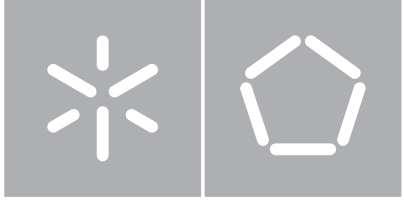
Universidade do Minho
Escola de Engenharia

Igor Gonçalo Gomes de Sá

Análise e Concepção de uma Framework de Reporting genérica e parametrizável

Outubro de 2012



Universidade do Minho
Escola de Engenharia

Igor Gonçalo Gomes de Sá

Análise e Concepção de uma Framework de Reporting genérica e parametrizável

Dissertação de Mestrado
Mestrado em Engenharia Informática

Trabalho realizado sob orientação de

Professor António Manuel Nestor Ribeiro

Outubro de 2012

Resumo

Actualmente as aplicações PRIMAVERA incluem componentes de reporting que exigem demasiado esforço de implementação e de manutenção, na medida em que todo o seu desenvolvimento é manual, repetitivo e assente em tecnologia desactualizada. Estes componentes de reporting são baseados nas soluções Crystal Reports, sendo necessária a construção/desenho em tempo de desenvolvimento de todos os relatórios que são pretendidos para um determinado produto. Cada um destes relatórios tem o seu desenho próprio, a suas próprias características e configurações, não existindo qualquer forma de partilhar determinadas propriedades que possam ser comuns aos vários relatórios. Por norma pretende-se que todos os relatórios de um produto tenham um aspecto uniforme, como por exemplo o layout ou fonte utilizada para determinados campos (por exemplo o título do relatório). Significa isto que é necessário na construção de cada um dos relatórios replicar todas estas características que são comuns, o que exige um esforço significativo e pode ser propício ao erro quando as regras de desenho de relatórios não estão bem definidas no início do desenvolvimento. Este problema torna-se mais evidente quando por exemplo num produto com um elevado número de relatórios se pretende fazer uma alteração numa destas características comuns. A simples alteração do tipo de fonte do título do relatório acaba por ser um processo bastante dispendioso, uma vez que é necessário editar todos os relatórios individualmente.

Esta dissertação surgiu da necessidade de desenvolver um novo componente de reporting que possa responder às limitações actuais. No âmbito do projecto PRIMAVERA ATHENA, está inserida a Framework de Reporting, cuja finalidade é dar suporte à criação, geração e apresentação de relatórios nos produtos desenvolvidos sobre a Framework ATHENA. Um dos principais objectivos da Framework de Reporting é a geração automática de relatórios a partir dos modelos das aplicações, acabando assim com todo o processo manual de criação de relatórios.

Abstract

Currently, PRIMAVERA applications include components for reporting that requires too much effort of implementation and maintenance, because the development is manual, repetitive and based on out dated technology. These components are based on Crystal Reports solutions, which require the construction/design at development time of all reports that are intended for a particular product. Each one of these reports has its own design, its own characteristics and settings, and there is no way to share certain properties that may be common to multiple reports. Usually it is intended that all reports of a product share a uniform appearance, such as the layout and font used for certain fields (for example the report title). This means that in the construction of each report is necessary to replicate all of these common characteristics, which requires a significant effort and may cause more errors if the design rules are not correctly defined in the beginning of the development. This problem becomes even more evident when in a product with a high number of reports is necessary to make a change in one of these common characteristics. The simple change of the report title font turns out to be a very expensive process, since it is necessary to individually edit all reports.

This work arose from a need to develop a new reporting component that can respond to the current limitations. Within the scope of PRIMAVERA ATHENA project, is the Reporting Framework, which aims to support the creation, generation and presentation of reports on products developed in the Athena Framework. One of the main objectives of the Reporting Framework is to provide automatic generation of reports from the applications model, ending with all the manual process in reports development.

Agradecimentos

E porque nunca teria conseguido realizar este trabalho sem ajuda e apoio de várias pessoas, aqui ficam expressos os meus agradecimentos:

Ao Professor António Manuel Nestor Ribeiro por ter aceitado orientar-me neste trabalho, e por toda a ajuda prestada durante todo o processo de desenvolvimento e escrita.

À PRIMAVERA Business Software Solutions, por me ter permitido desenvolver este trabalho no contexto do projecto ATHENA.

À equipa de desenvolvimento do projecto ATHENA, por toda a ajuda prestada no decorrer deste trabalho.

À minha namorada, pelo apoio e compreensão.

À minha família, sem a qual nunca teria sido possível chegar até aqui.

Conteúdo

Lista de figuras

Lista de tabelas

Capítulo 1

1 Introdução

1.1 Contexto

No mundo empresarial de hoje, só com elevados níveis de produtividade no processo de desenvolvimento de software é possível responder em tempo útil a um mercado cada vez mais alargado e exigente. Estas exigências podem passar por funcionalidades acrescidas no que diz respeito à integração de dados do negócio para melhor conhecer o estado das empresas, mas também pela possibilidade de acesso às aplicações a partir de qualquer lugar.

A PRIMAVERA Business Software Solutions dedica-se ao desenvolvimento e comercialização de soluções de gestão e plataformas para integração de processos empresariais num mercado global, disponibilizando soluções para as Pequenas, Médias, Grandes Organizações e Administração Pública (1). A PRIMAVERA pretende também manter o seu posicionamento como *early adopter* de tecnologia Microsoft, ganhando avanço para um posicionamento antecipado junto do mercado e dos seus clientes.

O projecto Athena surge neste contexto com o objectivo de conceber e produzir uma framework completa para suportar o desenvolvimento dos próximos produtos da empresa. Esta framework parte de princípios *Model-driven Development* (2) para estabelecer um paradigma complemente diferente para o desenvolvimento dos produtos PRIMAVERA. Um dos componentes principais do projecto Athena é a Framework de Reporting, cuja finalidade é dar suporte à criação, geração e apresentação de relatórios e informação de apoio à decisão nos produtos desenvolvidos sobre a Framework Athena.

Actualmente as aplicações PRIMAVERA incluem componentes de reporting e de exploração de dados baseados nas soluções Crystal Reports (3). A disponibilização desses componentes de reporting, bem como de todos os relatórios de cada aplicação, exige demasiado esforço de implementação e de manutenção, na medida em que todo o seu desenvolvimento é manual, repetitivo e assente em tecnologia desactualizada. A utilização desta tecnologia (*Crystal Reports*) obriga a que sejam criados em tempo de desenvolvimento todos os relatórios (“mapas”) que são pretendidos para um determinado produto. Isto é, com esta ferramenta é necessário “desenhar” todos os relatórios da aplicação, tendo estes de ser instalados em conjunto com a aplicação. Mesmo para relatórios que são semelhantes, quer no seu formato quer na forma de processar os dados (por exemplo: uma lista de clientes e uma lista de fornecedores), é necessário criar “mapas” diferentes para cada um, e uma vez que nestes “mapas” ficam todas as definições quer de dados e de apresentação (layout, fontes, cores, margens), significa que não existe qualquer noção de template na criação dos relatórios, sendo por isso necessário definir todas as propriedades de layout de forma independente para cada um dos relatórios. Isto tem um custo muito elevado na manutenção dos relatórios, pois mesmo quando se pretende efectuar uma simples alteração, por exemplo no estilo de uma fonte dos relatórios da aplicação, esta tem de ser executada em todos os “mapas”. Esta tecnologia, e abordagem, limitam bastante o utilizador do produto em termos de funcionalidades, pois este apenas tem disponíveis os relatórios que foram instalados com a

aplicação, ou então terá de construir os seus próprios “mapas” se pretender um relatório personalizado.
Como seria de esperar, todos os produtos possuem os “mapas” necessários para satisfazer a maioria das necessidades dos utilizadores, mas obriga a que num produto como o ERP PRIMAVERA, sejam construídos cerca de 2000 “mapas”.

Para a Framework Athena, pretende-se desenvolver uma solução completamente nova, que permita ultrapassar a grande maioria das limitações actuais, desde logo a necessidade de se construírem os relatórios manualmente, passando estes a ser gerados automaticamente em tempo de execução. Pretende-se que esta geração seja efectuada com base em templates, permitindo desta forma que apenas alterando o template, o relatório possa ser personalizado. É obrigatório também seleccionar uma nova tecnologia de Reporting sobre a qual serão desenvolvidos os diversos componentes necessários para a Framework de Reporting, abandonando-se assim o Crystal Reports.

1.2 Objectivos

O objectivo deste trabalho de dissertação, passa por desenvolver um componente de reporting que esteja por sua vez enquadrado com o desenvolvimento da Framework Athena. Pretende-se que este componente de reporting seja desenvolvido sob a Framework Athena, sendo nela integrado como um módulo independente. Isto é, este módulo é desenvolvido sob a tecnologia da Framework Athena, podendo ser instalado em qualquer produto que utilize a framework. Para a Framework de Reporting, estão definidos os seguintes objectivos principais:

- Enumeração e análise dos cenários de utilização da Framework de Reporting e dos seus requisitos. É necessário efectuar previamente o levantamento de requisitos para a Framework de Reporting, analisando também as implementações existentes actualmente deste componente nos produtos PRIMAVERA. Desta análise devem sair, para além das funcionalidades que são necessárias suportar, também as limitações existentes para melhor se perceber podem ser ultrapassadas.

- Melhoria das ferramentas de modelação da Framework Athena para suportar todos os cenários de reporting necessários para os produtos. Isto é, as ferramentas de modelação já existentes no Athena devem passar a suportar também a modelação de cenários de reporting.

- Avaliação e selecção dos componentes de terceiros que possam ser utilizados para o desenvolvimento do runtime da Framework de Reporting. Existe por isto a necessidade de fazer um levantamento das ferramentas existentes e que melhor satisfaçam os objectivos do Athena. Desta análise deve ficar claro qual o melhor componente a utilizar como base no desenvolvimento da framework de reporting.

- Desenvolvimento do motor de Reporting e sua integração com os restantes componentes da Framework Athena. Isto é, o desenvolvimento do motor de reporting é independente do Athena, sendo criado um módulo onde fica integrado o motor de *reporting*.

- Criação de todas as ferramentas de suporte necessárias para a gestão da framework de Reporting. Esta gestão é feita recorrendo ao desenvolvimento de um módulo a integrar na Framework Athena. Este módulo terá como objectivo suportar todas as

operações relacionadas com o componente de reporting. Neste módulo fica também integrado o motor de reporting.

1.3 Estrutura do documento

De seguida é apresentada a estrutura desta dissertação, onde é descrito resumidamente o conteúdo da cada capítulo.

No segundo capítulo é realizado o estudo de alguns conceitos úteis para o desenvolvimento desta dissertação: Frameworks, *Design Patterns* e *Model Driven Development*. É também feita uma breve introdução à Framework Athena, onde são descritos alguns dos seus objectivos, o que motivou o seu desenvolvimento e alguns dos princípios sob os quais assenta.

No terceiro capítulo é feito o levantamento de requisitos para a Framework de Reporting. Este capítulo começa por apresentar uma análise realizada ao componente de *reporting* utilizado actualmente no ERP PRIMAVERA, que tem como objectivo recolher informação acerca das necessidades em termos de funcionalidades e também das principais limitações existentes. É ainda feito o levantamento do estado actual de algumas ferramentas de *reporting* com o objectivo de seleccionar a que melhor se adequa aos objectivos definidos para a Framework de Reporting e que será utilizada como base no seu desenvolvimento.

No quarto capítulo são descritos os detalhes relativos à análise/desenho e implementação da Framework de Reporting. É apresentada a sua arquitectura, sendo descritos de forma detalhada cada um dos componentes desenvolvidos.

No quinto capítulo é apresentado o módulo de impressão onde vai ser integrada a Framework de Reporting. Neste capítulo é descrito de que forma as funcionalidades de impressão da Framework de Reporting são disponibilizadas aos produtos desenvolvidos utilizando a Framework Athena, mais concretamente o serviço de impressão. São ainda apresentados alguns exemplos de como o serviço de impressão pode ser utilizado para gerar os *reports*.

Por fim, no sexto capítulo, é realizado um resumo do trabalho desenvolvido, sendo também analisados os resultados conseguidos. São ainda apresentadas algumas ideias de trabalho futuro.

Capítulo 2

2 Desenvolvimento de Frameworks

Esta secção tem por objectivo fazer o levantamento de alguns conceitos úteis para o desenvolvimento deste trabalho: Frameworks, *Design Patterns* e *Model Driven Development*. Pretende-se também fazer uma breve introdução à Framework Athena, de forma a contextualizar onde vai ficar inserida a Framework de Reporting. São descritos alguns dos objectivos desta framework, o porquê da sua existência e alguns dos princípios sob os quais assenta.

2.1 Frameworks

Uma framework é um conjunto de classes que cooperam entre si, e que compõem uma solução reutilizável para um determinado tipo de software (4). É a reutilização do desenho de um sistema ou parte de um sistema expresso como um conjunto de classes abstractas e a forma como as instâncias dessas classes (subclasses das classes abstractas) colaboram entre si. As frameworks são uma forma particular de representar arquitecturas, por isso existem arquitecturas que não podem ser expressas como frameworks (5). Tipicamente são utilizados na implementação de componentes de maior dimensão, e são implementados recorrendo a simples classes (6).

A utilização de frameworks tem como grande vantagem a sua reutilização em diferentes aplicações, sem a necessidade de se reescrever todo o código que é específico da framework. Isto é, em vez de se desenvolver uma aplicação onde se implementa concretamente na própria aplicação tudo que é necessário, certos componentes podem ser desenvolvidos de forma independente e utilizados depois no contexto de várias aplicações. No caso concreto da framework de reporting, como foi já referido, o objectivo principal é que seja integrada na Framework Athena, mas sendo desenvolvida como uma framework independente, pode facilmente ser utilizada noutros contextos.

As frameworks ajudam a fornecer soluções para problemas de um determinado domínio e uma forma mais eficaz de manter essas soluções. Fornecem uma infra-estrutura bem desenhada e pensada para que quando novas peças são criadas, elas possam ser substituídas com um impacto mínimo nas restantes (7).

Uma framework não pode ser vista como uma simples biblioteca de classes. Embora existam diferenças claras entre estas, algumas bibliotecas apresentam um comportamento que pode ser considerado semelhante ao de uma framework, e algumas frameworks podem ser utilizadas simplesmente como uma biblioteca de classes (7). As principais características que distinguem uma framework de uma biblioteca de classes são:

- Controla o fluxo de execução
- Fornece um comportamento por omissão
- Fornece customização através da criação de subclasses
- Define a interacção entre objectos
- Executa funções do cliente

As frameworks definem aplicações “semi-completas” que encapsulam estruturas de objectos e funcionalidades de domínio específico (8) (9), enquanto uma biblioteca de classes fornece uma granularidade de reutilização relativamente pequena (8). Do ponto de vista de quem utiliza cada uma destas, a grande diferença é que no caso das bibliotecas de classes apenas é necessário conhecer a sua *interface* externa, sendo que na utilização de frameworks é necessário conhecer também a estrutura interna das classes que se pretende usar como base através de herança (10).

Uma framework pode ser classificada de acordo com a sua estrutura. Na Tabela 2.1 estão descritas essas classificações.

Classificação	Descrição
Manager-driven (orientadas ao *manager*)	Uma única função controla e inicia a maior parte das acções da framework, que é responsável por criar todos os objectos necessários e executar os passos correctos para uma determinada tarefa (7).
Architecture-driven (orientadas à arquitectura)	Baseiam-se na herança para a customização, criando novas classes que têm como base classes da framework reescrevendo o comportamento de determinadas funções (7).
Data-driven (orientadas aos dados)	Baseiam-se na composição de objectos para customização. O comportamento da framework depende do *input* que lhe é fornecido pelo cliente (7).

Tabela 2.1 - Classificação de uma framework

É difícil definir que determinado tipo framework é melhor ou pior que outro. Por exemplo, frameworks que são fortemente orientadas à arquitectura podem ser difíceis de utilizar, pois pode ser necessário escrever bastante código para que esta consiga produzir alguma coisa. Por outro lado, uma framework orientada aos dados é por norma mais fácil de utilizar, mas pode ser mais limitada no que diz respeito à customização. É importante encontrar um ponto de equilíbrio entre cada uma, tendo sempre em conta o tipo de framework que se está a desenvolver. Uma abordagem que pode ser seguida para construir frameworks fáceis de utilizar e de configurar, passa por desenvolver uma framework orientada à arquitectura com uma camada orientada aos dados (7).

A utilização de frameworks tem várias vantagens, principalmente ao nível da sua reutilização. Fazendo uma análise ao custo/benefício, pode afirmar-se que a sua utilização resulta principalmente em poupanças a longo termo. Isto porque apesar do esforço de desenvolvimento/aprendizagem serem mais elevados, a sua utilização permite a quem utiliza focar-se apenas nas funcionalidades da aplicação que está a desenvolver. Reduz também o custo de manutenção das aplicações que as utilizam, uma vez que estas têm menos código que possa causar problemas. E quando são encontrados problemas ao nível da própria framework, a sua correcção propaga-se pelas aplicações cliente (7).

2.2 Framework Athena

A indústria de desenvolvimento de software é ainda relativamente jovem e o processo de desenvolvimento tem uma grande componente de trabalho manual. Tal como no passado, noutras indústrias, a tendência é a automatização e a reutilização de componentes, o que dará origem a grandes economias de escala. Prevê-se que uma forma de atingir este objectivo é

através da utilização de linguagens específicas de domínio (11), em oposição às ferramentas clássicas tais como o UML, onde a proximidade da realidade ajude a reduzir a complexidade. Nessa proximidade o Analista trabalha a um nível de abstracção mais distante da linguagem dos computadores utilizando conceitos mais próximos do domínio do problema e onde é possível potenciar a informação presente no modelo para gerar artefactos a utilizar nos produtos.
Com o projecto Athena, a PRIMAVERA pretende desenvolver uma framework capaz de responder a estes pressupostos. Esta framework é desenvolvida partindo de princípios de *model-driven development*, baseada nas tecnologias Microsoft .NET e em linguagens de domínio específico.

A Framework Athena, tem em primeiro lugar, o objectivo de transformar o desenvolvimento dos produtos PRIMAVERA num processo *model-driven*, e atingir também um elevado nível de código gerado automaticamente a partir de modelos.

