando a mesma técnica de importação da biblioteca durante a compilação do filtro. Contudo, os filtros não podem ter dependências mútuas, porque a importação de uma biblioteca é efectuada no momento da compilação, e isso implica a existência prévia da biblioteca que é importada.

Os *templates* e filtros podem depender do metamodelo da linguagem de modelação, e.g. do metamodelo da linguagem XIS2. Esta dependência é definida nos artefactos através da directiva *"Assembly"*, que é também resolvida através do mecanismo de importação de bibliotecas.

A Figura 4.7 mostra as dependências do tipo importação entre as bibliotecas geridas pelo *TemplateEngine* e pelo *FilterEngine*, e a biblioteca do metamodelo da linguagem de modelação. As dependências deste tipo obrigam a que no momento da compilação de uma biblioteca, todas as bibliotecas da qual esta dependa estejam previamente compiladas.

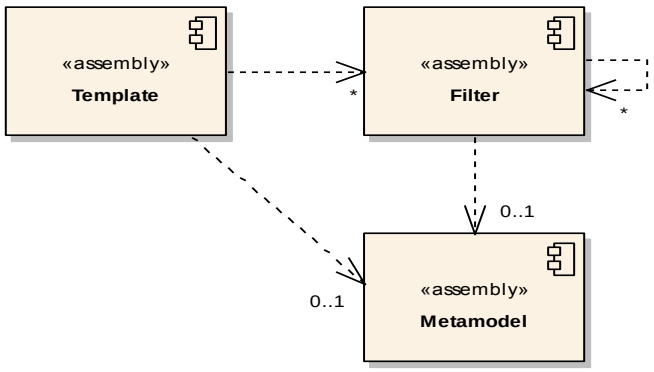


Figura 4.7 – Dependências do tipo importação entre as bibliotecas.

A detecção de erros de compilação nos *templates* e filtros fica condicionada ao momento da execução, isto porque o momento da compilação ocorre imediatamente antes do da execução (pois estes artefactos só são compilados quando são precisos). Num sistema de *pipeline*, onde numa primeira fase se efectua a compilação de todos os *templates* e, numa segunda fase se executam os que são necessários, os erros de compilação são detectados mais cedo, logo durante a primeira fase.

Este motor é uma adaptação do motor de *templates TemplateMaschine* [w10], com melhorias relativamente à: (1) separação de responsabilidades, através do mecanismo de filtros; (2) modularização dos *templates*, através das directivas *"Call"* e *"Include"*; e (3)

performance, através dos mecanismos de *cache* implementados nos registos de *templates* e filtros.

4.3 Generator.Plugin

O módulo Generator.Plugin é responsável pelo *plugin* para a plataforma Eclipse.NET e pelas interfaces de utilizador do ProjectIT-Studio/MDDGenerator.

As interfaces de utilizador são implementadas em editores desenvolvidos com a plataforma do Eclipse.NET. Estas interfaces permitem aos intervenientes técnicos da abordagem ProjectIT gerir (i.e. ver, criar, editar e apagar) os vários artefactos utilizados na configuração da geração, que são: (1) as arquitecturas de *software* – representações das plataformas para as aplicações; (2) os processos de geração – que são configurações que associam um modelo a uma arquitectura de *software*; e (3) os *templates* – que são definições das transformações modelo-para-código. Finalmente, também permitem desencadear e monitorizar a execução de um processo de geração.

Existem algumas dependências entre os artefactos criados pelos vários editores. A Figura 4.8 mostra que a os processos de geração dependem dos modelos e das arquitecturas de *software*, e que as *arquitecturas* de software dependem dos *templates*.

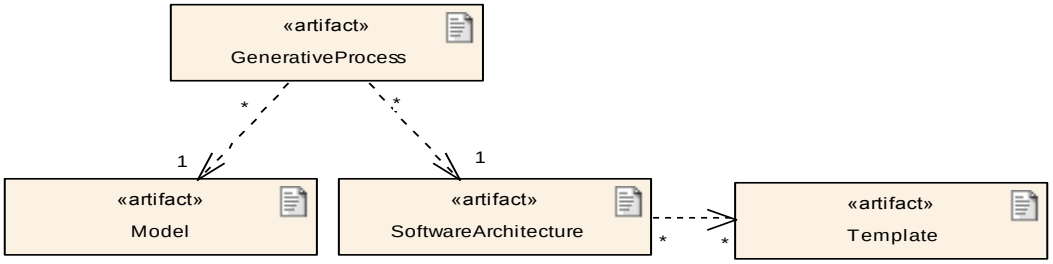


Figura 4.8 – Artefactos geridos.

4.3.1 Editor de Arquitecturas de *Software*

A arquitectura de *software* é uma representação da plataforma para a qual se pretende gerar a aplicação, que por sua vez é definida pelos *templates* que a constituem (ver Figura 4.9).

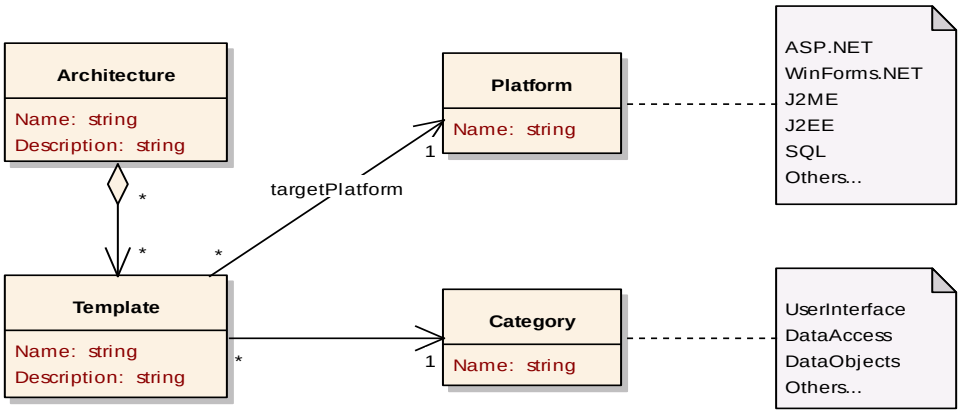


Figura 4.9 – Entidades persistentes da arquitectura de *software*.

O editor de arquitecturas de *software* permite ao arquitecto definir as arquitecturas de *software*, para tal ele pode, conforme mostra a Figura 4.10: (1) consultar a lista de *templates* disponíveis, e (2) inserir ou remover *templates* nas arquitecturas de *software*.