Esta Framework tem os seguintes objectivos:

- Todas as funcionalidades de negócio desenvolvidas devem ser analisadas segundo uma perspectiva de modelação, aplicando sempre que possível padrões de negócio e transformando os conceitos em partes do meta-modelo da framework.
- A geração de código e a reutilização são os principais objectivos da Framework Athena.
- O código de *user interface* deve ser 100% gerado.
- A percentagem de código gerado na implementação de serviços e lógica de negócio, deve ser bastante elevada.

Na figura Fig 2.1 está representada de forma lógica a Framework de Desenvolvimento Athena**:**

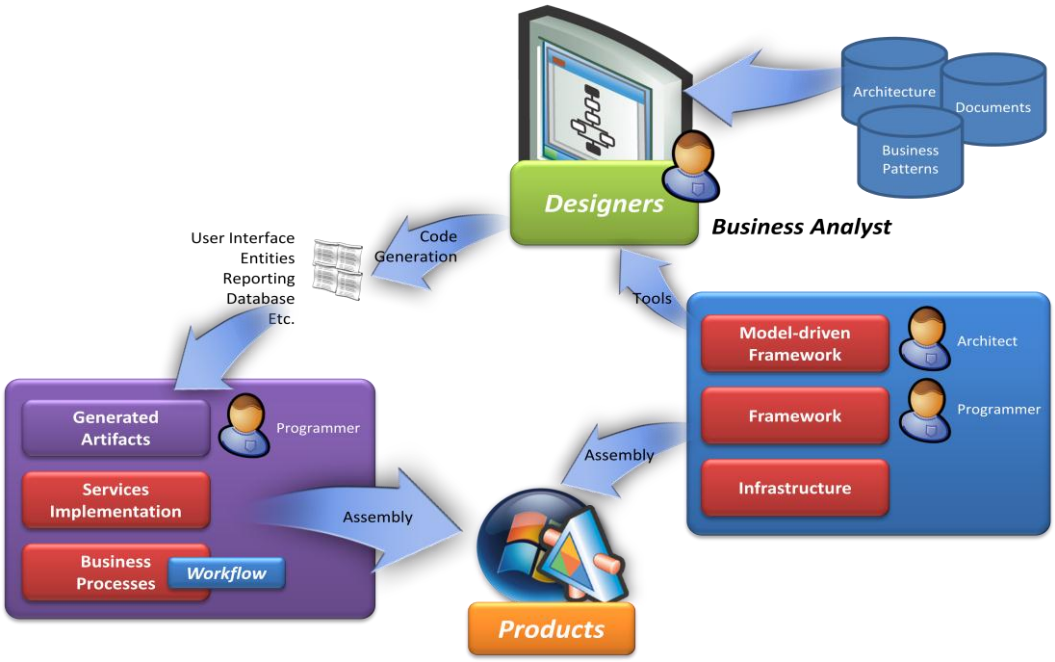


Fig 2.1 - Framework de desenvolvimento Athena

Esta framework fornece um conjunto de ferramentas (***Designers***) que permitem ao analista (***Business Analyst***) modelar o produto, isto é, utilizando as ferramentas de modelação do Athena o analista define as entidades (por exemplo: Clientes, Fornecedores, Artigos, Facturas) e os serviços e listas de exploração disponíveis para cada uma delas. A partir desta modelação do produto a framework é responsável pela geração de todo código necessário para implementar os serviços de CRUD bem como pela criação das respectivas tabelas na base de dados para cada uma das entidades definidas. São também gerados automaticamente a partir destes modelos os respectivos formulários (*user interface*) e as listas de exploração de dados. Numa fase seguinte, cabe ao programador implementar a lógica de negócio específica ou outro tipo de serviços necessários a cada uma das entidades e que não é possível ser gerada pela framework. O produto resulta então de código gerado a partir de modelos desenhados pelo analista, de código específico implementado manualmente pelo programador devido às necessidades específicas de negócio do produto e ainda de componentes disponibilizados pela framework.

No desenvolvimento de novos produtos/módulos, a framework abrange todo o processo de engenharia, desde os requisitos até ao produto final, sendo composta logicamente por dois conjuntos de funcionalidades como mostra a figura Fig 2.2 (12):

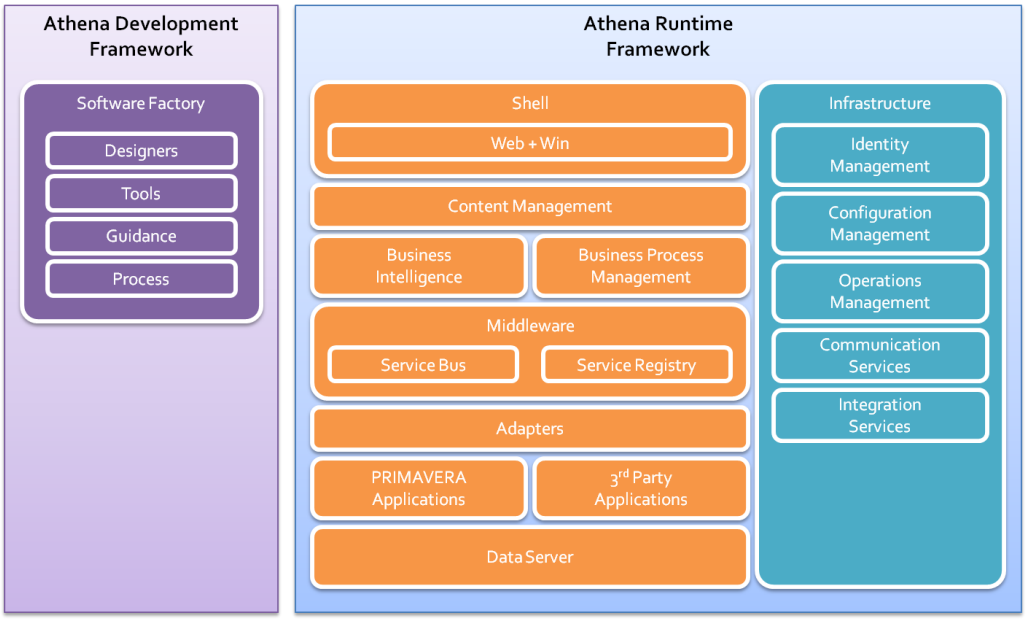


Fig 2.2 - Arquitectura Athena: principais componentes lógicos

A "*Development Framework*" inclui as metodologias, princípios e ferramentas necessárias para usar a Framework Athena no desenvolvimento de produtos. A "*Runtime Framework*" é composta pelos componentes que esses produtos utilizam (disponibilizam) para suportar as suas próprias funcionalidades (12).

Para a Framework de Reporting será construído um módulo usando a Framework Athena, ficando então este componente disponível aos produtos desenvolvidos sob o Athena através deste módulo (módulo de impressão).

2.3 Model Driven Development

Um dos problemas no desenvolvimento de software, é pensar na arquitectura da aplicação tendo em consideração qual o hardware e quais os outros softwares/tecnologias da qual vai estar dependente. Com o passar do tempo, vão existindo mudanças no hardware ou nas tecnologias das quais a aplicação está dependente, o que pode levar a que sejam feitas alterações na aplicação para que continue a funcionar, tendo estas alterações custos associados. É neste contexto que surge o Model-driven Development (MDD), que tem por objectivo modelar uma aplicação de forma independente da plataforma ou tecnologia. O MDD fornece um elevado nível de abstracção, permitindo assim ignorar os detalhes irrelevantes e focar-se apenas nos mais relevantes.

O MDD é um estilo de programação onde os principais artefactos de software são modelos, a partir dos quais é gerado código e outros artefactos (2). É o conceito de construir um modelo de um sistema que pode ser transformado num real sistema funcional (13). A característica que define o MDD é que o foco principal no desenvolvimento de software são precisamente os modelos e não os programas (14).

Podemos então afirmar que o propósito do MDD é melhorar a qualidade do software, reduzindo a sua complexidade, e aumentando o nível de reutilização dos componentes, permitindo a quem desenvolve, trabalhar num nível de abstracção mais elevado. Assim, o MDD tem então um grande potencial de reduzir os custos no desenvolvimento de soluções de software, e aumentar significativamente a sua qualidade. Isto é possível através da automatização de padrões de transformação, o que elimina o desenvolvimento repetitivo de trabalho de baixo valor acrescentado. Então em vez de se aplicar os conhecimentos técnicos no desenvolvimento repetitivo e manual de artefactos da solução, estes são aplicados directamente nas transformações. Isto traz vantagens como a consistência e a manutenção das soluções. Uma modificação numa transformação é rapidamente replicada para gerar artefactos da solução que reflectem uma alteração na arquitectura de implementação (2).

Algumas das vantagens mais importantes são:

- ***Aumento da produtividade**:* reduz os custos no desenvolvimento, gerando código e artefactos a partir de modelos.
- ***Manutenção**:* o progresso tecnológico leva a que componentes fiquem obsoletos. MDD permite ultrapassar este problema de forma mais fácil, pois alterações podem ser efectuadas de forma mais rápida e consistente.
- ***Adaptabilidade**:* adicionar ou modificar funcionalidades de negócio torna-se bastante simples, uma vez que o investimento na automatização já foi feito. Então quando se adiciona uma nova funcionalidade de negócio, apenas é necessário desenvolver o comportamento específico dessa funcionalidade.
- ***Consistência**:* MDD assegura que os artefactos são gerados de forma consistente.
- ***Independente da plataforma ou tecnologia:*** MDD fornece um elevado nível de abstracção, permitindo começar o desenvolvimento sem se ter ainda definido a plataforma ou tecnologia a utilizar.

Como seria de esperar, o MDD possui também as suas desvantagens. Elevar o nível de abstracção pode por vezes levar a uma simplificação excessiva, onde não existem detalhes

suficientes para retirar alguma informação útil. O princípio central do MDD é de que existem múltiplas representações dos artefactos inerentes a um processo de desenvolvimento de software, que representam diferentes pontos de vista ou níveis de abstracção sobre os mesmos conceitos. Uma vez que estas são manualmente criadas, é necessário duplicar o trabalho e gerir a consistência (15).

Os padrões desempenham um papel chave no MDD. Um padrão descreve uma solução para um problema recorrente no desenvolvimento de software (16).
MDD revela o potencial dos padrões para criar soluções bem desenhadas, e os padrões fornecem o conteúdo para o MDD (2).

O aumento substancial de produtividade é, uma das principais vantagens do MDD, conseguindo-se isso através da geração de código e artefactos a partir dos modelos. A automatização é a principal característica que distingue o MDD de outras técnicas que utilizam modelação (2).

A automatização é possível usando duas técnicas:

- Transformação: automatiza a geração de artefactos a partir de modelos, o que inclui a geração de código e também a geração de modelos mais detalhados (2).
- Padrões: automatizam a criação e modificação de elementos do modelo dentro do modelo para aplicar um determinado padrão de software. Os padrões podem ocorrer a qualquer nível de abstracção, por exemplo, padrões de arquitectura, padrões de design, e padrões de implementação (2).

Como já vimos, o MDD apresenta várias vantagens ao permitir a quem desenvolve trabalhar a um nível de abstracção elevado, tendo por isso um grande impacto no desenvolvimento de software, reduzindo a sua complexidade e aumentando a sua qualidade.

2.4 Design Patterns

Um padrão descreve uma solução genérica e reutilizável para um determinado problema que ocorre frequentemente no desenvolvimento de software (4).

Os padrões capturam o objectivo numa arquitectura identificando os objectos, como é que os objectos interagem entre si, e como são distribuídas as responsabilidades entre eles. A reutilização de padrões comuns abre novas perspectivas no conceito da reutilização de software, onde as implementações variam, mas as microarquitecturas representadas pelos padrões se continuam a aplicar.

Ao contrário das frameworks, os padrões não são implementações concretas. São sim uma solução para um determinado problema que terá depois de ser implementada sempre que se pretender utilizar no contexto do desenvolvimento de uma qualquer aplicação. Os padrões são por isso mais abstractos que as frameworks. São elementos arquitecturais mais pequenos que as frameworks. Uma framework pode conter vários padrões, mas o contrário nunca se verifica (4).

Os padrões ganharam um especial enfâse depois do lançamento do livro *Design Patterns: Elements of Reusable Object-Oriented Software* (4), escrito por Erich Gamma, Richard Helm, Ralph Johnson, e John Vlissides. Neste livro, os autores apresentam os padrões divididos segundo dois critérios:

- **Objectivo:** reflecte o que o padrão faz
    - **Criação:** dizem respeito ao processo de criação dos objectos
    - **Estrutural:** lidam com a composição de classes ou objectos
    - **Comportamental:** caracterizam as formas como classes ou objectos interagem e distribuem responsabilidades
- **Contexto:** especifica se o padrão se aplica principalmente às classes ou aos objectos
    - **Classe:** lidam com as relações entre classes e as suas subclasses
    - **Objecto:** responsáveis pelas relações entre objectos

A Tabela 2.2 apresenta os padrões que foram indicados no livro referido anteriormente:

		Objectivo		
		Criação	**Estrutural**	**Comportamental**
Contexto	**Classe**	Factory Method	Adapter	Interpreter
				Template Method
	Objecto	Abstract Factory	Adapter	Chain of Responsibility
		Builder	Bridge	Command
		Prototype	Composite	Iterator
		Singleton	Decorator	Mediator
			Facade	Memento
			Flyweight	Observer
			Proxy	State
				Strategy
				Visitor

Tabela 2.2 - Design Patterns

Um dos objectivos do livro era o de criar um catálogo de padrões, onde cada padrão está identificado por um nome, qual o problema que pretende resolver e seu contexto, a respectiva solução, e as consequências na sua aplicação.

Alguns destes padrões são especialmente úteis no desenvolvimento de frameworks. Nas secções seguintes são apresentados alguns desses padrões.

2.4.1 Singleton

Este padrão tem como objectivo garantir que apenas existe uma instância de uma determinada classe, e fornecer um ponto global de acesso (4).

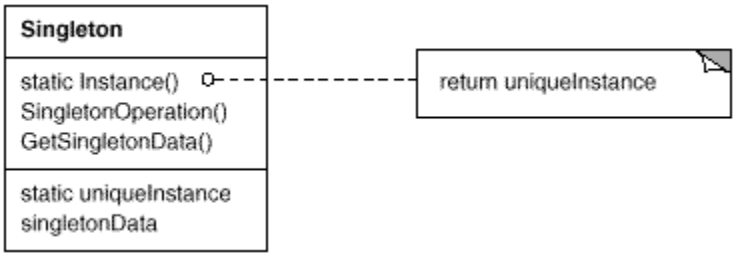


Fig 2.3 - Padrões: Singleton

A implementação deste padrão tem de garantir que apenas existe uma instância da classe, sendo isto controlado pela própria classe. Esta mantém a instância numa variável estática, ficando disponível através de um método estático, não sendo possível criar novas instâncias a partir de código exterior à própria classe. Este padrão é precisamente útil quando se pretende que uma classe tenha apenas uma instância, e que esta fique acessível globalmente na aplicação.

2.4.2 Abstract factory

Tem como objectivo fornecer uma interface para a criação de famílias de objectos relacionados ou dependentes sem a necessidade de especificar as suas classes concretas (4).

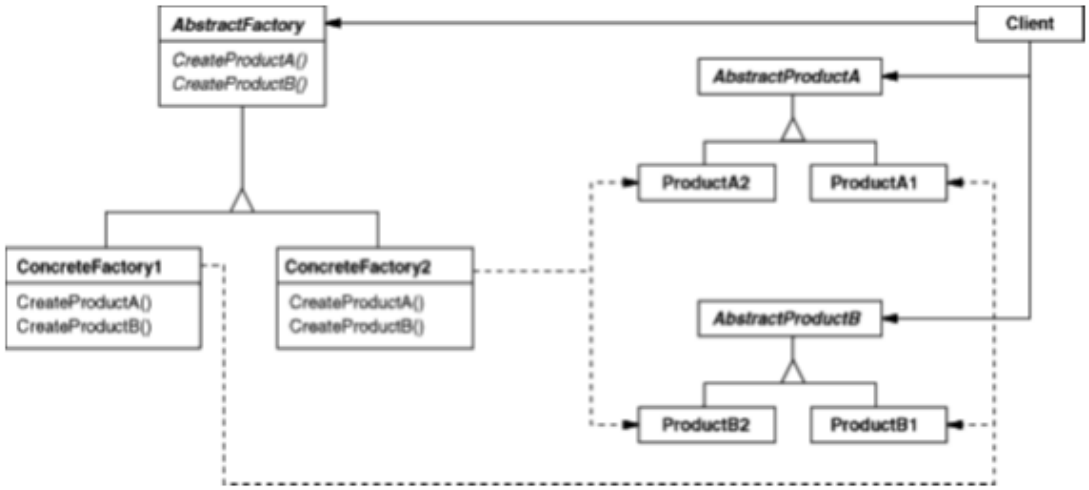


Fig 2.4 - Padrões: Abstract factory

A ***AbstractFactory*** declara uma interface para operações que são responsáveis pela criação dos objectos de produtos abstractos. A implementação concreta desta interface é feita pelo ***ConcreteFactory***, isto é, esta classe é responsável pela criação dos objectos de produtos concretos.
A interface para os objectos do tipo dos produtos é declara pelo ***AbstractProduct***, sendo esta implementada pelos ***ConcreteProduts***. Cada um dos ***ConcreteProducts*** é criado pela respectiva ***ConcreteFactory***.
Por fim, o ***Client*** referência apenas as interfaces declaradas pela ***AbstractFactory*** e pelo ***AbstractProduct***. Isto significa uma redução substancial das referências para classes concretas, e consequentemente a redução da quantidade de código que é necessário alterar quando as classes concretas mudam (17).

Este padrão deve ser utilizado quando se pretende que um sistema seja independente de como os seus produtos são criados, compostos e representados.

2.4.3 Factory Method

Define uma interface para criar um objecto, mas permite que sejam as subclasses a decidir que classe instanciar. Permite que uma classe delegue a responsabilidade de instanciação às subclasses (4).

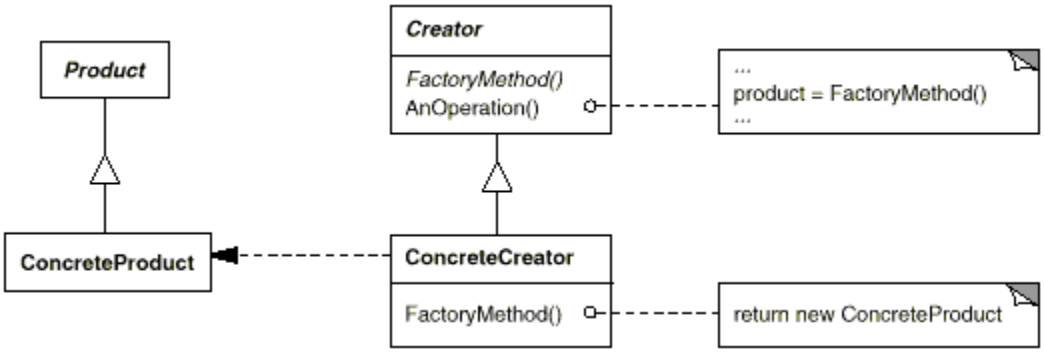


Fig 2.5 - Padões: Factory Method

O ***Product*** define a interface dos objectos que são criados pelo *factory method*, sendo esta implementada pelo ***ConcreteProduct***. A classe ***Creator*** declara o *factory method*, que retorna objectos do tipo ***Product***. O ***ConcreteCreator*** redefine o *factory method* para retornar objectos do tipo ***ConcreteProduct*** (4). Neste caso, a classe ***Creator*** define a interface para criar um objecto, mas é a subclasse ***ConcreteCreator*** que efectivamente decide qual o objecto vai instanciar.

Este padrão é útil quando uma classe não consegue antecipar que objectos têm de criar e também quando pretende delegar essa tarefa para as subclasses (4).

2.4.4 Facade

Tem como objectivo fornecer uma interface única para um conjunto de interfaces num subsistema. Define uma interface num nível superior que torna o subsistema mais simples de utilizar (4).

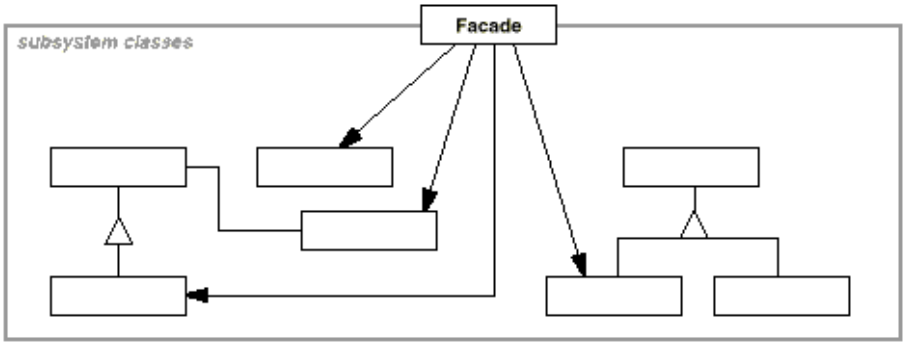


Fig 2.6 - Padrões: Facade

Um subsistema mais complexo passa a estar disponível através de um ***Facade***, sendo este o ponto único pelo qual este subsistema pode ser acedido, tornando a sua utilização mais simples e fácil de perceber por parte de quem o utiliza. Este padrão reduz a dependência entre o cliente e as existentes no subsistema fornecendo um ponto de entrada único.

2.4.5 Template Method

Define o esqueleto de um algoritmo numa operação, delegando alguns passos às subclasses (4). Permite por isso que estas subclasses possam redefinir determinados passos do algoritmo sem alterar a sua estrutura.

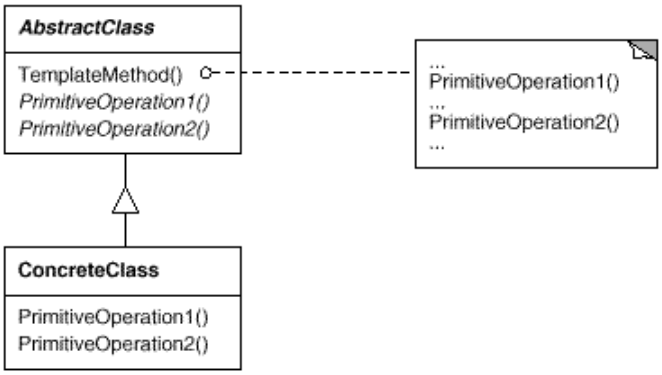


Fig 2.7 - Padrões: Template Method

A ***AbstractClass*** define as operações primitivas abstractas que fazem parte de um algoritmo, onde a sua implementação concreta é feita pelas subclasses (***ConcretClass***). O ***TemplateMethod*** definido na ***AbstractClass*** executa os passos do algoritmo.

Este padrão deve ser utilizado quando se pretende implementar as partes que não variam de um algoritmo num único local, e deixar para as subclasses a implementação do comportamento que varia (4).

Capítulo 3

3 Análise de Requisitos

Este capítulo começa por apresentar uma análise feita à componente de reporting da versão actual do ERP PRIMAVERA. Esta primeira análise serve essencialmente para identificar as necessidades existentes em termos de reporting, quais as funcionalidades que tem de ser suportadas na nova plataforma e também quais os problemas existentes para que estes possam ser ultrapassados.

No seguimento desta análise, e também com base nela, são levantados os requisitos da Framework de Reporting.

3.1 Análise do componente de reporting do ERP PRIMAVERA

Neste capítulo será analisada solução de reporting que está actualmente implementada no ERP PRIMAVERA. Esta análise tem por objectivo recolher informação acerca das necessidades em termos de funcionalidades existentes, bem como dos problemas existentes e que devem ser ultrapassados pela nova Framework de reporting.

Em primeiro lugar é necessário definir o que é um relatório:

"*Relatórios são documentos que apresentam conteúdo específico, focalizado – frequentemente o resultado de uma experiência, investigação ou inquérito – a uma audiência específica.*" (18)

Para a PRIMAVERA, um relatório pode ser definido como um conjunto de informação que é necessária dar ao utilizador, como por exemplo uma factura, um balancete ou uma simples lista de clientes.
Conceptualmente um relatório pode ser definido da forma apresentada na Fig 3.1:

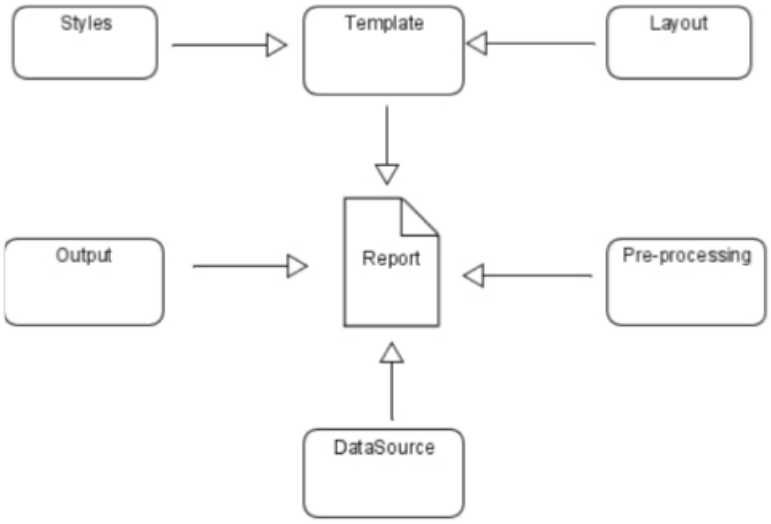


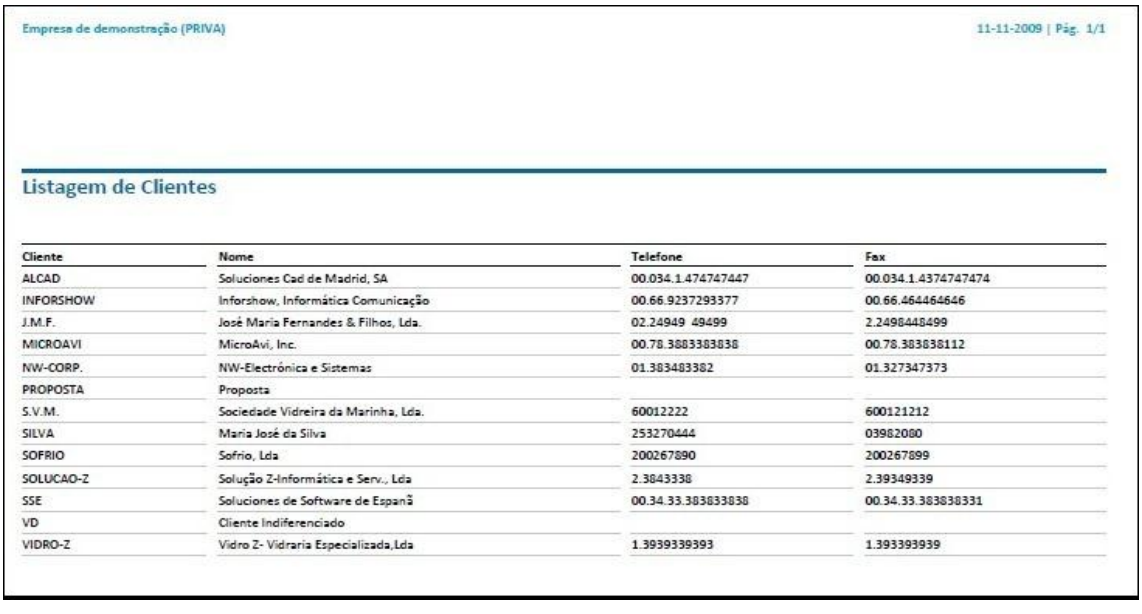
Fig 3.1 - Definição conceptual de um relatório

Com este diagrama pretende-se definir quais os conceitos que estão envolvidos num relatório, para a sua construção e para seu processamento.

- **DataSource:** os dados que são usados no relatório. Estes dados podem ser mapeados directamente da base de dados nos cenários mais simples, sendo que em relatórios mais complexos existe a necessidade de usar tabelas temporárias devido à elevada complexidade para calcular a informação a usar no relatório.
- **Template:** define a aparência do relatório, em termos estruturais e de estilo dos controlos (texto, linhas, margens).
- **Layout:** define a estrutura do relatório.
- **Styles:** define o estilo dos controlos usados no relatório.
- **Pre-processing:** é o processamento que é necessário efectuar antes de colocar os dados no relatório. Nos relatórios mais simples, pode não ser necessário efectuar este processamento, sendo apenas necessário nos relatórios mais complexos.
- **Output:** é o que o utilizador vai obter como resultado. Existem várias formas de output: pode ser visualizado como pré-visualização, pode ser enviado directamente para uma impressora, ou pode ser gravado num ficheiro (pdf, doc, xls)

Podemos agora fazer um conjunto de questões sobre os relatórios: Como são construídos? Como é definida a sua aparência? Como é colocada a informação no relatório?
Usando um exemplo prático para mais fácil compreensão, imaginemos que o utilizador quer um relatório com a lista dos seus clientes (Fig 3.2). Os dados dos clientes estão armazenados numa base de dados, e são apresentados ao utilizador. Para passar esses dados para um relatório é necessária a existência de um "mapa", onde está definido qual a informação que vai ser processada e qual a sua aparência. Um mapa é um ficheiro que é usado pelo motor de impressão para processar os dados que este vai receber.

Ainda no exemplo da lista de clientes, imaginemos que o utilizador quer um relatório onde seja apresentado o identificador do cliente, o seu nome, e os números de telefone e fax. Para isto, é necessário construir um mapa, onde esteja definido que vão ser estes os dados a usar no relatório, e também qual a sua aparência, desde o tipo de fonte a utilizar, até a posição dos campos no relatório.


Empresa de demonstração (PRIVA) 11-11-2009 | Pág. 1/1

Listagem de Clientes

Cliente	Nome	Telefone	Fax
ALCAD	Soluciones Cad de Madrid, SA	00.034.1.474747447	00.034.1.4374747474
INFORSHOW	Inforshow, Informática Comunicação	00.66.9237293377	00.66.464464646
J.M.F.	José Maria Fernandes & Filhos, Lda.	02.24949 49499	2.2498448499
MICROAVI	MicroAvi, Inc.	00.78.3883383838	00.78.383838112
NW-CORP.	NW-Electrónica e Sistemas	01.383483382	01.327347373
PROPOSTA	Proposta		
S.V.M.	Sociedade Vidreira da Marinha, Lda.	60012222	600121212
SILVA	Maria José da Silva	253270444	03982080
SOFRIO	Sofrio, Lda	200267890	200267899
SOLUCAO-Z	Solução Z-Informática e Serv., Lda	2.3843338	2.39349339
SSE	Soluciones de Software de Espanã	00.34.33.383833838	00.34.33.383838331
VD	Cliente Indiferenciado		
VIDRO-Z	Vidro Z- Vidraria Especializada,Lda	1.3939339393	1.393393939

Fig 3.2 – Relatório exemplo de uma lista de clientes

Mas agora se o utilizador quiser a mesma lista de clientes, mas que seja apresentada também a morada do cliente? Apesar de continuar a ser uma listagem de clientes, que vai ser apresentada da mesma forma que a anterior, estamos a adicionar um novo campo. Uma vez que este novo campo não está definido no mapa anterior, é necessário construir um novo mapa. Com este exemplo conseguimos perceber um dos principais problemas do actual componente de reporting, que é a não existência de nenhuma forma de partilhar características comuns entre vários mapas. Apesar das características destas duas listagens serem quase na sua totalidade iguais, como por exemplo a fonte de dados (clientes), o layout ou forma como os campos estão formatados (tipo de fonte, cores), não é possível partilhar estas configurações entre os dois mapas, logo, é necessário na construção de cada um definir sempre cada uma das configurações. O impacto desta limitação num produto com dimensão do ERP PRIMAVERA é bastante evidente, e que traduzida em números leva a que seja necessário a existência de cerca de 2000 mapas diferentes para responder a todas as necessidades de reporting. Este esforço de criação e manutenção destes mapas é muito elevado, sendo por isso necessário encontrar uma solução que permita reduzir substancialmente este esforço.

Olhando novamente para o exemplo da lista de clientes, vemos que quer na primeira listagem (sem morada) quer na segunda (com morada), o processo de construção dos relatórios é o mesmo. Mesmo em termos de estrutura são os dois semelhantes, onde a informação é apresentada em forma de tabela, em que a única diferença é que no segundo caso existe mais uma coluna. A pergunta que fica, é se com a tecnologia existente actualmente, será mesmo necessária a criação de mapas para a construção de relatórios.

A tecnologia em que está assente a componente de reporting do ERP é o Crystal Reports. Esta tecnologia não está actualmente à altura dos objectivos propostos pela Framework Athena para o componente de reporting, que passa principalmente pela geração automática de relatórios. Para que este objectivo seja cumprido, é necessário perceber exactamente quais as necessidades existentes para os produtos PRIMAVERA em termos reporting. Para isso é necessário analisar os produtos existentes, mais concretamente o ERP Priamavera.

Idealmente, o objectivo é que a totalidade dos mapas seja gerada automaticamente sem que haja qualquer distinção entre os mapas. Como seria de esperar, esta é uma visão um pouco desviada da realidade, dado o elevado número de mapas bem como a complexidade de alguns deles. Olhando para o conjunto dos mapas, apercebemo-nos de que estes podem ser divididos em grupos, consoante as suas características e também dependendo do seu propósito. Para cada um dos tipos é necessário proceder à análise das várias funcionalidades que são usadas no Crystal Reports para a construção de cada um, isto para que na nova solução se consigam abranger todos os casos de relatórios.

3.1.1 Características gerais dos mapas

Todos os mapas usados no ERP PRIMAVERA têm uma base comum, isto é, determinadas características estão presentes em todos eles.

A grande maioria dos mapas é composta pelos mesmos componentes estruturais, por exemplo o cabeçalho, detalhes, rodapé, definição de margens. Existem também alguns campos que podem estar presentes em todos os mapas, é o caso do título, do número de páginas ou da

data. Outros campos sempre presentes são o nome da empresa, o *copyright* e o nome da licença.

Em Crystal Reports (Fig 3.3), um mapa é constituído por um conjunto de diferentes áreas ou secções, que definem como são processados os controlos associados a cada uma delas.

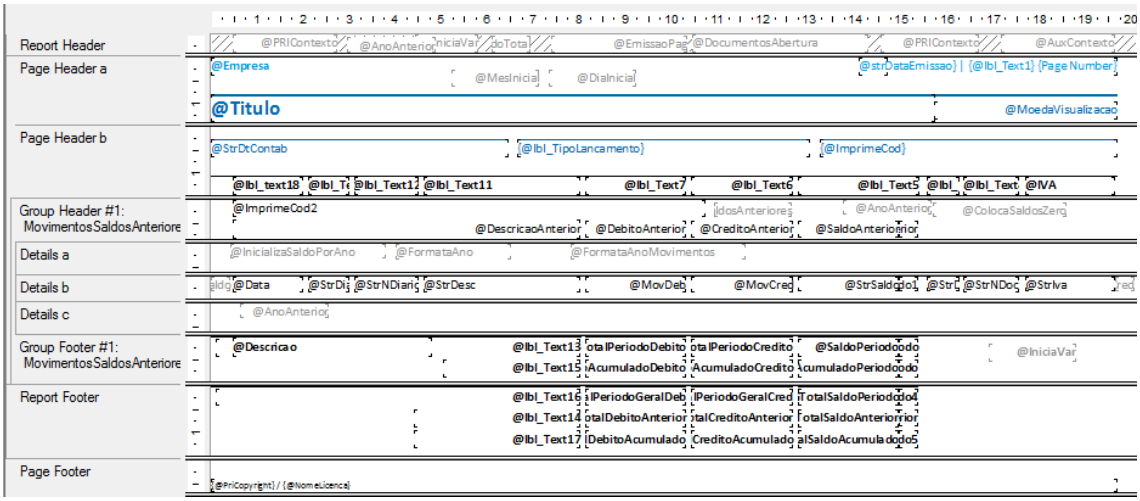


Fig 3.3 - Relatório no Crystal Reports

- **Report Header**: está localizado no início do relatório; pode ser utilizado para apresentar alguma informação introdutória.
- **Report Footer**: está localizado no fim do relatório.
- **Page Header**: cabeçalho da página; apresenta informação no início de cada página.
- **Page Footer**: está localizado no fim de cada página.
- **Details**: está localizada na página entre todas as outras secções.
- **Group Header**: está presente no início de cada grupo
- **Group Footer**: localizado no final de cada grupo

As cinco primeiras secções apresentadas anteriormente estão na base de todos os relatórios, existindo contudo a possibilidade de configurar algumas opções de visualização das mesmas. É possível por exemplo esconder uma secção se uma determina condição for verificada.

Cada uma destas secções pode ser dividida em várias subsecções. Isto pode ser usado por exemplo para organizar a informação a colocar no relatório, colocando informação relacionada na mesma subsecção. Podem também ser criadas subsecções "vazias", isto é, que não contem nenhuma informação, servindo apenas para colocar espaço vazio entre outras secções/subsecções.