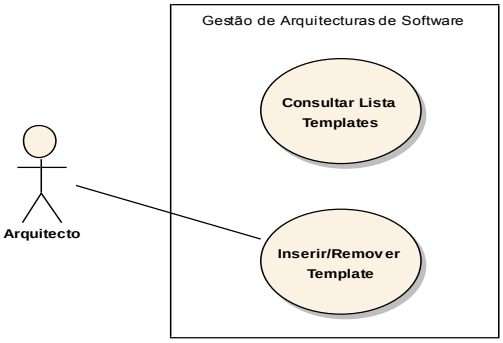


Figura 4.10 – Funcionalidades do gestor de arquitecturas de *software*.

Este editor herda da classe *"Editor"* da plataforma do Eclipse.NET, e a interface de utilizador é desenvolvida com as bibliotecas JFace e SWT, que acompanham esta plataforma.

A Figura 4.11 mostra a interface de utilizador do editor de arquitecturas de *software*. O utilizador pode ver os *templates* que fazem parte da arquitectura através da lista localizada do lado esquerdo, e na do lado direito pode ver os *templates* que estão disponíveis. Para adicionar *templates* à arquitectura, o utilizador arrasta os *templates* da lista dos disponíveis para a lista dos *templates* da arquitectura, e para retirar faz a operação inversa, i.e. arrasta os *templates* da lista dos *templates* pertencentes à arquitectura para a lista dos disponíveis.

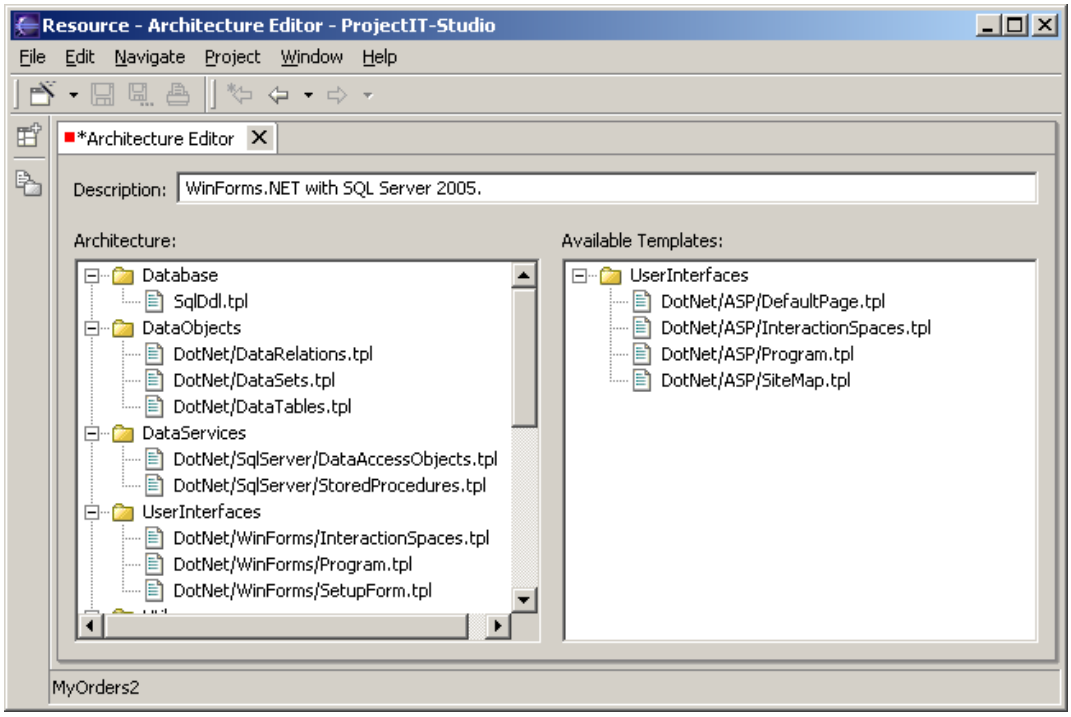


Figura 4.11 - Editor de arquitecturas de *software*.

4.3.2 Editor de Processos de Geração

O processo de geração é uma configuração do *input* do gerador, que associa o modelo à arquitectura de *software* da aplicação (ver Figura 4.12). O modelo é composto por subsistemas (tais como as entidades do modelo de domínio, ou as interface de utilizador da aplicação), e a arquitectura de *software* é composta por *templates* (tais como de geração de *scripts* SQL ou ficheiros de código C#).

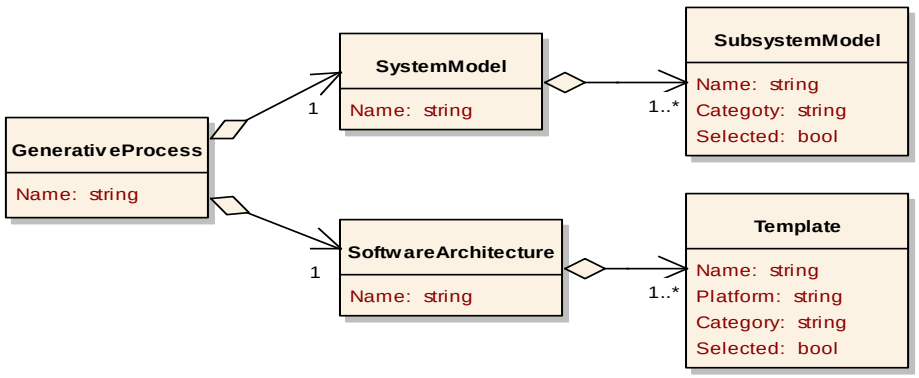


Figura 4.12 – Entidades persistentes do processo de geração.

No editor de processo de geração, o programador pode configurar o gerador através da selecção de uma arquitectura de *software* e de um modelo da aplicação. Também pode configurar filtros nas arquitecturas de *software* e nos modelos, para que apenas se gere o subconjunto da aplicação necessário, e.g. devido à correcção de um erro num *template* ou no modelo. Após ter o processo de geração bem configurado, o programador pode executar o processo de geração para gerar a aplicação. Estas funcionalidades estão representadas na Figura 4.13.