Existem ainda outros aspectos a considerar na construção de um mapa. Tendo o ERP PRIMAVERA suporte para diferentes línguas, os mapas têm também de suportar os diferentes idiomas. Significa isto que no mapa não podem ser colocados valores estáticos, por exemplo, o título do mapa não pode ser introduzido aquando da sua construção, pois este depende de qual o idioma que está a ser usado. Então tem de ser criada uma fórmula para este campo, sendo depois processado quando o relatório é criado.

3.1.2 Tipos de Mapas

De seguida irá ser realizada a apresentação formal dos tipos de mapas que são utilizados pelo ERP PRIMAVERA. Após análise de uma percentagem aceitável dos 2000 mapas que compõe a aplicação, estes foram agrupados em 3 grandes categorias.

Esta classificação tem por objectivo a redução do número de mapas através da sua tipificação, bem como o estabelecimento de regras para cada um dos tipos. A definição de regras tem como objectivo simplificar o processo de criação de relatórios, quer esta seja manual ou automática.

3.1.2.1 Mapa Tipo 1

Este tipo de mapa é o mais simples de construir. Os dados são apresentados em forma de tabela. São usados normalmente para apresentar listagens, como por exemplo a lista de artigos, de clientes, fornecedores, plano de contas. Neste tipo de mapas não existe a necessidade de calcular fórmulas complexas, uma vez que os dados são na maioria das vezes mapeados directamente da base de dados, em que os dados de cada coluna na base de dados são inseridos numa coluna do relatório (ver exemplo em **Anexo A**).

Características:

- Em forma de tabela
- Directamente da base de dados
- Filtros
- Grupos
- Totalizadores simples

Os mapas analisados deste tipo foram os seguintes: lista de clientes (**Anexo A**), lista de artigos, lista de fornecedores e lista do plano de contas.

3.1.2.2 Mapa Tipo 2

Estes mapas têm origem em janelas que servem de suporte a todo o ERP PRIMAVERA. São janelas que usualmente apresentam regras de negócio simples, mas que trazem por isso uma complexidade mais elevada aos mapas. O objectivo é que o mapa reflicta o que está a ser mostrado no ecrã. Nestas janelas, o utilizador tem a capacidade de seleccionar os campos ou grupos de informação que pretende que estejam visíveis, o que torna necessário que os mapas possuam a capacidade de mostrar ou não determinada informação, dependendo das opções seleccionadas pelo utilizador.

Estes mapas são usados para impressão de entidades, por exemplo imprimir a ficha de um cliente ou artigo. Cada entidade possui diferentes campos, o que torna a lógica de construção de um mapa mais complexa. Uma vez que a estrutura destes mapas não é uma simples tabela, torna-se necessário ter mais cuidados em relação ao posicionamento dos controlos (ver exemplo em **Anexo B**).

Características:

- Aplicação de filtros
- Grupos

- Totalizadores
- Cálculo de campos

Foram analisados os seguintes mapas: ficha de artigo (**Anexo B e C**), ficha de cliente e ficha de fornecedor.

3.1.2.3 Mapa Tipo 3

Este tipo de mapas é o mais exigente. Estes podem ser mais complexos quer no tipo de processamento para calcular determinados campos, quer por vezes na sua estrutura, tendo esta de ser mais cuidada por se tratar de documentos que terão por vezes de ser entregues a entidades exteriores à empresa que os cria (bancos, finanças).
Uma das características principais deste tipo de mapas é a elevada complexidade no cálculo de determinados campos. São usadas por vezes tabelas temporárias devido à enorme complexidade presente na lógica de negócio para calcular os dados a introduzir no mapa (ver exemplo em **Anexo D**).

Características:

- Layout pode ser de elevada complexidade
- Uso de tabelas temporárias
- Cálculo de campos complexos
- Grupos
- Filtros
- Totalizadores complexos

Para este tipo, foram analisados os seguintes mapas: factura (**Anexo D**), extracto de conta.

3.1.3 Regras na construção de mapas

Para construir um novo mapa é necessário seguir um conjunto de regras. Estas regras são genéricas e que devem ser aplicadas a cada um dos tipos acima referidos. Por exemplo, elas definem qual o tipo de letra a usar, qual o cumprimento das margens, as cores de determinados campos, a espessura das linhas usadas numa tabela. Estas regras podem ser vistas como um guia para construir um relatório.

Para mais informação sobre estas regras, consultar o documento da PRIMAVERA: *Reports PBSS, Guidelines and usage rules* (versão 1 de Janeiro 2009).

3.2 Framework de Reporting

Nesta secção estão descritos os requisitos de *reporting* para a Framework ATHENA.

Olhando para os problemas da componente de *reporting* actual, facilmente se conseguem extrair os objectivos principais para o ATHENA. Temos como principal objectivo eliminar/reduzir o processo manual de construção de relatórios, passando estes a serem gerados automaticamente.

Automatizar o processo de construção dos relatórios significa acabar com a necessidade de os criar fisicamente, passando os relatórios a ser gerados em runtime. A ideia é tornar possível esta geração a partir dos modelos das aplicações.

As vantagens ganhas com esta nova abordagem são imensas, começando desde logo no desenvolvimento das aplicações, uma vez que todo o processo de criação de mapas deixa de ser necessário. Isto traduz-se num aumento significativo da produtividade na fase de desenvolvimento, e também um ganho depois na fase de manutenção, pois os erros introduzidos no processo manual de construção dos mapas deixam de existir.

Existem também novas funcionalidades que podem ser dadas ao utilizador final, como por exemplo, escolher exactamente qual a informação que quer nos relatórios. Voltando ao exemplo da listagem de clientes apresentado anteriormente, para que fosse possível utilizar diferentes combinações de campos, era necessário criar para cada combinação um mapa. Agora passa a ser possível indicar quais são os campos a colocar no relatório, sendo este gerado em runtime.

A Framework de Reporting tem de suportar os seguintes requisitos:

- Geração automática de relatórios a partir dos modelos das aplicações.
- Disponibilização de um motor de impressão de relatórios.
- Disponibilização de uma API programática para o motor de impressão.
- Impressão centralizada e remota de relatórios.
- Disponibilização de um designer para construção de novos relatórios e/ou adaptação (modificação) de relatórios existentes.

Requisitos funcionais do motor de reporting:

- Capacidade de agrupar a informação
- Guardar o layout
    - Capacidade de guardar o layout do report, para que seja possível que o utilizador o possa editar.
- Exportar para diferentes tipos de documentos (PDF, Excel, html, Text File ect.)
- Criação de secções do relatório.
    - Capacidade de criar secções como por exemplo o cabeçalho ou o rodapé do relatório.
- Formatação condicional.
    - Refere-se à capacidade de formatar o conteúdo do relatório de acordo com uma determinada condição.
- Ordenação dos dados
- Filtrar dados de acordo com um determinado critério
- Criação de expressões
- Quebra de página
- Cálculo de campos
    - Capacidade de criar campos com base em cálculos efectuados sobre outros campos.
- Cálculo de sumários
    - Capacidade de aplicar funções aritméticas para obter um resultado.
- Possibilidade de passar parâmetros para o relatório.
- Capacidade de alterar/configurar o layout da página para A4, A3, landscape, portrait.

- Criação de relatórios dinamicamente
- Definição de templates
- Disponibilização de uma API
- Possibilidade de configurar/escolher a fonte de dados

De seguida são apresentados os requisitos para o motor de impressão:

- Existência de um template de sistema por omissão.
    - Definir um template para ser usado por um determinado tipo de relatórios. Neste template devem já estar definidos: cabeçalhos, rodapé, números de página, título do relatório, data.
- Personalização de templates.
    - Deve ser possível aos utilizadores personalizar os templates. O utilizador pode criar novos templates com base em templates já existentes, onde podem fazer alterações básicas, como o cabeçalho ou o título.
- Possibilidade de o utilizador seleccionar uma página ou um conjunto de páginas para imprimir.
- Os dados no relatório têm de reflectir o que está a ser apresentado ao utilizador. Tem de ser suportados agrupamentos, filtros, formatações.

Como já foi dito anteriormente, a tecnologia actual sobre a qual estão assentes todas as funcionalidades de reporting, está desactualizada e não se adequa aos propósitos da Framework ATHENA. Existe então a necessidade de seleccionar uma nova ferramenta sobre a qual será desenvolvida a Framework de Reporting, que permita responder aos requisitos apresentados anteriormente. Existe também a necessidade de disponibilizar um End-User Designer que permita responder aos seguintes requisitos:

- Edição/ criação de relatórios. O utilizador define quais os campos que quer no relatório.
- Dar a possibilidade de o utilizador editar ou criar novos templates.
- Alteração da fonte de dados
- Facilidade de utilização
- Possibilidade de extensibilidade das funcionalidades base

3.3 Análise de componentes de reporting

Nesta secção é feito o levantamento do estado actual de algumas ferramentas de reporting. Este estudo tem como motivação seleccionar a ferramenta que melhor serve os objectivos da Framework de Reporting, e que será usada como base no seu desenvolvimento. Alguns dos aspectos a ter em consideração nesta análise são:

- Maturidade da ferramenta.
- Disponibilização de uma API que permita criar relatórios dinamicamente.
- Possibilidade de extensibilidade dos seus componentes e de criação de novos componentes.
- Funcionalidades e usabilidade do designer, e também a sua extensibilidade.
- Qualidade ao nível da documentação e de suporte ao cliente.

3.3.1 ActiveReports

O ActiveReports (19) apresenta-se como uma das possíveis ferramentas para utilizar como base no desenvolvimento da Framework de Reporting. A versão analisada desta ferramenta foi o ActiveReports 6.0.

Listam-se a seguir algumas das características desta ferramenta:

- Permite o uso de vários tipos de fonte de dados: OleDBDataSource, SqlDBDataSource, ListDataSource, XMLDataSource.
- Existe a possibilidade de passar apenas o *DataSet* (colecção com os dados) como fonte de dados. Neste caso, quando se guarda o relatório o *DataSet* não é guardado.
- Quando se guarda o relatório com a ligação à base de dados, esta pode ser facilmente alterada quando se faz o carregamento do relatório.
- Para filtrar ao nível da fonte de dados é necessário passar os parâmetros na expressão SQL, uma vez que os filtros usados no relatório apenas são aplicados depois de se obterem os dados da base de dados, que é uma desvantagem, pois no caso de grandes volumes de dados não é útil esta forma de os filtrar.
- Não tem a secção *Master-Detail* (apresentação hierárquica dos dados), não sendo por isso possível criar relatórios onde os dados sejam apresentados hierarquicamente.
- Permite exportar para pdf, html, xls, RTF, Text e TIFF.
- Possui um designer que pode ser usado no Visual Studio ou como end-user designer.
    - Quando utilizado como end-user designer é necessário construir os componentes do designer manualmente (tool bars…), e apesar de ser possível a sua personalização, esta é bastante complexa.
    - Possui alguns problemas de usabilidade, como por exemplo, quando se cria uma conexão à base de dados é necessário introduzir a expressão SQL manualmente, o que obriga a quem desenha o relatório tenha de estar familiarizado com esta linguagem.
    - É possível criar o componente sub-report, mas não é possível associar este componente a um report (isto tem de ser feito por código).
- Documentação fraca, o que dificulta bastante a aprendizagem desta ferramenta.

A principal desvantagem desta ferramenta está no seu designer (Fig 3.4), apresentando bastantes problemas de usabilidade. Um destes problemas é por exemplo quando se pretende associar ao relatório uma fonte de dados, onde é necessário, depois de escolher a base de dados introduzir a expressão SQL manualmente. Outra desvantagem é quando se pretende inicializar o designer por código, sendo neste caso necessário codificar todos os componentes (toolbars) que se pretendem disponíveis para o utilizador.
Outra desvantagem é não possuir a secção Master-Detail, não sendo por isso possível criar relatórios apresentando os dados hierarquicamente. Outro problema que esta ferramenta apresenta, é em termos de documentação, não sendo esta a melhor. Isto leva a que o tempo de aprendizagem e de resolução de problemas seja por vezes bastante elevado.

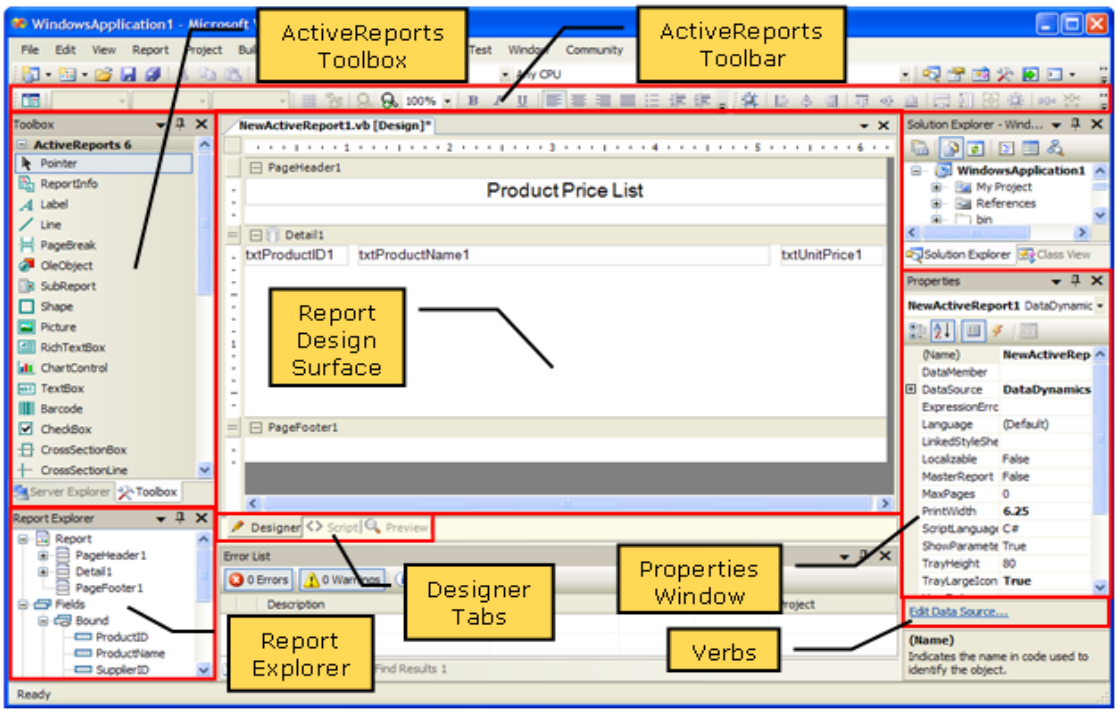


Fig 3.4 - Active Reports Designer

3.3.2 StimulSoft Reports

Nesta secção será analisada o Stimulsoft Reports (20).

Listam-se a seguir algumas das características mais relevantes desta ferramenta:

- Possui uma API bastante completa, que permite a criação de relatórios facilmente.
- Permite ter várias secções (bands) do mesmo tipo para todos tipos existentes. Significa isto que é possível no mesmo relatório utilizar várias fontes de dados, que podem ou não ser completamente independentes. Desta forma é possível criar dentro do mesmo relatório vários sub-relatórios sem a necessidade recorrer efectivamente ao componente sub-report.
- Possibilidade de passar como fonte de dados apenas o DataSet, permitindo assim que os dados sejam obtidos previamente sem que seja necessário que *report-engine* efectue *queries* à base de dados.
- Existe o conceito de página (Page): podem ser criadas várias páginas, cada uma com as suas secções (bands). Cada página só aparece depois de anterior ter sido completamente processada. Isto pode ser considerado mais uma forma de criar sub-reports.
- Conceito de *Panel*: em cada página é possível criar vários *Panels*, sendo cada um colocado numa posição qualquer. Dentro de cada um podem ser colocadas quaisquer tipos de secções.
- Quando se guarda o layout do relatório, as *connection strigns* são também guardadas, e estão acessíveis no designer ou por código, podendo facilmente ser alteradas. Isto permite que depois de se gerar um relatório, facilmente se consegue alterar a sua fonte de dados.

- Para aplicar filtros ao nível da fonte de dados é necessário passar os parâmetros na expressão SQL, o que pode trazer alguma complexidade na construção de relatórios que comportem um elevado volume de dados.
- Permite exportar para vários tipos de documentos: Csv,Excel,ExcelXml,Excel 2007,Html,Images,Pdf, Rtf,Txt, Xml.
- Em termos de documentação é bastante fraca, não apresentando também uma boa base para suporte aos problemas dos utilizadores.

Esta ferramenta apresenta-se bastante completa, quer em termos de API, quer também no designer (Fig 3.5), sendo grande a variedade de formas de que se pode construir um relatório. Uma das principais vantagens é facto de se poderem criar várias secções de cada um dos tipos, permitindo assim que ter várias secções do tipo *Data*. Significa isto que é possível ter para o mesmo relatório várias fontes de dados, sem que haja qualquer relação entre elas. Um dos pontos fracos presente nesta ferramenta é a sua documentação, sendo esta bastante fraca. Em termos de suporte/comunidade também não se apresenta muito favorável, sendo isto um ponto negativo muito importante, pois leva a que até os problemas mais simples de resolver sejam por vezes ultrapassados apenas depois de muitas horas e de múltiplas tentativas.

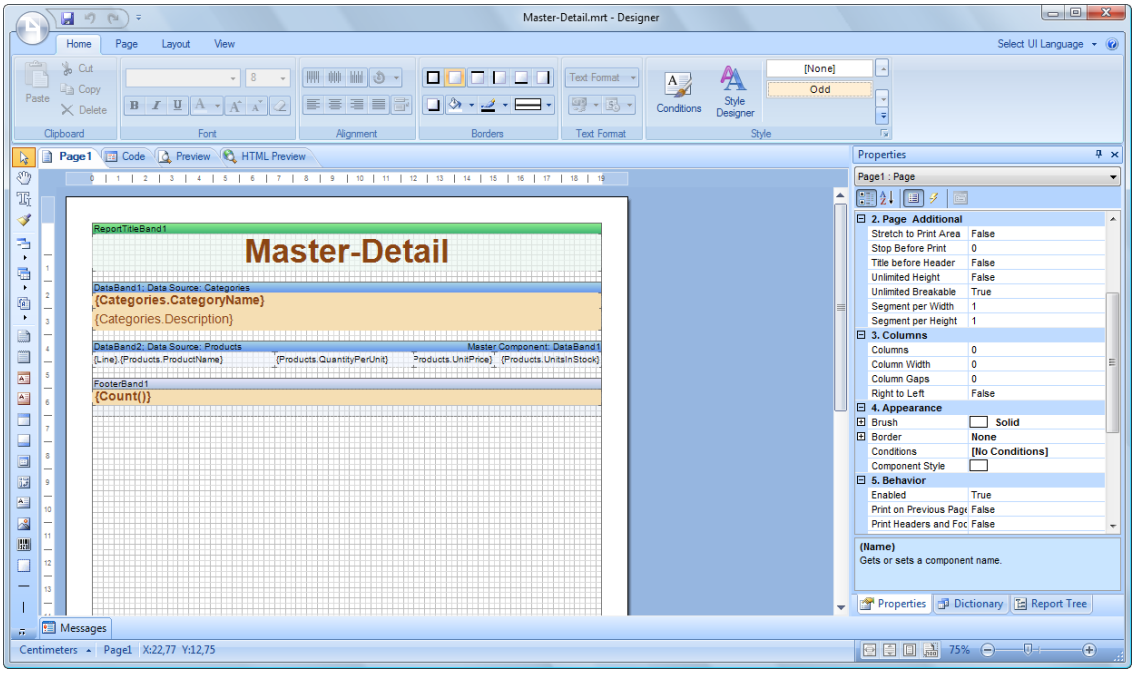


Fig 3.5 - Stimulsoft Report Designer

3.3.3 DevExpress XtraReports

Por último será analisado XtraReports da DevExpress (21). A versão analisada desta ferramenta foi a DXperience 9.3.1.

Listam-se de seguida algumas das suas características:

- Apresenta uma API bastante completa, que permite criar relatórios dinamicamente de forma simples.
- Permite associar vários tipos de fontes de dados: Sql Data Adapter, Table Adapter, OleDbDataAdapter.
- Permite passar como fonte de dados do relatório apenas um DataSet, permitindo desta forma que os dados sejam obtidos antes de se gerar o relatório.
- É possível guardar o relatório gerado em ficheiro ou numa base de dados, podendo ser carregado posteriormente, sendo possível efectuar operações sobre a fonte de dados. Mas para que isto seja possível é necessário garantir que quando de guarda o relatório, também são guardados os *adapters* da fonte de dados, sendo estes depois acedidos aquando do carregamento do relatório. Neste cenário é também possível gerar o DataSet a partir dos *adapters* guardados no relatório.
- É possível alterar a *connection string*, uma vez que se tem acesso aos DataAdapter's.
- Para aplicar filtros ao nível da fonte de dados é necessário passar os parâmetros na expressão SQL.
- Possui também um designer bastante completo para a construção de relatórios. Apesar de algumas funcionalidades não serem suportas, estas são bastante "simples" de implementar, pois é possível fazer *override* de todas a operações do designer. Uma das limitações do designer é por exemplo quando se carrega um relatório previamente construído. Neste caso, não é feito o carregamento dos *DataAdapters*, sendo por isso necessário implementar uma operação que trate este caso.
- Permite exportar para vários formatos: PDF, HTML, MHT, RTF, TXT, CSV e Excel, e para imagens: BMP, EMF, GIF, JPEG, PNG, TIFF, WMF.
- Possui uma documentação bastante completa, onde são apresentados também bastantes exemplos práticos, o que torna a aprendizagem desta ferramenta bastante mais fácil que as apresentadas anteriormente. No que diz respeito ao suporte ao cliente, apresenta também uma boa base de dados, com os problemas de outros utilizadores, tendo também vários fóruns de discussão.

Esta ferramenta apresenta-se bastante completa, sendo bastante simples criar um relatório quer por código quer usando o designer. Apresenta uma API que permite realizar todas as operações necessárias sobre os relatórios, sendo ao mesmo tempo bastante simples de compreender e usar.
O designer (Fig 3.6) apresenta no entanto alguns pequenos problemas de usabilidade, como por exemplo no caso dos sub-reports. Não é possível esta operação usando apenas o designer, sendo necessário associar o relatório principal aos sub-reports por código.
Um ponto forte desta ferramenta é em termos de documentação e suporte, sendo a sua qualidade bastante boa. Normalmente quando existe a necessidade de resolver um problema, este, ou se encontra na documentação, ou se encontra no fórum da comunidade, sendo bastante fácil encontrar uma solução para os problemas.

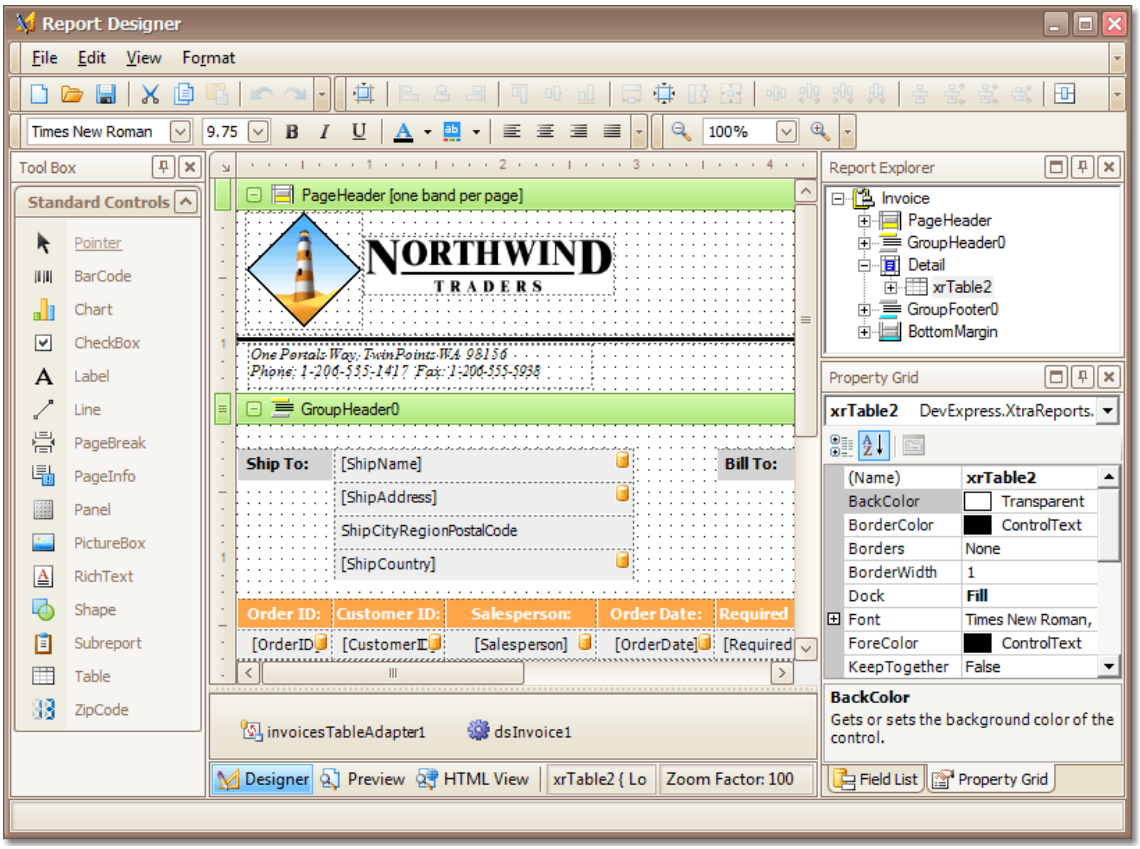


Fig 3.6 - XtraReports Report Designer

3.3.4 Comparação das ferramentas

Comparando agora a três ferramentas, podemos afirmar que o Active Reports e o XtraReports são até muito semelhantes. São bastante parecidos na forma de construir um relatório, quer ao nível do código quer ao nível do designer. As secções de um relatório são as mesmas nas duas ferramentas, apenas com a desvantagem de o Active Reports não possuir a secção Master-Detail. Apesar das semelhanças, o XtraReports apresenta num nível superior quando se fala em usabilidade do designer ou na facilidade de construir um relatório. O XtraReports possui também um *wizard* que permite a criação inicial de um relatório.

A conclusão a que se chega comparando o Active Reports e o XtraReports, é que o Active Reports não trás nada de novo em relação ao XtraReports, isto é, tudo que é possível fazer no Active Reports, é também possível fazer no XtraReports de forma mais simples. Existe também a grande vantagem para o XtraReports no que diz respeito à documentação e ao suporte, sendo neste aspecto bastante superior.

Fazendo agora a comparação entre o XtraReports e o Stimulsoft Reprots, apercebemo-nos que a principal vantagem do Stimulsoft Reports está no facto de ser possível criar várias secções do mesmo tipo. Tirando este facto, o XtraReports apresenta-se superior em vários pontos: na simplicidade de criar relatórios por código, tendo uma API menos complexa e mais simples de perceber; possui também uma documentação e um suporte bastante superior, sendo resolvidos os problemas mais rapidamente que no caso da Stimulsoft Reports.

Não sendo um ponto-chave para a Framework de Reporting o facto de ser necessário criar várias secções do mesmo tipo, e apresentando-se o XtraReports como uma boa ferramenta, que responde às necessidades da Framework de Reporting e que em termos de suporte/documentação é bastante superior às outras aqui analisadas, é esta uma boa escolha para o desenvolvimento da Framework de Reporting.

Capítulo 4

4 Reporting Framework

Neste capítulo são descritos os detalhes relativos à análise/desenho e implementação da Framework de Reporting.

A Framework de Reporting é integrada na Framework Athena como sendo um módulo desta framework. Isto é, terá de ser construído um módulo desenvolvido sob a Framework Athena, respeitando todos os seus princípios, sendo que neste módulo ficará o componente de reporting. Este módulo passará a ser referido de agora em diante como o **módulo de impressão** e será descrito de forma mais detalhada no capítulo seguinte. Nas secções que se seguem está descrita cada uma das etapas do desenvolvimento da framework de reporting desde a análise/desenho até à implementação.

4.1 Arquitectura

Nesta secção pretende-se apresentar de uma forma geral a Framework de Reporting, dando a conhecer os principais componentes desenvolvidos e de que forma estão organizados. Na Fig 4.1 é apresentada uma visão geral de arquitectura da Framework de Reporting:

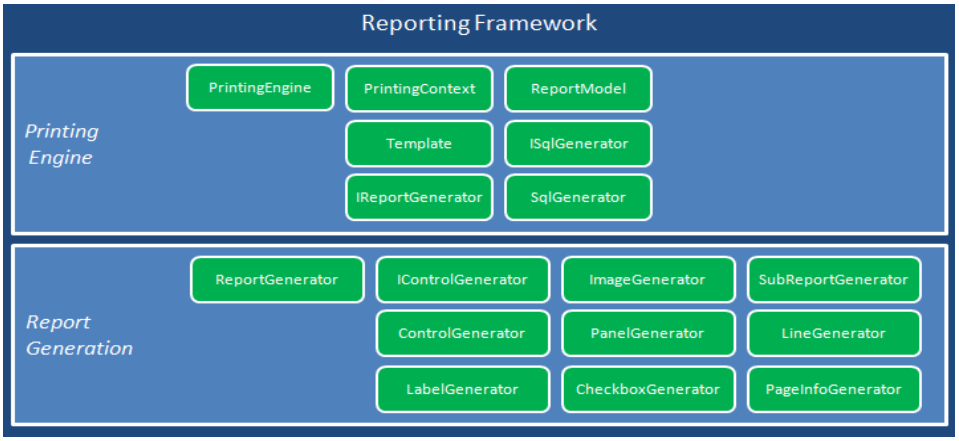


Fig 4.1 - Arquitectura: Reporting Framework

Como podemos observar na Fig 4.1, a Framework de Reporting está dividida em dois componentes principais: o modelo de dados que suporta e geração (***PrintingEngine***), e o próprio componente de geração (***ReportGeneration***).

O componente a partir do qual será executa a geração de *reports* é o **motor de impressão** (*PrintingEngine*). Este motor recebe um conjunto de dados como o template ou o *ReportModel*, e executa o componente de geração de acordo com determinadas definições definidas no contexto de impressão (*PrintingContext*). O motor de impressão não tem definido o tipo concreto para o componente de geração, mas sim uma interface (***IReportGenerator***) através da qual executa a geração dos *reports*, sendo por isso necessário que este seja inicializado com o tipo concreto para a geração. Este tipo de implementação tem a grande vantagem de o motor de impressão não estar dependente de uma implementação concreta do componente de geração, permitindo assim que este possa ser facilmente substituído sem que

afecte os restantes componentes. Por exemplo, este componente de geração vai gerar *reports* para a para a ferramenta seleccionada anteriormente, o ***XtraReports***, mas se um dia se pretender substituir esta ferramenta por algum motivo, apenas será necessário implementar um novo componente de geração, deixando tudo o resto inalterado. Outra vantagem de o componente de geração ser totalmente independente, é o facto de poder seguir uma arquitectura e conceitos próprios.

Os detalhes de cada um dos componentes desenvolvidos para a Framework de Reporting são apresentados nas seguintes secções.

4.2 Motor de impressão

O motor de impressão (***PrintingEngine***) é o componente principal da Framework de Reporting, isto porque é esta classe que implementa os métodos para as funcionalidades de impressão. Na Fig 4.2 está representado o motor de impressão e os seus componentes principais:

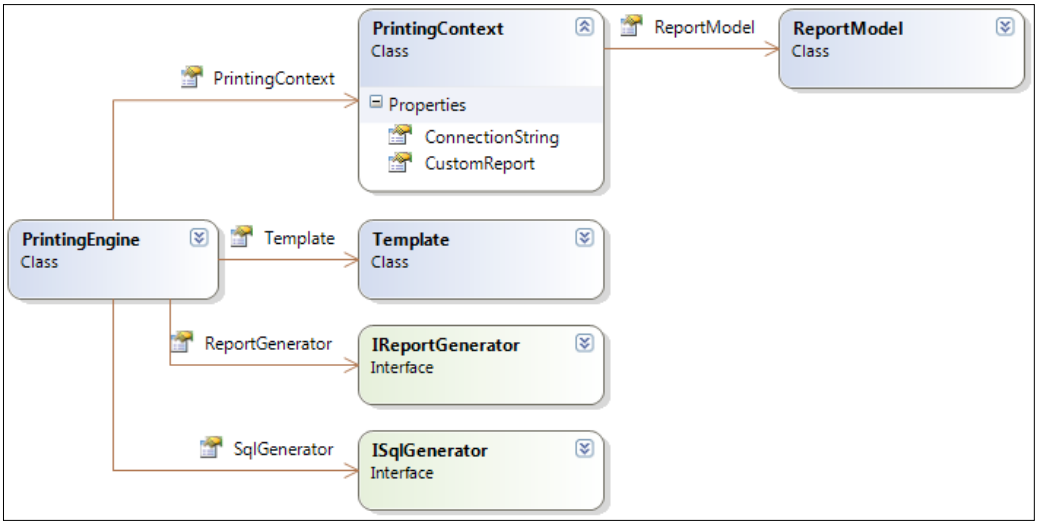


Fig 4.2 - Motor de impressão

O motor de impressão tem por objectivo inicializar o componente de geração, e executar a respectiva geração com base no contexto de impressão (***PrintingContext***) e no template (***Template***). A interface *IReportGenerator* expõe os métodos necessários à geração de um report com base no contexto de impressão e num determinado template. No contexto de impressão encontram-se os dados relativos à conexão à base de dados, o *ReportModel* ou então um *CustomReport*. O motor de impressão executa o componente de geração de acordo com determinadas configurações no contexto de impressão. Por exemplo, o componente de geração tem a capacidade de gerar um report com base num *ReportModel* ou apenas processar um *CustomReport* aplicando determinadas configurações definidas no *Template*, sendo esta decisão tomada pelo motor de impressão de acordo com o que está definido no contexto de impressão, se o *ReportModel* ou o *CustomReport.*

A expressão SQL que vai ser executada no processamento dos dados na base de dados pode ser gerada a partir do *ReportModel*. O componente responsável por esta tarefa tem obrigatoriamente de implementar a interface *ISqlGenerator*, e pode ser passado como parâmetro para o motor de impressão.

O motor de impressão recebe também como parâmetro um template, que é responsável por toda a informação relativa a formatações e estilos que vão ser aplicados no report. Este componente será detalhado num capítulo posterior.

Uma vez que o motor de impressão executa o componente de geração pela interface *IReportGenerator*, tem a vantagem de ser possível a criação de diversas implementações para a geração de *reports*. Isto significa que se for necessário alterar o componente de geração, não é necessário alterar todos os outros componentes utilizados pelo motor de impressão. Isto torna a Framework de Reporting bastante mais flexível, não ficando exclusivamente dependente de uma implementação concreta para determinada ferramenta de Reporting. Como já foi referido num capítulo anterior, o componente de Reporting escolhido foi o ***XtraReports*** da ***DevExpress***.