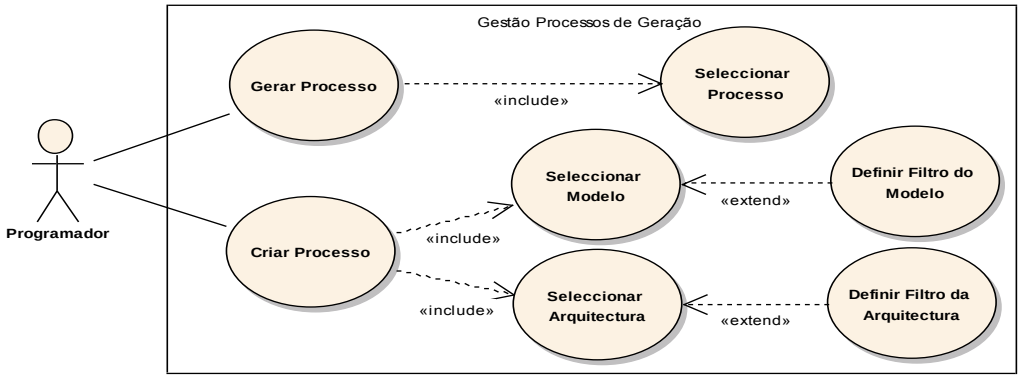


Figura 4.13 - Funcionalidades do gestor de processos de geração.

À semelhança do editor de arquitecturas, este também é desenvolvido com base na classe *"Editor"* da plataforma do Eclipse.NET, e a interface de utilizador é desenvolvida com as bibliotecas JFace e SWT.

A Figura 4.14 mostra o editor de processos de geração. A interface permite definir o nome do processo de geração, i.e. o nome da aplicação que é gerada; do lado esquerdo, permite escolher a arquitectura de *software* e seleccionar os *templates* que devem ser executados; do lado direito, permite escolher o modelo e seleccionar os subsistemas que são incluídos no modelo instanciado utilizado na geração. A geração é iniciada através da acção localizada na *toolbar* (assinalada na figura com a seta), e a monitorização dos erros de compilação ou execução dos *templates* são listados na janela *Tasks*, que pertence à plataforma do Eclipse.NET.

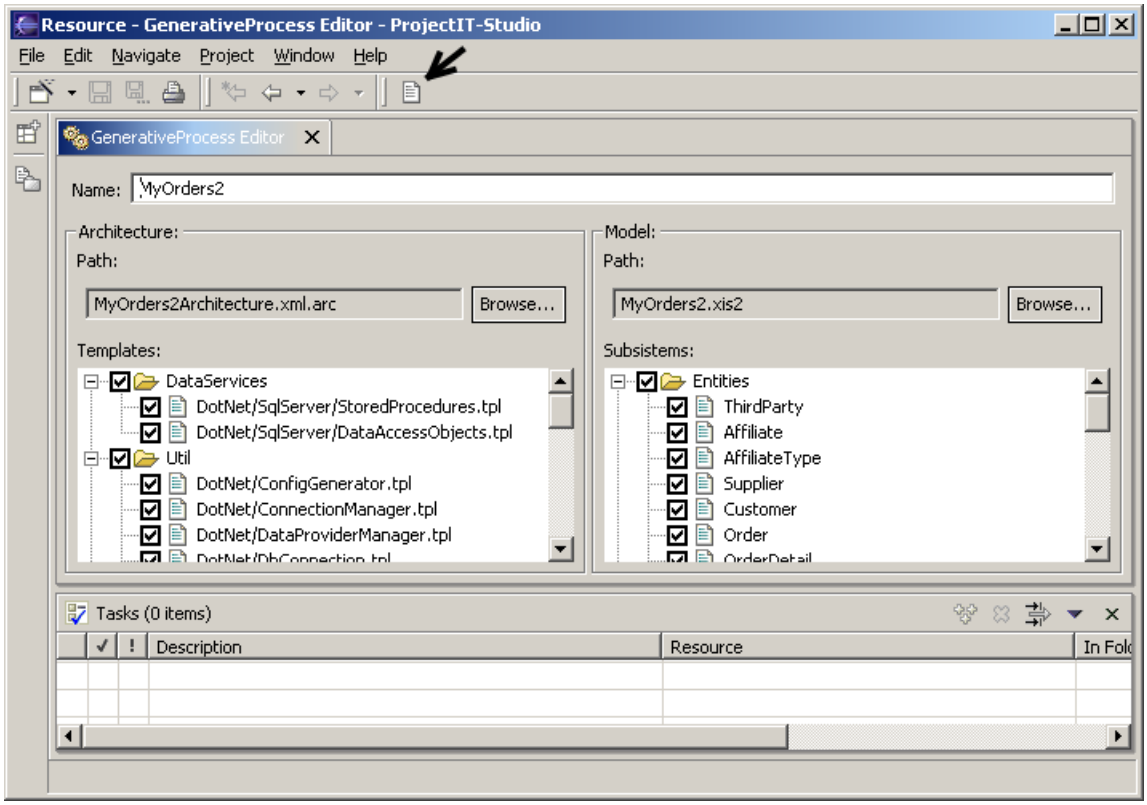


Figura 4.14 - Editor de processos de geração.

4.3.3 Editor de *Templates*

Os *templates* são definições de transformação modelo-para-código definidas pelo arquitecto em ficheiros de texto, na linguagem suportada pelo motor de *templates* (que está explicada no manual de utilizador no Anexo A). Portanto, este editor deve ter as funcionalidades básicas dos editores de texto, com funcionalidades extras, tais como *sintaxhighlight*, *autocomplete*, *folding*, de modo a aumentar a produtividade do processo de definição dos *templates*. Neste trabalho, implementou-se no editor (ver Figura 4.15) apenas as funcionalidades de *syntaxhighlight* e de sinalização de erros ocorridos durante a execução do processo de geração.

Os filtros também são definidos em ficheiros de texto, mas são escritos na linguagem C#. Desta forma, pode-se utilizar editores de código C#, e.g. o Visual Studio 2005, para definir os filtros.

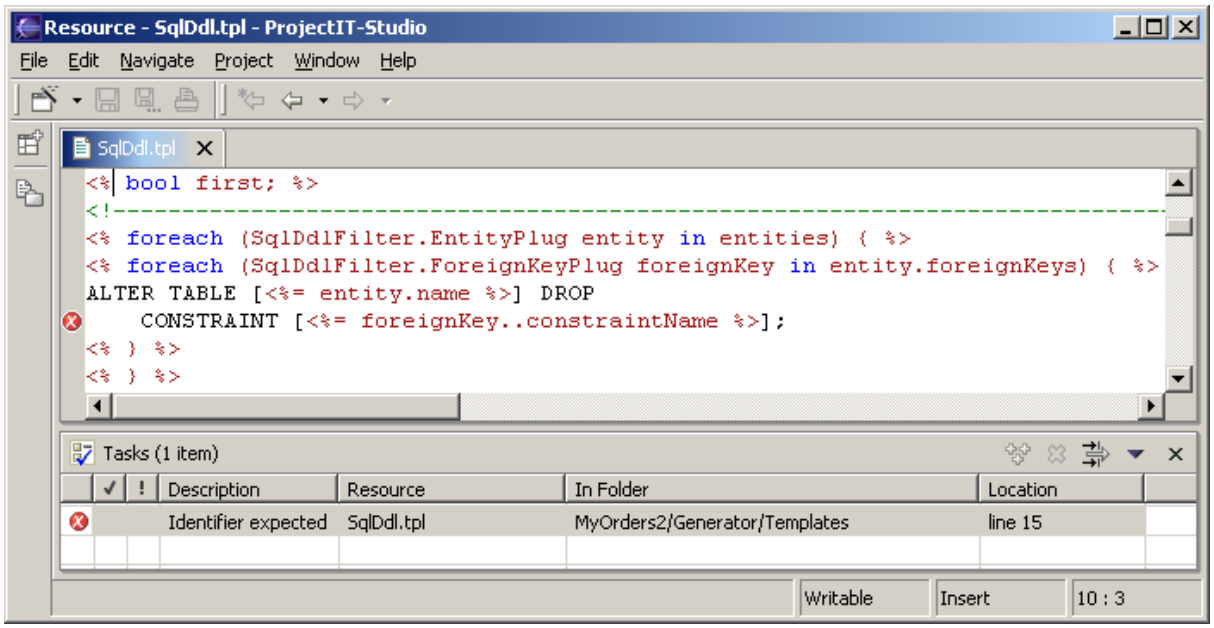


Figura 4.15 – Editor de *templates*.

4.3.4 Assistente de Conversão de Modelos UML2

O assistente de conversão de modelos UML2 (ver Figura 4.16) permite fazer a conversão do modelo da aplicação representado através do metamodelo UML2 da ferramenta de modelação, neste caso o do ProjectIT-Studio/UMLModeler, para o metamodelo da linguagem de modelação escolhida, e.g. o da linguagem XIS2. Este assistente é instalado pelo ProjectIT-Studio/MDDGenerator no ProjectIT-Studio/UMLModeler, através do mecanismo de comunicação entre *plugins* da plataforma Eclipse.NET.

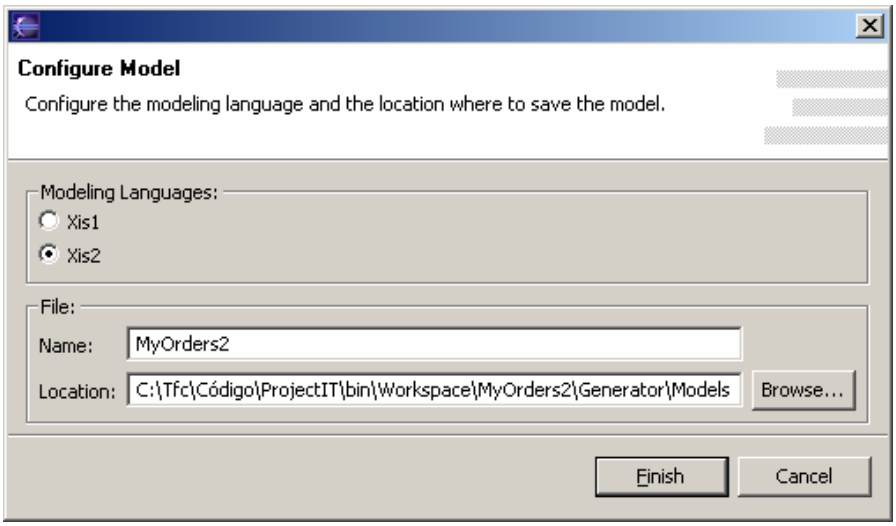


Figura 4.16 – Assistente de conversão de modelos UML2.

4.4 Generator.Core

O módulo Generator.Core é responsável por instanciar o modelo da aplicação e executar os *templates* da arquitectura de *software*. Para tal, recebe como entrada um processo de geração, conforme mostra a Figura 4.12, que tem um modelo da aplicação definido numa linguagem de modelação, e.g. a linguagem XIS2, e uma arquitectura de *software* da plataforma onde se pretende implementar a aplicação.

Com base na informação providenciada pelo processo de geração, o gerador – componente responsável pela geração – funciona como mostra a Figura 4.17: (1) instancia o modelo e filtra os subsistemas que não estão seleccionados; e (2) executa cada um dos *templates* seleccionados da arquitectura de *software*, que por sua vez dão origem aos vários artefactos da aplicação.

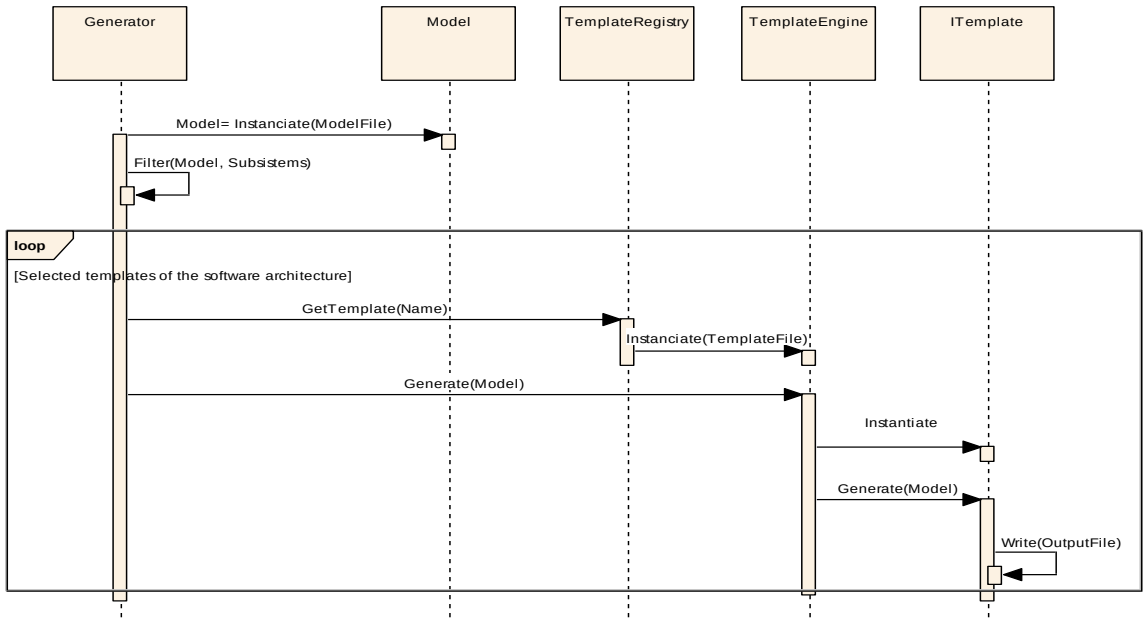


Figura 4.17 - Funcionamento da geração.