4.3 Report Model

Pretende-se com o *Report Model* representar um report, definindo as suas secções e os respectivos campos em cada uma delas, contendo toda a informação necessária para que seja utilizado pelo componente de geração para gerar o respectivo report. Este modelo foi construído com base na análise realizada anteriormente às características de um report, e para além de definir os campos que aparecem em cada secção, define também a sua posição no report, sendo possível definir grupos de campos (*ReportContainer*), o respectivo alinhamento e orientação. Na Fig 4.3 apresenta-se o modelo de objectos do *Report Model*:

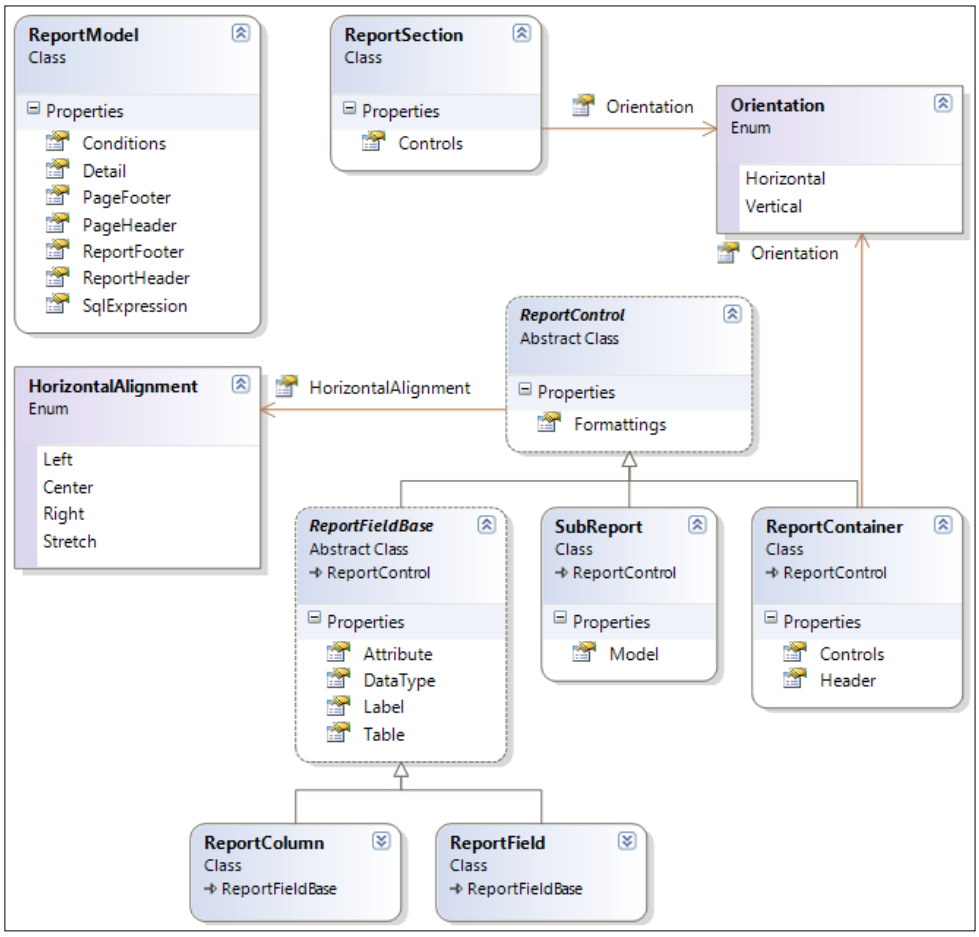


Fig 4.3 - Report Model

Os campos no *Report Model* são definidos pela classe base *ReportFieldBase*, tendo esta duas especializações, o *ReportColumn* e o *ReportField*. O motivo pelo qual se definiram um *ReportField* e um *ReportColumn* prende-se essencialmente com o facto de ser necessário distinguir os campos que são usados na impressão de entidades e na impressão de listas. Isto porque no caso das listas a descrição do campo (*label*) tem obrigatoriamente de ser colocado por cima da respectiva coluna. Por isso, a classe base contém todas as propriedades quer para os campos nas listas e nas entidades.

- **Attribute:** o nome da coluna na tabela.
- **DataType:** o tipo do campo. Por exemplo texto, numérico ou data.
- **Label:** a descrição do campo. Texto que identifica o campo no report, por exemplo o nome da coluna no caso de uma lista.
- **Table:** o nome da tabela onde está o campo que se pretende apresentar no report.

Para se compreender melhor de que forma se pode definir um *ReportModel*, analisemos o seguinte exemplo onde será utilizado o modelo de dados apresentado na Fig 4.4:

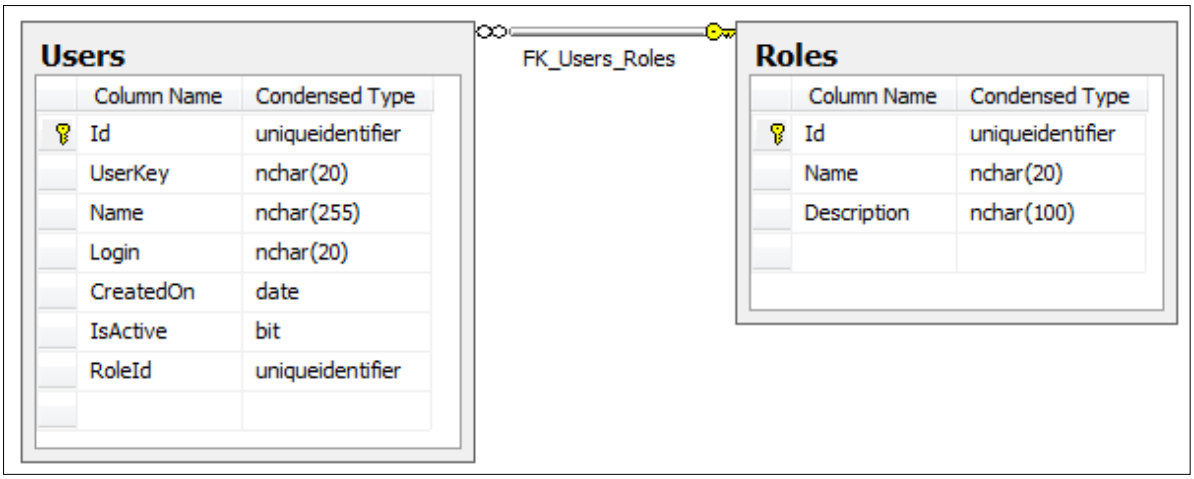


Fig 4.4 - Exemplo: Modelo de dados

Neste exemplo temos os dados de registo dos utilizadores. Temos então na base de dados duas tabelas, os utilizadores (***Users***) e o tipo de utilizador (***Roles***). Para criar um *report* de um determinado utilizador, com os campos nome, login, data de criação e o nome do *role* a que pertence, teríamos os seguintes campos no *Report Model*:

```
ReportField name = new ReportField()
{
    Label = "Name",
    Table = "Users",
    Attribute = "Name",
    DataType = "Text"
};

ReportField login = new ReportField()
{
    Label = "Login",
    Table = "Users",
    Attribute = "Login",
    DataType = "Text"
};

ReportField date = new ReportField()
{
    Label = "Date",
    Table = "Users",
    Attribute = "CreatedOn",
    DataType = "Date"
};

ReportField role = new ReportField()
{
    Label = "Role",
    Table = "Roles",
    Attribute = "Name",
    DataType = "Text"
};
```

Estes campos podem depois ser organizados de diferentes formas no report, colocando-os em diferentes secções e utilizado o controlo *ReportContainer* para definir grupos de campos. Na Tabela 4.1 é apresentada a descrição para os componentes que fazem parte do *Report Model*.

Componente	Descrição
ReportModel	Representa um *report*, definindo os campos a apresentar e seu layout. Trata-se da classe principal, sendo que um *Report Model* vai dar origem a um *report* gerado pelo componente de geração. Nesta classe estão definidas as secções (***ReportSection***) do report: *Detail*, *PageHeader, PageFooter, ReportHeader, ReportFooter.* Podem também ser definidas condições para serem aplicadas no processamento dos dados na base de dados, sendo possível desta forma por exemplo definir filtros para as listas ou filtrar por um determinado registo no caso da impressão de entidades.
ReportSection	Representa uma secção no *report*. Numa secção são definidos o conjunto de controlos (***ReportControl***) que dela fazem parte. Pode também ser definida a orientação (***Orientation***) pela qual os controlos são apresentados na respectiva secção.

ReportControl	É a classe base para todos controlos que podem ser adicionados ao *ReportModel*, podendo ser definida o tipo de alinhamento horizontal (***HorizontalAlignment***). A um *ReportControl* podem ser também associadas formatações condicionais (***Formattings***), sendo estas formatações aplicadas ao controlo mediante o resultado da condição que está associada à formatação condicional.
ReportContainer	Tem por objectivo agrupar um conjunto de controlos. Da mesma forma que numa secção, é também possível definir a orientação dos controlos que fazem parte do *container*. Sendo o *container* um controlo do *ReportModel*, é possível ter *containers* dentro de *containers*.
SubReport	O seu objectivo é dar a possibilidade de se poder definir um *report* "dentro" de outro *report*. Esta classe tem como propriedade o *ReportModel*, significa isto que num *SubReport* é possível ter todas a definições que se têm para qualquer outro *report*. Este de controlo tipo é utilizado em cenários *Master-Detail* e também no caso listas hierárquicas. Quer num caso quer no outro, é necessário definir para o *SubReport* a condição de filtro relativa ao *report* "pai", isto para que no *SubReport* apenas sejam apresentados os resultados correspondentes ao "pai".
ReportFieldBase	Classe base para os campos de num *report*.
ReportField	Representa um campo na impressão de uma entidade.
ReportColumn	Representa uma coluna na impressão de uma lista.
Orientation	Define a orientação (horizontal ou vertical) pela qual os controlos são apresentados, numa secção ou num *container*.
HorizontalAlignment	Define o alinhamento horizontal de um controlo. • **Left**: alinha o controlo à esquerda. • **Center**: o controlo fica alinhado ao centro. • **Right**: alinhamento à direita. • **Stretch**: o controlo ocupa todo o espaço disponível.

Tabela 4.1 - Report Model

4.3.1 Geração da expressão SQL

A expressão SQL pode ser definida directamente no *ReportModel*, ou então gerada a partir dos campos que nele estão definidos. O componente responsável por gerar a expressão SQL é executado pelo motor de impressão, e tem de implementar a interface *ISqlGenerator*.

Também aqui, o recurso a uma interface e não a uma implementação concreta tem a vantagem de se poderem criar diferentes implementações para diversos motores de base de dados sem a necessidade de alterar os restantes componentes da Framework de Reporting, apenas sendo necessário inicializar no motor de impressão com o gerador pretendido.

Na Fig 4.5 é apresentado o componente de geração da expressão SQL:

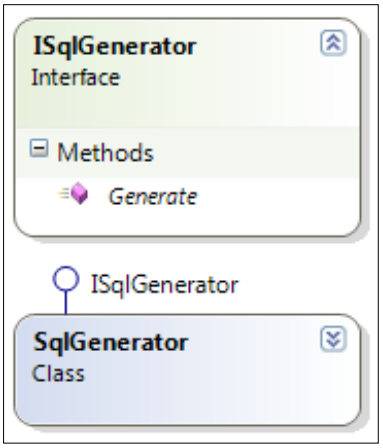


Fig 4.5 - Gerador da expressão SQL

Na Framework de Reporting foi implementado um gerador (***SqlGenerator***) para o motor de base da dados ***SqlServer 2008***, que é utilizado para gerar a expressão *SQL* sempre que esta não esteja definida e quando o motor de impressão não é inicializado com outro componente de geração.

A expressão SQL gerada para os campos apresentados no exemplo anterior é a seguinte:

```
SELECT    [Users].[Name] AS [Users.Name],
          [Users].[Login] AS [Users.Login],
          [Users].[CreatedOn] AS [Users.CreatedOn],
          [Roles].[Name] AS [Roles.Name]

    FROM [Users]
           INNER JOIN [Roles] ON [Users].[Id] = [Roles].[Id]
```

4.3.2 Condições

Ao *ReportModel* podem ser adicionadas condições para que seja possível filtrar os registos por um determinado critério. No caso da impressão de entidades, uma vez que apenas se pretende mostrar um registo único, a condição de filtro tem obrigatoriamente de existir e será relativa à chave primária dessa tabela. Para as listas, as condições já terão de suportar cenários mais complexos, uma vez que se pretende dar a possibilidade ao utilizador de filtrar a lista por qualquer coluna que esteja visível, e também a construção de condições complexas. Com base na condição definida no *ReportModel* é gerada a cláusula ***WHERE*** a ser posteriormente injectada na expressão SQL.

Na Fig 4.6 está represento o modelo de classes relativo às condições num *ReportModel:*

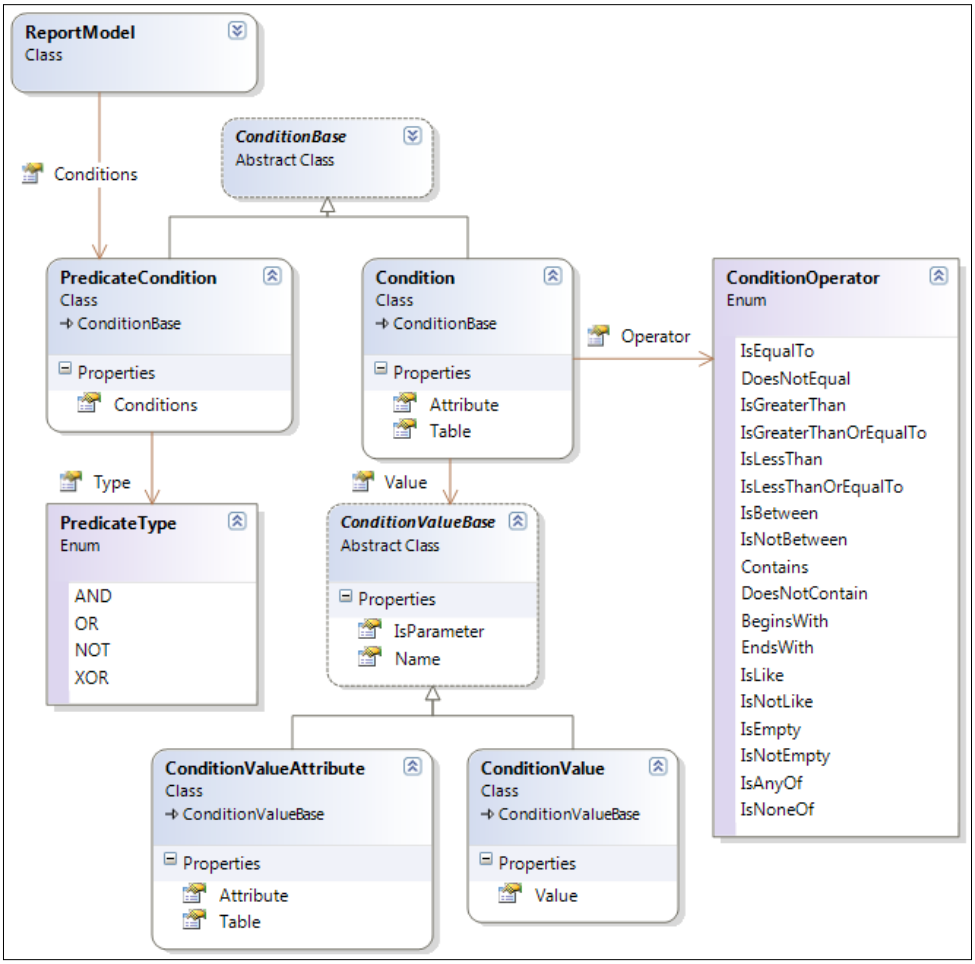


Fig 4.6 - ReportModel: Condições

Existem dois tipos de condições: ***PredicateCondition*** que representa um conjunto de condições e ***Condition*** que representa uma condição simples. Cada uma destas tem como base a classe abstracta ***ConditionBase***. Uma *PredicateCondition* contem uma lista de condições do tipo *ConditionBase*, o que significa que pode conter dentro desta lista outras condições do mesmo tipo, isto é, um conjunto de condições pode conter outros conjuntos de condições. Relativamente à classe ***Condition***, esta define a tabela e a coluna à qual vai ser aplicada a condição, sendo possível definir o valor da condição de duas formas: usando uma ***ConditionValue*** que faz a comparação com um valor fixo, ou usando uma ***ConditionValueAttribute*** que permite fazer uma comparação com uma coluna de outra tabela. O enumerado ***ConditionOperator*** representa os operadores que são possíveis aplicar a uma *Condition.*

Na Tabela 4.2 é apresentada a descrição de cada um dos componentes que fazem parte do modelo de condições do *ReportModel*.

Componente	Descrição
ConditionBase	Classe base para a condições que podem ser aplicadas ao *report.*
PredicateCondition	Representa um conjunto de condições.
Condition	Representa uma condição.
ConditionValueBase	Classe base para o valor de uma condição. O valor de uma condição pode ser marcado como parâmetro.

ConditionValue	Condição com um valor fixo. Exemplo: `[Users].[Name] = 'user'`
ConditionValueAttribute	Condição em que o valor é outra coluna da tabela. Exemplo: `[Users].[Name] = [Roles].[Name]`
PredicateType	Tipo de operador que é aplicado a um conjunto de condições. • **And**: todas as condições são verdadeiras. • **Or**: pelo menos uma condição é verdadeira. • **Not**: nenhuma condição é verdadeira. • **Xor**: alguma das condições pode ser verdadeira
ConditionOperator	Operador aplicado à condição.

Tabela 4.2 - ReportModel: Condições

Usando o exemplo do registo de utilizadores apresentado na Fig 4.4, uma possível condição poderia ser por exemplo listar todos os utilizadores que foram criados a partir do ano 2000 e em que o nome (coluna *Name*) começa ou termina com a letra 'a'. A seguir é apresentado este exemplo traduzido para o respectivo código C#:

```
PredicateCondition condition = new PredicateCondition()
{
    Type = PredicateType.AND
};

Condition c1 = new Condition()
{
    Table = "Users",
    Attribute = "CreatedOn",
    Operator = ConditionOperator.IsGreaterThan,
    Value = new ConditionValue()
    {
        Value = "10-01-2000"
    }
};
condition.Conditions.Add(c1);

PredicateCondition predicate = new PredicateCondition()
{
    Type = PredicateType.OR
};
condition.Conditions.Add(predicate);

Condition p1 = new Condition()
{
    Table = "Users",
    Attribute = "Name",
    Operator = ConditionOperator.BeginsWith,
    Value = new ConditionValue()
    {
        Value = "a"
    }
};
predicate.Conditions.Add(p1);

Condition p2 = new Condition()
{
    Table = "Users",
    Attribute = "Name",
    Operator = ConditionOperator.EndsWith,
    Value = new ConditionValue()
    {
        Value = "a"
    }
};
predicate.Conditions.Add(p2);
```

A expressão SQL gerada pelo componente de geração de SQL é a seguinte:

```
SELECT  [Users].[Name] AS [Users.Name],
        [Users].[Login] AS [Users.Login],
        [Users].[CreatedOn] AS [Users.CreatedOn],
        [Roles].[Name] AS [Roles.Name]

  FROM [Users]
        INNER JOIN [Roles] ON [Users].[Id] = [Roles].[Id]

        WHERE ([Users].[CreatedOn] > '10-01-2000'
          AND ([Users].[Name] LIKE 'a%'
               OR [Users].[Name] LIKE '%a'))
```

4.3.3 Formatação Condicional

Formatação condicional é a capacidade de definir certas propriedades de aparência de um controlo no *report* quando uma determinada condição é satisfeita.