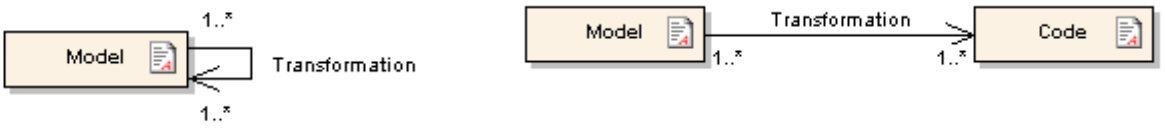
5 Transformações de Modelos

Um dos conceitos fundamentais das abordagens de desenvolvimento baseadas em modelos é a transformação de modelos de forma automática. A abordagem ProjectIT também se baseia neste conceito, pretende que os modelos definidos com a linguagem de modelação, e.g. a linguagem XIS2, sejam transformados automaticamente nos artefactos da aplicação.

Genericamente, as transformações recebem modelos e produzem modelos ou artefactos. As especificações das transformações consistem em *mapeamentos* dos elementos dos metamodelos de entrada para os dos metamodelos de saída. Quando executadas, as transformações recebem os modelos, e com base nos *mapeamentos* definidos, produzem os modelos ou os artefactos correspondentes.

5.1 Tipos de Transformações

As transformações dividem-se em dois tipos (ver Figura 5.1): (a) modelo-para-modelo, e (b) modelo-para-código. As transformações modelo-para-modelo recebem modelos e produzem outros modelos. As transformações modelo-para-código recebem modelos e produzem os artefactos para a plataforma de instalação da aplicação.



a) Modelo-para-Modelo.

b) Modelo-para-Código.

Figura 5.1 – Tipos de transformações.

É relevante questionar se as transformações modelo-para-código são diferentes das transformações de modelo-para-modelo. Do ponto de vista formal, o código também tem um modelo subjacente, portanto as transformações modelo-para-código são um caso particular das transformações modelo-para-modelo [17]. A diferença reside na aplicação prática deste conceito, pois numa transformação modelo-para-modelo, o metamodelo de origem e o metamodelo de destino estão bem definidos logo à partida, antes de se proceder à especificação da transformação. No entanto, numa transformação modelo-para-código o metamodelo de destino não é conhecido ou é ignorado propositadamente, pois uma especificação completa do metamodelo do código seria, nestes casos, um trabalho muito exaustivo e difícil. Deste modo, resulta que na prática as transformações modelo-para-modelo são especificadas de forma diferente das transformações modelo-para-código.

Normalmente, nas transformações modelo-para-modelo define-se o metamodelo de origem e o de destino, e depois define-se o *mapeamento* de um metamodelo para o outro. Assim, quando a transformação é executada o modelo de entrada é transformado no de saída.

5.2 Transformações Modelo-para-Código

As transformações modelo-para-código são definidas sem recurso aos metamodelos de saída. A forma mais simples é através da escrita de programas, onde a entrada é o modelo e a saída é o artefacto desejado. A definição da transformação consiste fundamentalmente numa sequência de instruções de escrita para o ficheiro de saída, com fluxo e conteúdo definidos a partir dos dados provenientes do modelo (ver Figura 5.2). Esta forma de definir transformações é semelhante à geração dinâmica das páginas HTML na Internet através de CGI (*Common Gateway Interface*).

```
foreach (XisEntity entity in model.entityModel.entityList) {
      Console.WriteLine("DROP TABLE [" + entity.name + "];");
}
```

Figura 5.2 – Exemplo de um excerto de programa que define uma transformação.

Contudo, uma evolução que surgiu naturalmente na Internet foi o aparecimento de JSP (*Java Server Pages*) e ASP (*Active Server Pages*) para a geração dinâmica das páginas HTML. Esta nova tecnologia utiliza a estrutura do artefacto de saída para a especificação da transformação (ver Figura 5.3), e depois converte-a num programa semelhante a um CGI. As transformações definidas através desta técnica designam-se por *templates* [10].

```
<% foreach (XisEntity entity in model.entityModel.entityList) { %>
DROP TABLE [<%= entity.name %>];
<% } %>
```

Figura 5.3 - Exemplo de um excerto de um *template*.

Os programas são razoavelmente bons a controlar o modelo de entrada da transformação, i.e. restringem e organizam os dados do modelo de entrada para ficarem de acordo com a especificação do modelo de saída. No entanto, são maus a definir a forma do artefacto criado pela transformação, porque as instruções de escrita das linguagens de programação baseiam-se no conceito de que a saída é um cadeia de caracteres, sendo o formato definido através de sucessivas instruções de escrita com auxilio a caracteres de escape, o que esconde o aspecto esperado do artefacto.

A motivação para a utilização de *templates* prende-se com a facilidade de se especificar a saída da transformação, pois os *templates* funcionam com base no conceito de substituição, pelo que a forma do artefacto é tida em conta na definição do *template* [14]. Contudo, numa abordagem puramente baseada em *templates*, as instruções de definição do aspecto do artefacto confundem-se com o controlo do modelo de entrada [12].

Existe no entanto o padrão *TWO-STEP-VIEW* [14], que permite conciliar estes dois mundos através de uma sequência de dois passos. No primeiro passo, controla-se a entrada, i.e. retiram-se os dados em excesso e organizam-se num novo modelo, que está adaptado à estrutura do artefacto da saída para facilitar a definição da transformação. No segundo passo, faz-se a definição da saída com base nos dados deste novo modelo. Este padrão permite utilizar um programa no primeiro passo e um *template* no segundo. As vantagens e desvantagens de se combinar o padrão *TWO-STEP-VIEW* com a tecnologia de *templates* estão na Tabela 5.1.

Vantagens	**Desvantagens**
• A transformação tem o aspecto mais parecido com o do artefacto. • Separação da lógica da transformação da definição do aspecto do artefacto. • Performance, porque o modelo é percorrido menos vezes.	• Sem suporte para programação orientada a objectos – a herança é substituída pela inclusão de *templates*.

Tabela 5.1 –*Templates* com o padrão *TWO-STEP-VIEW.*

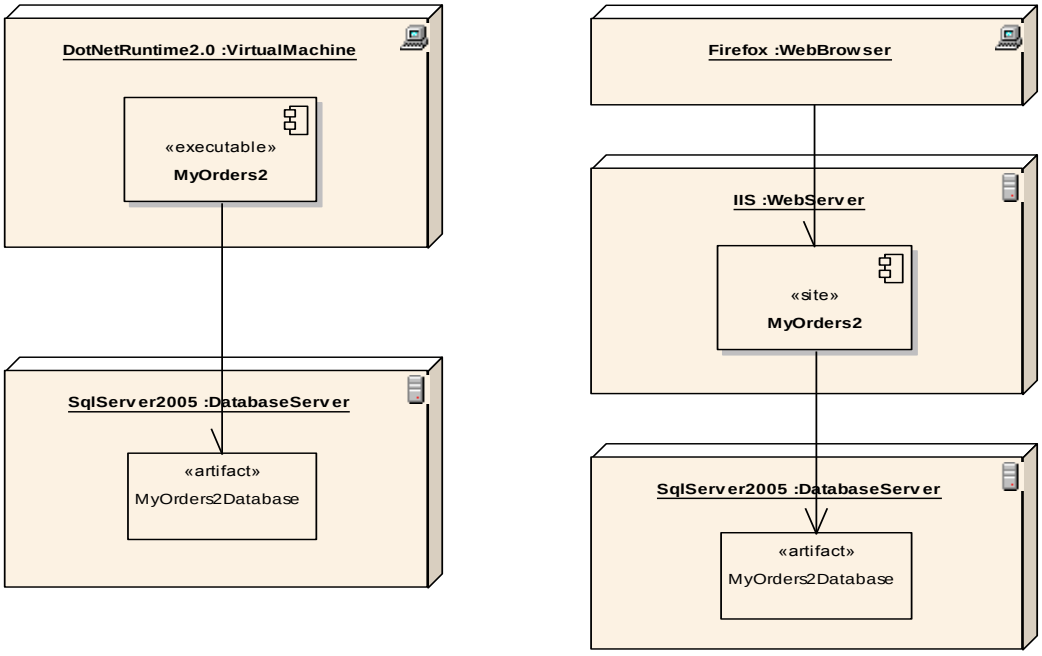
Neste trabalho, pretende-se suportar a definição das transformações modelo-para-código através de *templates*, com a possibilidade de aplicação do padrão *TWO-STEP-VIEW*. Desta forma, a linguagem para definir os *templates* deve: (1) ser *Turing-Complete*, i.e. deve poder executar qualquer tarefa computacional, e (2) permitir separar o processamento da entrada da definição da saída.

Para satisfazer estes requisitos, desenvolveu-se uma linguagem de definição de *templates* semelhante à dos ASPs, com suporte para expressões escritas em linguagem *C#*, o que satisfaz o requisito (1). Além disso, desenvolveu-se um conceito, a que se chamou de “filtro”, que permite pré-processar a entrada do *template* para simplificar o modelo utilizado na definição do mesmo, de forma a satisfazer o requisito (2). A explicação desta linguagem está no manual de utilizador (Anexo A).

5.3 Arquitecturas de *Software*

Na abordagem ProjectIT, as arquitecturas de *software* caracterizam as plataformas para as quais se podem gerar os artefactos das aplicações. Desta forma, são compostas por transformações modelo-para-código, organizadas de modo a que os artefactos gerados reflictam os elementos arquitecturais, que caracterizam a estrutura da aplicação gerada [18].

Neste trabalho, desenvolveram-se duas arquitecturas de *software*: a (1) *WinFroms.NET* e a (2) *ASP.NET*. A Figura 5.4 mostra estas arquitecturas de *software* para o caso de estudo *MyOrdes2*. A arquitectura de *software WinForms.NET* (ver Figura 5.4.a) tem uma arquitectura de instalação do tipo *TWO-TIERS* [18], um nível para a base de dados e outro para a aplicação com interfaces em *Windows Forms* da plataforma .NET. A arquitectura de *software ASP.NET* (ver Figura 5.4.b) tem uma arquitectura de instalação do tipo *THREE-TIERS* [18], com um nível para a base de dados, outro para o site com a lógica da aplicação, e outro para a apresentação através do navegador Internet.



a) *WinForms.NET*. b) *ASP.NET*.

Figura 5.4 - Arquitecturas de instalação.

Independentemente da arquitectura de instalação, a organização destas duas arquitecturas de *software* envolve três módulos, como mostra a Figura 5.5: (1) módulo com objectos que representam os dados do repositório; (2) módulo de acesso a dados, que gere os acessos à

base de dados; e (3) módulo de interfaces de utilizar para edição dos dados. O que difere entre a arquitectura de *software WinForms.NET* e a *ASP.NET* é o módulo das interfaces de utilizador.

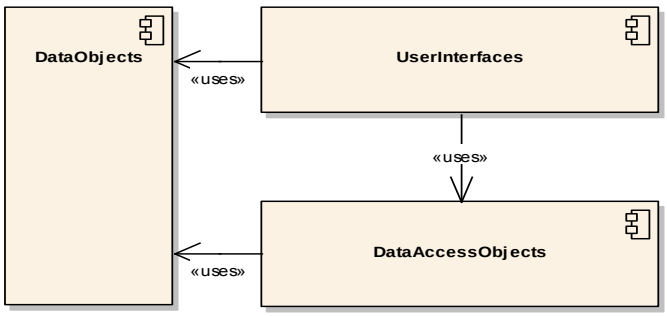
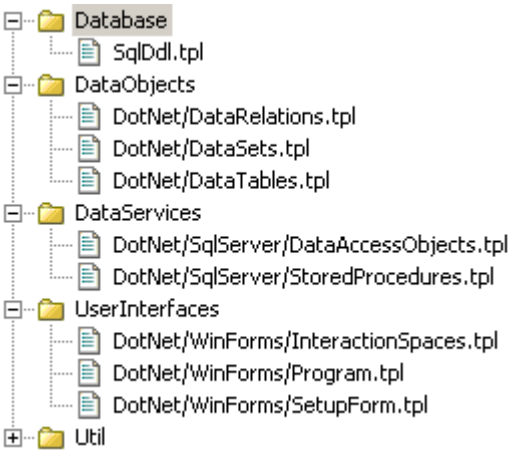
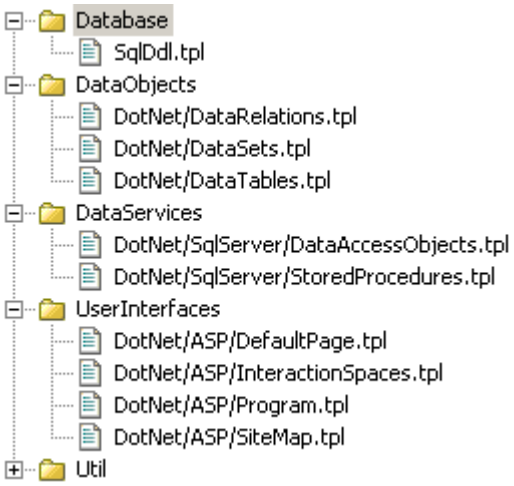


Figura 5.5 - Arquitectura de módulos.