Na Fig 4.7 é apresentado o modelo de classes utilizado para a formatação condicional:

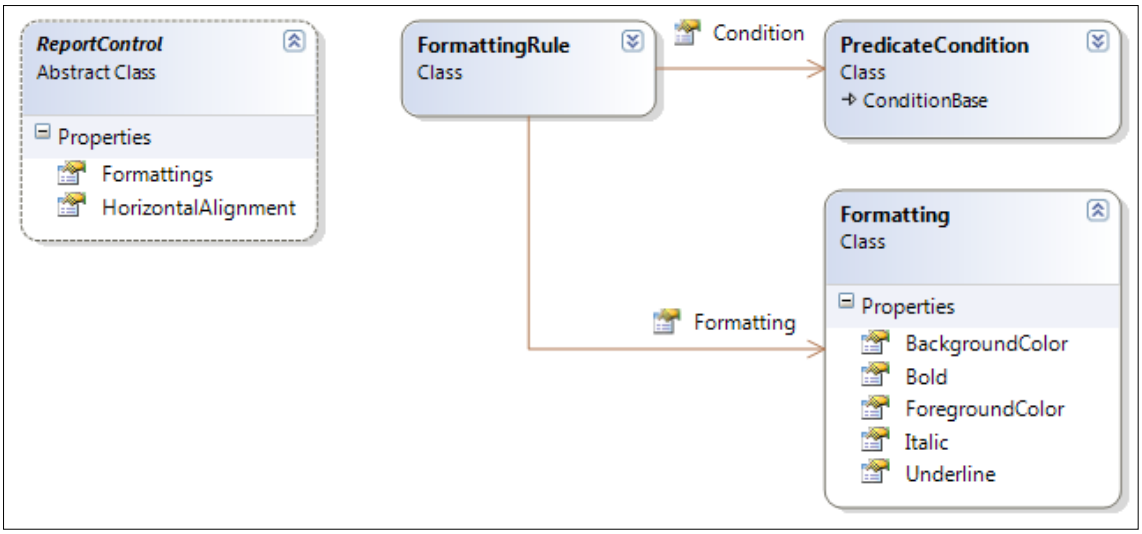


Fig 4.7 - Formatação condicional

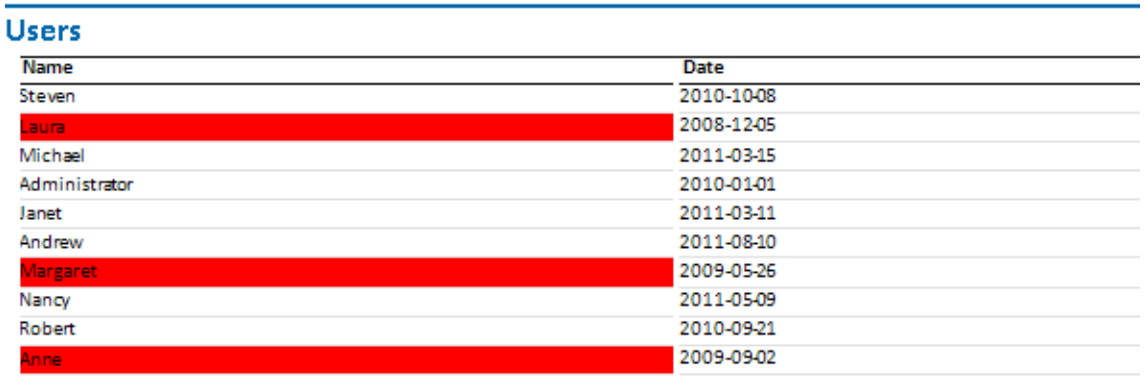
Na Tabela 4.3 é apresentada a descrição de cada um dos componentes que fazem parte do modelo da formatação condicional.

Componente	Descrição
ReportControl	Um *ReportControl* pode conter uma lista de formações condicionais (***FormattingRule***).
FormattingRule	Representa uma regra de formatação a ser aplicado a um controlo, a qual contem a formação e a respectiva condição.
PredicateCondition	Condição que é avaliada para a aplicação da formatação. Foi reaproveitado o modelo utilizado nas condições do *ReportModel* a partir das quais é gerada a condição de filtro da expressão *SQL*.
Formatting	Formatação aplicada ao controlo. • **BackgroundColor**: cor de fundo do controlo. • **Bold**: se verdadeiro, o texto no controlo fica a negrito.

	• **ForegroundColor**: cor do texto. • **Italic**: se verdadeiro, o texto fica a itálico. • **Underline**: se verdadeiro, o texto fica sublinhado.

Tabela 4.3 - Formatação condicional

Uma condição de formatação pode ser aplicada a qualquer um dos controlos do *ReportModel,* podendo a respectiva condição, à semelhança das condições do próprio *ReportModel,* ser feita sob qualquer campo. Por exemplo, num *ReportModel* podem ser declaradas duas *ReportColumns*, o nome do utilizador e a data de criação, sendo aplicado uma formatação condicional à coluna do nome em que a condição é feita sob a data de criação. Na Fig 4.8 é apresentado o resultado deste exemplo, onde é aplicado o vermelho como cor de fundo para a coluna dos nomes onde a data de criação é inferior a 01-01-2010:



Users

Name	Date
Steven	2010-10-08
Laura	2008-12-05
Michael	2011-03-15
Administrator	2010-01-01
Janet	2011-03-11
Andrew	2011-08-10
Margaret	2009-05-26
Nancy	2011-05-09
Robert	2010-09-21
Anne	2009-09-02

Fig 4.8 - Exemplo: Formatação condicional

4.4 Templates

O conceito principal dos templates é proporcionar uma forma flexível de configurar um *report*, possibilitando que todos os *reports* de um produto partilhem as mesmas configurações. Estas configurações estão relacionadas com formatações e *layout* dos campos gerados no *report*. Um exemplo básico pode ser a formatação do título, sendo possível definir no template a fonte, a cor, tamanho e posição do mesmo. Desta forma, quando se alterar determinada configuração num template, todos os *reports* de um produto que sejam impressos com base neste template vão automaticamente reflectir essas alterações.

Para além de formatações, é também possível definir no template controlos para serem adicionados ao *report* pelo componente de geração. Por exemplo, é possível definir no template um controlo do tipo imagem que representa o logótipo da empresa, definindo propriedades como a posição e o tamanho da mesma. Significa isto que o componente de geração adiciona ao *report* controlos com base na definição do *ReportModel* e também nos controlos definidos no template.

Na Fig 4.9 é apresentado o modelo de dados do template:

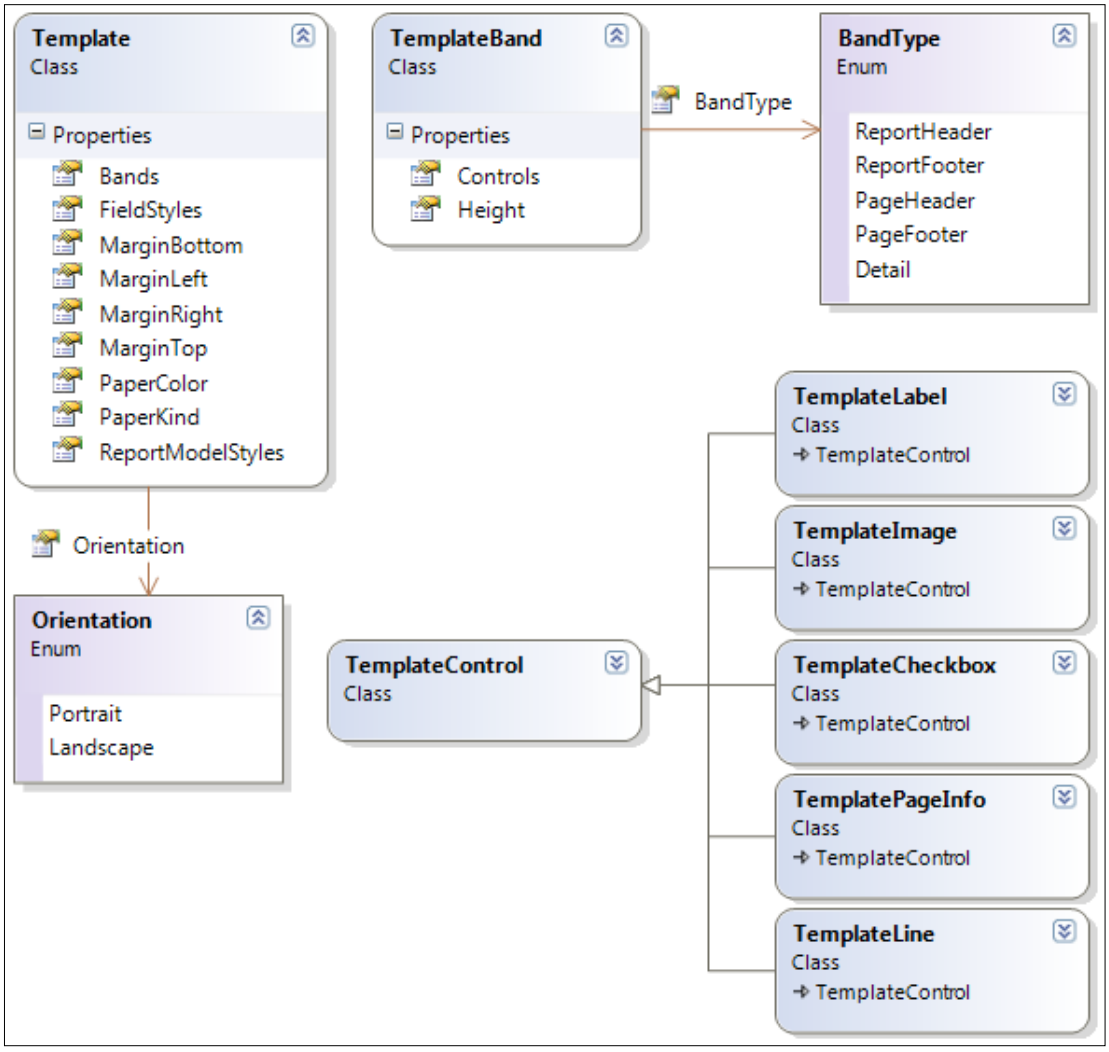


Fig 4.9 - Template

A principal classe deste modelo é a classe ***Template***. Nesta classe são definidas configurações como as margens do *report*, a orientação da página, a cor de fundo, os estilos dos campos e ainda outros controlos a adicionar ao *report* numa determinada secção. Estes controlos são definidos numa ***TemplateBand***, que é depois traduzida numa secção do *report* quando o template for processado pela Framework de Reporting. Significa isto que um controlo definido por exemplo numa *TemplateBand* do tipo (***BandType***) *PageHeader*, irá ser introduzido no cabeçalho do *report*. Cada um destes controlos tem como base a classe ***TemplateControl***, na qual estão definidas as propriedades comuns a todos eles, como a posição, o tamanho ou estilo a aplicar.

Na Tabela 4.4 é apresentada a descrição de cada um dos componentes que fazem parte do modelo de Templates:

Componente	Descrição
Template	Classe que define um template.
TemplateBand	Representa uma secção no *report*, na qual pode ser definido um conjunto de controlos. Os controlos definidos numa secção são adicionados ao *report* na respectiva secção do *report* de acordo com a ***BandType***.

BandType	Tipo da secção. • **ReportHeader**: cabeçalho do *report*. • **ReportFooter**: rodapé do *report*. • **PageHeader**: cabeçalho da página. • **PageFooter**: radapé da página. • **Detail**: detalhe do *report*.
Orientation	Define a orientação da página do *report*, podendo ser do tipo ***Portrait*** e ***Landscape***.
TemplateControl	Classe base para os controlos que podem ser definidos no template. Estes controlos são adicionados numa **TemplateBand**, podendo ser definidas propriedades como a posição na respectiva secção o e tamanho do controlo.
TemplateLabel	Representa um campo de texto.
TemplateImage	Classe que representa uma imagem que pode ser definida no template. Pode ser utilizada por exemplo para definir o logótipo dos *reports*.
TemplateCheckbox	Representa uma caixa de selecção (*ckeckbox*).
TemplatePageInfo	Representa a data e a numeração das páginas.
TemplateLine	Define uma linha.

Tabela 4.4 - Template

Como vimos, o template permite definir controlos para serem adicionados ao *report* pelo componente de geração. O exemplo a seguir mostra dois controlos que podem ser definidos no template, que são neste caso uma imagem com o logótipo da empresa e ainda um controlo to tipo *TemplatePageInfo* para apresentar a data e a paginação.

```
<Template xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Bands>
    <TemplateBand Id="PageHeader">
      <Controls>
        <TemplateControl xsi:type="TemplatePageInfo" Id="infoPageNumber">
          <Left>0</Left>
          <Top>0</Top>
          <PageInfoType>PageNumberDate</PageInfoType>
          <Style>style_infoPageNumber</Style>
        </TemplateControl>
        <TemplateControl xsi:type="TemplateImage" Id="logo">
          <Right>0</Right>
          <Top>0</Top>
          <Path>C:\Images\PrimaveraLogo.png</Path>
          <ImageSizeMode>AutoSize</ImageSizeMode>
        </TemplateControl>
      </Controls>
    </TemplateBand>
  </Bands>
  <FieldStyles>
    <FieldStyle Id="style_infoPageNumber">
      <TextAlignment>TopRight</TextAlignment>
      <FontColor>Blue</FontColor>
      <Width>450</Width>
      <Height>35</Height>
    </FieldStyle>
  </FieldStyles>
</Template>
```

O resultado da aplicação deste template pode ser observado na Fig 4.10:

Fig 4.10 - Exemplo: Template

No template é também possível definir estilos para serem aplicados pelo componente de geração aos controlos gerados quer a partir da definição do *ReportModel*, quer a partir dos controlos definidos no próprio template. Na Fig 4.11 é apresentado o modelo de classes dos estilos que podem ser definidos no template:

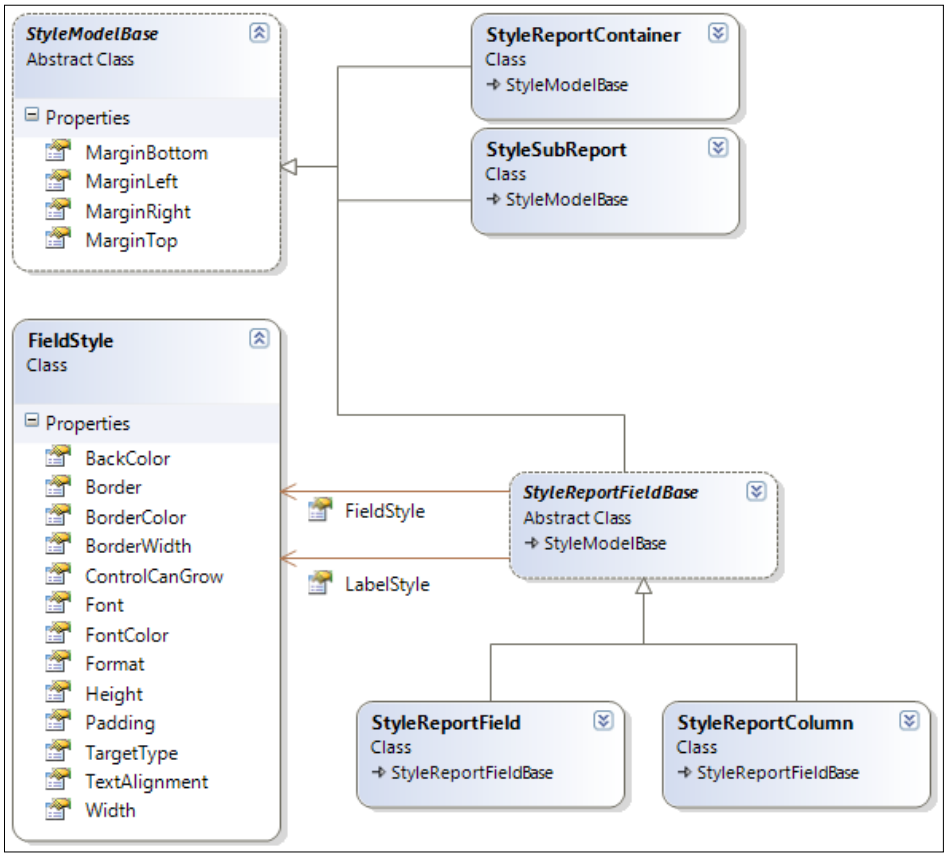


Fig 4.11 - Template: Estilos

Na Tabela 4.5 é apresentada a descrição de cada um dos componentes que fazem parte do modelo de classes dos estilos que podem ser definidos no template:

Componente	Descrição
FieldStyle	Define o estilo que é aplicado a um controlo. • **BackColor**: define a cor de fundo. • **Border**: define que *borders* são aplicadas ao controlo. • **BorderColor**: cor da *border.* • **BorderWidth**: largura da linha da *border.* • **ControlCanGrow**: define se o tamanho do controlo pode aumentar quando o tamanho do seu conteúdo é superior ao do próprio controlo.

	• **Font**: tipo de fonte. • **FontColor**: cor da fonte • **Format**: formação aplicada ao texto do controlo. É utilizado por exemplo para formatar a data (DD/MM/YYYY ou YYYY/MM/DD). • **Height**: altura do controlo. • **Padding**: espaço entre o texto do controlo e cada uma das suas margens. • **TargetType**: corresponde à propriedade **DataType** dos *ReportFieldBase* num *ReportModel.* • **TextAlignment**: define o alinhamento do texto dentro de um controlo. • **Widht**: define a largura do controlo.
StyleModelBase	Classe base que contém as propriedades comuns aos estilos definidos para os controlos do *ReportModel.*
StyleReportContainer	Estilo aplicado ao controlo *ReportContainer.*
StyleSubReport	Aplicado ao controlo *SubReport.*
StyleReportFieldBase	Aplicado ao *ReportFieldBase*, onde podem ser definidos estilos para a descrição do campo e para o valor desse campo.
StyleReportField	Aplicado ao *ReportField.*
StyleReportColumn	Aplicado ao *RpeortColumn.*

Tabela 4.5 - Template: Estilos

Para se perceber melhor de que forma é que os estilos funcionam, e como são aplicados aos controlos do *report*, é apresentado o seguinte template:

```
<Template xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <FieldStyles>
    <FieldStyle Id="style_type_Text" TargetType="Text">
      <Border>None</Border>
      <FontColor>Black</FontColor>
    </FieldStyle>
    <FieldStyle Id="style_type_Boolean" TargetType="Boolean">
      <Border>Bottom</Border>
      <BorderWidth>1</BorderWidth>
      <BorderColor>Blue</BorderColor>
    </FieldStyle>
    <FieldStyle Id="style_type_Date" TargetType="Date">
      <Border>Top</Border>
      <BorderColor>Red</BorderColor>
      <FontColor>Blue</FontColor>
    </FieldStyle>
  </FieldStyles>
  <ReportModelStyles>
    <StyleModelBase Id="reportField" xsi:type="StyleReportField">
      <MarginLeft>50</MarginLeft>
      <MarginRight>10</MarginRight>
      <MarginTop>10</MarginTop>
      <MarginBottom>10</MarginBottom>
      <LabelWidth>200</LabelWidth>
      <LabelFieldSpace>20</LabelFieldSpace>
      <LabelStyle>
        <Border>Bottom</Border>
        <BorderWidth>1</BorderWidth>
        <TextAlignment>TopRight</TextAlignment>
        <FontColor>Black</FontColor>
      </LabelStyle>
    </StyleModelBase>
  </ReportModelStyles>
</Template>
```

Neste template são definidos três estilos nos *FieldStyles* que serão aplicados aos controlos de acordo com o *DomainType* dos campos no *ReportModel*. O primeiro estilo definido é para ser aplicado nos campos do tipo texto (***Text***), sendo aplicado como cor da fonte o preto. O segundo estilo vai ser aplicado aos tipos booleanos (***Boolean***), onde está definido que deve ser aplicado uma *border* na parte inferior do controlo, sendo a sua espessura de 1mm e de cor azul. Por último temos um estilo para os tipos data (***Date***), onde está definida uma *border* na parte superior do controlo com a cor vermelho, e a cor azul para ser aplicada ao texto.