A Figura 5.6 mostra o conjunto dos *templates* que constituem as arquitectura de *software WinForms.NET* e *ASP.NET*.



a) *Winforms.NET*.



b) *ASP.NET*.

Figura 5.6 – Arquitecturas de *software* para *Winforms.NET* e *ASP.NET*.

6 Conclusão

Neste trabalho, desenvolveu-se uma aplicação de *software* – o ProjectIT-Sudio/MDDGenerator – com o objectivo de suportar a actividade de geração automática de artefactos segundo a abordagem ProjectIT. Esta aplicação foi construída sobre a plataforma Eclipse.NET. Como forma de continuar a evolução do trabalho que tem vindo a ser desenvolvido no âmbito de três TFCs anteriores, desenvolveu-se também uma nova linguagem de modelação e uma nova forma de definição das transformações modelo-para-código.

Desenvolveu-se a linguagem de modelação XIS2 com o objectivo de permitir, entre outras coisas, definir as interfaces de utilizador com maior detalhe. Assim, alterou-se significativamente a abordagem de modelação das interfaces de utilizador, estas passaram a ser definidas com base no padrão *COMPOSITE*, i.e. através da composição dos vários elementos visuais, sejam estes botões, tabelas, caixas de texto, etc. Isto para substituir a abordagem anterior, que consistia apenas na associação de um controlador com uma entidade de negócio.

Evoluiu-se a forma de definição de transformações modelo-para-código, de uma abordagem baseada na escrita de "programas" para uma abordagem baseada na definição de *"templates"*. Isto envolveu a conversão de transformações modelo-para-código para a nova abordagem, a escolha foi determinada pela importância que cada transformação teria na nova linguagem XIS2. Assim, as transformações convertidas (ver Figura 6.1) estavam relacionadas com a camada de acesso a dados das aplicações geradas, isto porque os modelos de domínio de ambas as linguagens de modelação, XIS1 e XIS2, são bastante semelhantes. Além da conversão, também se implementou o suporte para a modelação das associações muitos-para-muitos nestes *templates*.

Figura 6.1 – Conjunto de transformações convertidas para *templates*.

Finalmente, validou-se este trabalho através do desenvolvimento de uma aplicação de teste, o *MyOrders2*, seguindo a abordagem ProjectIT e utilizando as ferramentas do ProjectIT-Studio. Para tal, foi necessário desenvolver os *templates* das interfaces de utilizador para as arquitecturas de *software* Winforms.NET e ASP.NET. Esta tarefa contou com a colaboração de outro elemento da equipa de desenvolvimento do ProjectIT, que ficou responsável pelo desenvolvimento dos *templates* que geram o código que dispõem os elementos da interface de utilizador nos ecrãs; enquanto que neste trabalho se desenvolveu os *templates* que geram o código com a lógica associada aos eventos desencadeados nas interfaces de utilizador, relacionados com o carregamento dos dados nas interfaces, com a actualização dos dados no repositório, e com a navegação entre os vários ecrãs.

6.1 Avaliação dos Resultados

No decorrer do desenvolvimento deste trabalho, detectaram-se alguns problemas que surgiram devido às soluções implementadas. Destacam-se os problemas detectados na linguagem de modelação, na definição das transformações modelo-para-código, na aplicação ProjectIT-Studio/MDDGenerator e no desenvolvimento da aplicação para validação do trabalho, o *MyOrder2*.

6.1.1 Linguagem de Modelação

Na linguagem XIS2, o padrão *COMPOSITE* dificulta a especificação da semântica da linguagem de modelação das interfaces de utilizador. Tal acontece, porque existem muitas combinações possíveis para a composição dos tipos de elementos de interacção. Desta forma, é necessário utilizar uma linguagem textual, e.g. OCL *(Object Constraint Language)*, para ajudar a definir completamente a linguagem de modelação referente às interfaces de utilizador. Neste trabalho, fez-se a especificação da semântica de algumas destas combinações (ver Anexo B), focou-se mais nas necessárias para modelar a aplicação *MyOrders2*.

Um último aspecto a considerar na linguagem de modelação XIS2 é o de não suportar associações n-árias. No entanto, este aspecto pode ser contornado através da inclusão de uma entidade no modelo que faça o papel da associação n-ária, com ligações binárias para cada uma das outras entidades participantes na associação.

6.1.2 Transformações Modelo-para-Código

As definições dos *templates* exigem alguma pré-configuração nos cabeçalhos dos *templates*, antes de se iniciar a definição da transformação propriamente dita, que pode ser

em alguns casos demasiado extensa e repetitiva, e.g. um *template* com sete argumentos tem de utilizar sete vezes a directiva *"Argument"* na pré-configuração, e a directiva *"Include"* com a referência para a biblioteca do metamodelo da linguagem de modelação é repetida em praticamente todos os *templates* que suportam esse metamodelo.

6.1.3 ProjectIT-Studio/MDDGenerator

O editor de *templates* desenvolvido neste trabalho é insuficiente para permitir a definição das transformações modelo-para-código de um modo produtivo. Isto acontece, porque o editor carece de algumas funcionalidades que aumentam a produtividade, tais como *autocomplete*, detecção de erros de sintaxe enquanto se escreve e formatação automática.

6.1.4 Desenvolvimento do *MyOrders2*

O modelo da aplicação *MyOrders2* desenvolveu-se segundo a versão *"dummy"* da abordagem ProjectIT, isto porque a versão *"smart"* ainda está em fase de desenvolvimento, no contexto de outros trabalhos. Tal como se esperava, a construção das interfaces através da versão *"dummy"* consumiu um esforço físico significativo, com algumas tarefas um quanto ou tanto repetitivas. Porém, as funcionalidades da aplicação corresponderam às expectativas criadas, pela análise das características da linguagem XIS2 e pelos *templates* desenvolvidos.

6.2 Trabalho Futuro

No âmbito da tese de mestrado, propõem-se o seguinte:

- Desenvolver uma componente na linguagem XIS2 para definir a lógica das operações de gestão específicas do negócio.
- Suportar nos *templates* os mecanismos de controlo de acesso e mais padrões das interfaces de utilizador.
- Incorporar no motor de *templates* um mecanismo que permita na definição dos *templates* especificar secções protegidas nos artefactos gerados, que são mantidas durante as várias execuções do *template* sobre o mesmo artefacto de saída.
- E finalmente, desenvolver mais funcionalidades no editor de *templates,* tais com *autocomplete*, detecção de erros de sintaxe enquanto se escreve e formatação automática, para aumentar a produtividade deste processo.

No âmbito de outros trabalhos relacionados com o ProjectIT, sugere-se o seguinte:

- Estudo de um mecanismo para criar modelos de domínio com base nos esquemas relacionais, para se estender a abordagem ProjectIT no sentido de suportar a evolução das aplicações legadas.
- Implementação no ProjectIT-Studio de um mecanismo de gestão de versões, para se suportar de forma integrada o trabalho em equipa.

Referências

[1] B. Selic, *"The Pragmatics of Model-Driven Development"*. IEEE Software, Vol. 5, No. 5, pp. 19-25, Setembro/Outubro 2003.

[2] A. Silva, *"O Programa de Investigação 'ProjectIT'"*, INESC-ID & Instituto Superior Técnico, Versão 1.0, Outubro de 2004.

[3] Object Management Group. *"MDA Guide"*, Versão 1.0.1, 2003.

[4] G. Lemos e T. Matias. *"Relatório de TFC Projecto XIS – Abordagem e Ferramenta de Desenvolvimento"*. Instituto Superior Técnico, Julho de 203.

[5] R. Queiroga. *"Relatório de TFC The XIS CASE Tool"*. Instituto Superior Técnico, Outubro de 2004.

[6] J. Saraiva. *"Relatório Final de TFC Desenvolvimento Automático de Sistemas"*. Instituto Superior Técnico, Setembro de 2005.

[7] A. Silva, *et al*., *"The ProjectIT-Studio, an integrated environment for the development of information systems"*, in *Proceedings of the Second International Conference of Innovative Views of .NET Technologies* (IVNET'06), (Florianópolis, Brazil), LNCS, Springer, d. p. [Outubro de 2006].

[8] A. Silva. *"Visão Geral do Projecto XIS"*. Grupo de Sistemas de Informação, LAVC/INESC-ID. Versão 1.2, Novembro de 2002.

[9] J. Saraiva e A. Silva. *"Eclipse.NET: An Integration Platform for ProjectIT-Studio"*. Em *Proceedings of the First International Conference of Innovative Views of .NET Technologies*. Instituto Superior Engenharia do Porto, Junho de 2005.

[10] M. Voelter. *"A Catalog of Patterns for Program Generation"*. Voelter, Versão 1.6, Abril de 2003.

[11] A. Silva, C. Videira. *"UML Metodologias e Ferramentas Case"*. Volume1, 2ª Edição, Centro Atlantico, 2005.

[12] T. Parr. *"Enforcing Strict Model-View Separation in Template Engines"*. University of San Francisco, Maio de 2004.

[13] D. Harel, B. Rumpe, *"Meaningful Modeling: What's the Semantics of 'Semantics'?"*, Computer 37:10 (October 2004), IEEE Press, 64-72 (cover feature).

[14] Fowler M., et al., *"Patterns of Enterprise Application Architecture"*, Addison Wesley, 2002.

[15] E. Gamma, et al., *"Design Patterns – Elements of Reusable Object-Oriented Software"*, Addison Wesley, 1994.

[16] A. Silva, J. Saraiva, R. Silva, C. Martins, *"XIS – UML Profile for eXtreme Modeling Interactive System"*, relatório técnico, INESC-ID, 2006. Disponível em apêndice.

[17] K. Czarnecki, S. Helsen, *"Classification of Model Transformation Approaches"*, *OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture*, 2003.

[18] P. Clements, et al., *"Documenting Software Architectures: Views and Beyond"*, Addison Wesley, 2002.

Referências para a Internet:

[w1] openArchitectureWare.org. *"openArchitectureWare"*. www.openarchitectureware.org, consultado em Setembro de 2006.

[w2] M. Voelter. *"oAW Introduction and Overview"*, http://www.eclipse.org/gmt/oaw/doc/4.0/05_IntroductionAndOverview.pdf, consultado em Outubro de 2006.

[w3] Compuware. *"OptimalJ"*. www.compuware.com/products/optimalj, consultado em Setembro de 2006.

[w4] INESC-ID, Grupo de Sistemas de Informação. *ProjectIT*. berlin.inesc.pt/alb/ProjectIT@81.aspx, consultado em Janeiro de 2006.

[w5] Object Management Group. *"Model Driven Architecture"*. www.omg.org/mda, consultado em Janeiro de 2006.

[w6] *"Object Management Group"*. www.omg.org, consultado em Janeiro de 2006

[w7] Microsoft. *"Visual Studio 2005 Team System Modeling Strategy and FAQ"*. Maio de 2005. msdn.microsoft.com/vstudio/DSLTools/default.aspx?pull=/library/en-us/dnvs05/html/vstsmodel.asp, consultado em Janeiro de 2006.

[w8] Microsoft. *".Net Framework 2.0"*. http://msdn2.microsoft.com/en-us/netframework/aa569294.aspx, consultado em Outubro de 2006.

[w9] Sun Microsystems, Inc. *"JavaServer Pages Technology"*. java.sun.com/products/jsp, consultado em Janeiro de 2006.

[w10] S. Sarstedt. *"TemplateMaschine – An open source template engine for C#"*. www.stefansarstedt.com/templatemaschine.html, consultado em Janeiro de 2006.

[w11] Object Management Group. *"MOF 2.0 / XMI Mapping Specification, v2.1"*. www.omg.org/technology/documents/formal/xmi.htm, consultado em Janeiro de 2006.

[w12] World Wide Web Consortium. *"XSL Transformations (XSLT)"*. Versão 1.0, Novembro de 1999. www.w3.org/TR/xslt, consultado em Janeiro de 2006.

[w13] Object Management Group. *"Unified Modeling Language"*. www.uml.org, consultado em Janeiro de 2006.

[w14] Microsoft. *"ASP.NET"*. asp.net, consultado em Janeiro de 2006.

[w15] World Wide Web Consortium. *"Extensible Markup Language (XML)"*. www.w3.org/TR/xslt, consultado em Janeiro de 2006.