Nos estilos para serem aplicados aos controlos do *ReportModel*, temos definido um estilo do tipo ***StyleReportField*** que é aplicado aos controlos do tipo *ReportField*. Nele estão definidas margens, o tamanho da descrição do campo e também o espaço entre a descrição e o valor do campo. Está ainda definido o estilo específico para ser aplicado à descrição do campo (***LabelStyle***), e neste exemplo está definido para a descrição uma *border* na parte inferior com espessura de 1mm, e ao texto é aplicada a cor preta sendo alinhado ao canto superior direito.

O resultado das formatações definidas neste template é apresentado no seguinte *report* (Fig 4.12):

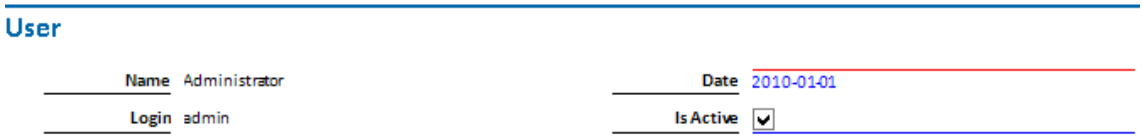


Fig 4.12 - Exemplo: Definição de estilos no template

No exemplo anterior podemos verificar que o template está definido num ficheiro XML, significa isto que um template pode ser serializado para XML e guardado num ficheiro em disco ou base de dados para posteriormente ser carregado e utilizado pela Framework de Reporting na geração do *report*.

É possível definir uma hierarquia de templates, isto é, dizer para determinado template que tem por base um outro template. Para a concretização desta funcionalidade, foi necessário desenvolver um componente de leitura de templates, que permite identificar se um template tem um template base, e caso se verifique tal, herdar todas as configuração da base que não estão definidas.

No exemplo abaixo (Fig 4.13), podemos observar que é possível ter um template base (Master Template) com configurações gerais a vários templates, e num segundo nível vários templates que herdam essas configurações.

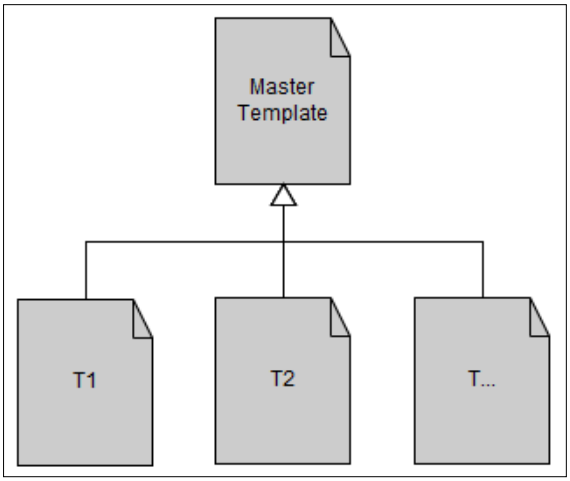


Fig 4.13 - Hierarquia de templates

Esta possibilidade de herança entre templates tem a grande vantagem de se puderem definir configurações gerais para todos os *reports* de uma aplicação, sendo depois criados templates com características especificas a aplicar a um *report* ou conjunto de *reports* que herdam estas configurações gerais.

4.5 Componente de geração

O componente de geração é responsável por gerar os *reports* a partir de um *ReportModel*, aplicando determinas configurações definidas no contexto de impressão e formatações especificadas num template. De uma forma simplista, pode dizer-se que este componente recebe como parâmetros um *ReportModel* e um template, e dá origem a um *report*. A sua execução é realizada pelo motor de impressão, que apenas conhece a interface *IReportGenerator*, sendo por isso obrigatório a sua implementação por parte do gerador. Isto significa que este componente é completamente independente dos restantes componentes da Framework de Reporting, podendo por isso ter uma arquitectura própria. Outra vantagem nesta forma de implementação, está relacionada com o facto do componente de geração poder ser facilmente substituído, uma vez que o motor de impressão o executa pela interface *IReportGenerator* e não pela implementação concreta do componente. A Fig 4.14 apresenta o componente de geração de *reports:*

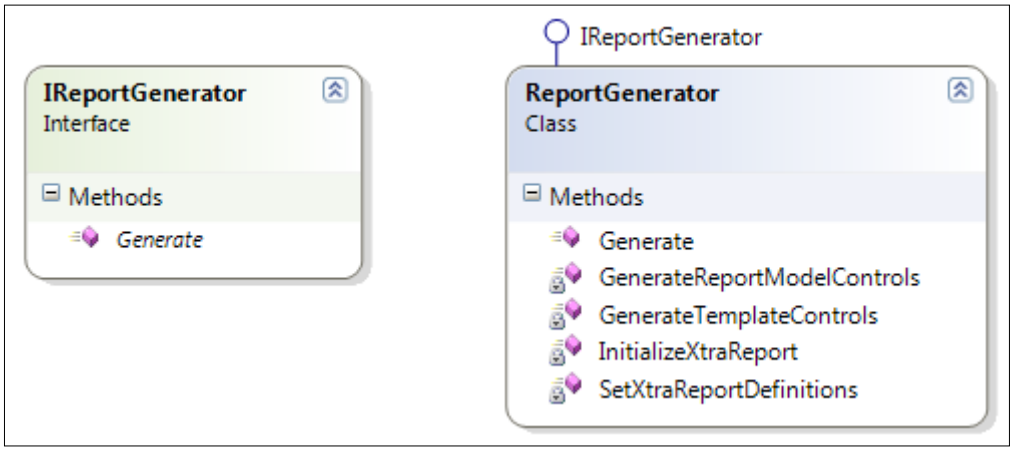


Fig 4.14 - Report Generator

Como já foi referido anteriormente, a ferramenta de *reporting* seleccionada foi o ***DevExpress XtraReports***, logo, o componente de geração da Framework de Reporting vai gerar os *reports* para serem lidos por esta ferramenta, aos quais vamos chamar apenas de ***XtraReports***.

Na imagem anterior (Fig 4.14) podemos observar as funções principais definidas no *ReportGenerator*. Estas funções definem o algoritmo executado internamente por este componente quando o motor de impressão executa a função *Generate*. A seguir é apresentada uma descrição simples e a ordem pela qual estas funções são executas.

1. **InitializeXtraReport**: inicializa o *XtraReport* com as definições base, como por exemplo a conexão à base de dados e a expressão SQL a ser executada.
2. **SetXtraReportDefinitions**: define propriedades do *report* com base no template, como por exemplo as margens ou a orientação da página.
3. **GenerateTemplateControls**: adiciona ao *report* os controlos definidos no template, como por exemplo uma imagem com o logótipo da empresa.
4. **GenerateReportModelControls**: geração dos controlos a partir do *ReportModel*.

As partes mais complexas do componente de geração são exactamente aquelas que estão relacionadas com a geração dos controlos, quer a partir do template quer a partir do *ReportModel*. Quer o template quer o *ReportModel*, foram desenhados para que fossem independentes que qualquer outro componente. Significa isto que não é possível fazer uma conversão directa a partir de qualquer um destes para o modelo de dados de um *XtraReport*. A um *XtraReport* podem ser adicionados controlos de vários tipos: texto, imagens, *containers*, sendo necessário definir para cada um deles o tamanho e a respectiva posição. É neste ponto que diferem os controlos definidos no templo relativamente aos que são definidos no *ReportModel*, sendo que para o caso do template, os controlos já tem o seu posicionamento e tamanho definido. No caso do *ReportModel*, o posicionamento e tamanho de cada controlo tem de ser calculado, pois depende da própria definição do *ReportModel*.

O tamanho de um controlo que é gerado a partir de um campo do *ReportModel* é, como já vimos anteriormente, definido num estilo do *Template* de acordo com o seu *DomainType*, sendo depois ajustado de acordo ao tamanho disponível no *report*. Já o tamanho de um controlo que representa um *ReportContainer* é calculado somando o tamanho dos seus filhos e das margens entre eles. No caso dos *SubReports* não existe a necessidade de calcular o seu tamanho uma vez que quando este é adicionado ao *XtraReport* o espaço por ele ocupado apenas é calculado aquando do seu processamento, sendo que todos os controlos definidos a seguir ao *SubReport* vão ser apresentados correctamente depois do *SubReport* independentemente do espaço que este venha a ocupar.

Para efectuar todos estes cálculos de tamanho e posicionamento dos controlos, recorreu-se a um modelo intermédio para facilitar toda a lógica de geração do *report*. A este modelo deu-se o nome de ***Report Visual Tree***, e que basicamente se define como sendo o conjunto de elementos que vão ser adicionados ao *XtraReport* sendo calculada a informação de posicionamento e tamanho de cada controlo. Este modelo é apresentado na Fig 4.15:

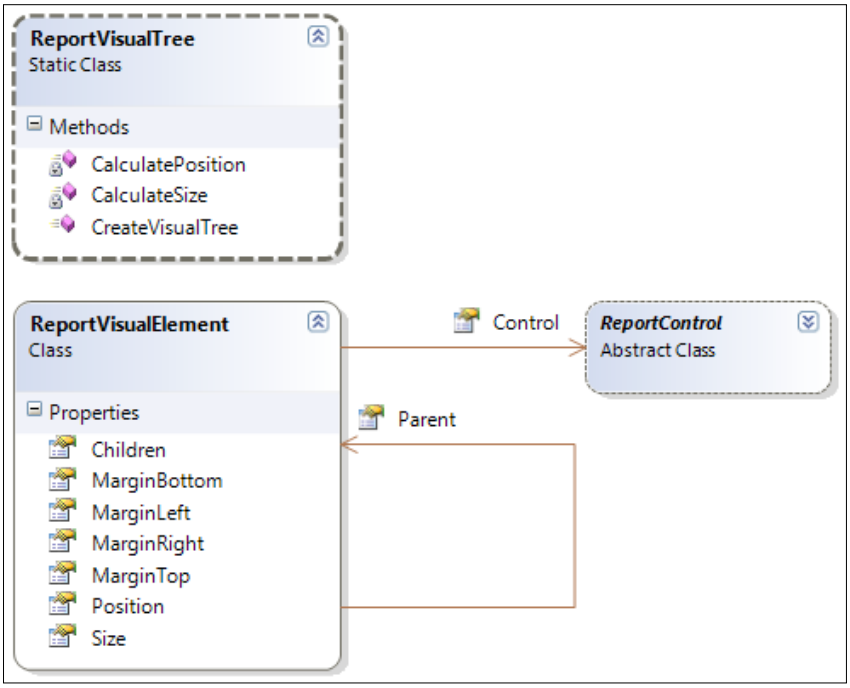


Fig 4.15 - Report Visual Tree

Este modelo em forma de árvore é criado com base no *ReportModel*, sendo calculados os tamanhos, posições e margens de cada controlo também com base nas definições do template associado para a geração do *report*. Cada *ReportControl* definido no *ReportModel* vai dar origem a um *ReportVisualElement* na *Report Visual Tree*, sendo que no caso dos *ReportContainers*, este elemento vai conter uma lista dos respectivos filhos, contendo cada um deles uma referência para o elemento pai. Como exemplo, analisemos o seguinte *ReportModel*:

```
ReportModel model = new ReportModel();

/* Root Container */
ReportContainer root = new ReportContainer();
root.Orientation = Orientation.Horizontal;
model.Detail.Controls.Add(root);

/* Left Container */
ReportContainer left = new ReportContainer();
left.Orientation = Orientation.Vertical;
root.Controls.Add(left);

ReportField f1 = new ReportField();
ReportField f2 = new ReportField();
ReportField f3 = new ReportField();
left.Controls.Add(f1);
left.Controls.Add(f2);
left.Controls.Add(f3);

/* Right Container */
ReportContainer right = new ReportContainer();
right.Orientation = Orientation.Vertical;
root.Controls.Add(right);
ReportField f4 = new ReportField();
ReportField f5 = new ReportField();

right.Controls.Add(f4);
right.Controls.Add(f5);
```

Neste exemplo temos um *ReportModel* que define um *container root*, que por sua vez contém dois *containers* filhos com orientação horizontal. No primeiro (*Left*) estão definidos três campos, e no segundo (*Right*) estão definidos dois. Quer num quer noutro, os campos ficam orientados verticalmente. A *Report Visual Tree* que resulta deste *ReportModel* é apresentada na Fig 4.16:

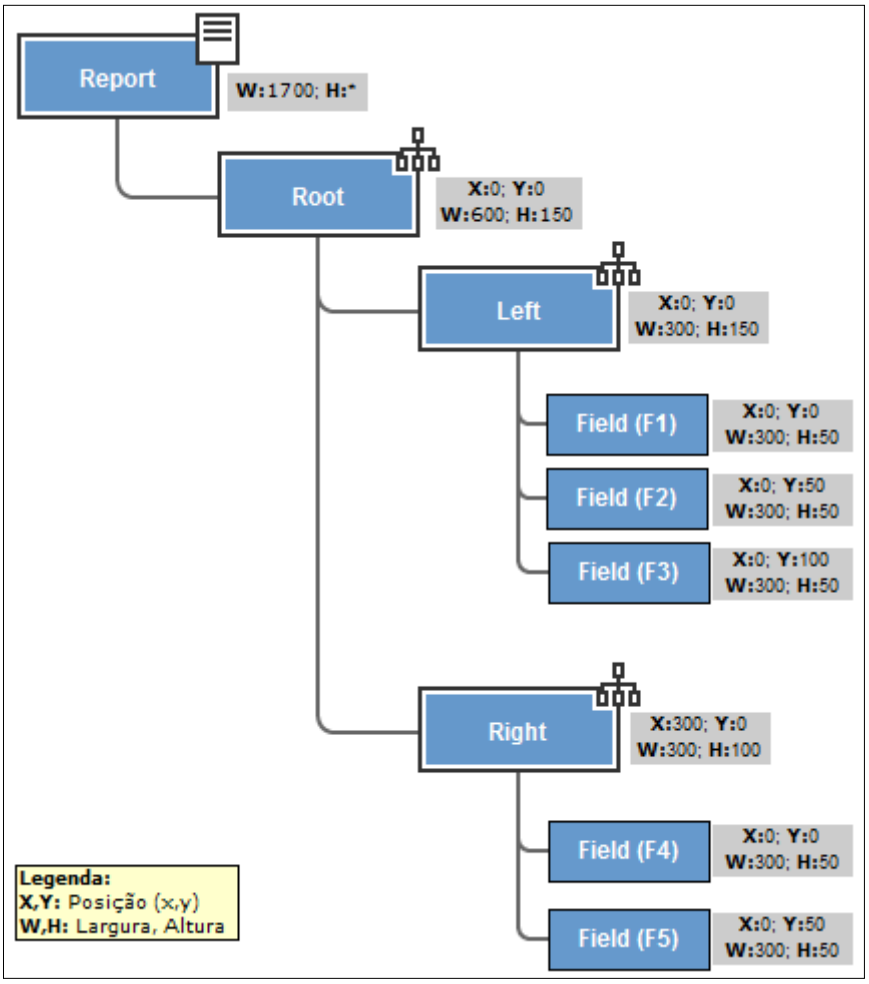


Fig 4.16 - Exemplo: Report Visual Tree

Na Fig 4.16 podemos então observar a *Report Visual Tree* criada para o *ReportModel* apresentado. Temos como raiz desta arvore o próprio *report*, no qual está definido uma largura de página de 1700 milímetros, que é calculada a partir da largura total de página menos a largura das margens esquerda e direita definidas no template. Relativamente à altura do *report*, esta é teoricamente infinita, pois os controlos são automaticamente colocados na página seguinte. Para os restantes nós da árvore (*containers* e *campos*), é necessário efectuar o cálculo do tamanho e do posicionamento de cada um, exactamente por esta ordem.

O cálculo do tamanho é efectuado em dois passos distintos. Em primeiro lugar o tamanho é calculado dos nós mais exteriores para a raiz, isto é, dos campos (*fields*) para o *report*, sendo que o tamanho inicial atribuído a cada campo está definido no template para cada *domain type* a que cada campo corresponde. Neste primeiro passo o tamanho dos *containers* é calculado somando o tamanho dos seus filhos e ainda as margens entre eles definidas no template, e tendo em conta a orientação definida. Terminado este primeiro passo, e

dependendo da profundidade da árvore e do tamanho definido para os campos no template, é provável que a largura disponível da página (neste caso 1700 milímetros) seja ultrapassada. Daí a necessidade de um segundo passo no cálculo do tamanho de cada controlo, mas desta vez é efectuada da raiz para os elementos mais exteriores. Então para cada nó da árvore, se a sua largura for superior ao espaço disponível do seu pai, esta é ajustada. No exemplo apresentado, este segundo passo não tem qualquer efeito, uma vez que a profundidade da árvore é de apenas três níveis e o tamanho definido no template para os campos é relativamente pequeno.

Depois de calculado o tamanho dos elementos do *report*, é a vez de ser calculado o posicionamento de cada um. Este cálculo é relativamente mais simples quando comparado como o cálculo do tamanho dos elementos, uma vez que apenas está dependente do tamanho de cada um, e no caso dos *containers* da orientação dos seus filhos. Desta forma, o cálculo é efectuado da raiz para os elementos mais exteriores.

Agora que a *Report Visual Tree* está criada e com todos os cálculos já concluídos, resta apenas gerar os controlos no *XtraReports*. A geração dos controlos é efectuada percorrendo todos os elementos da árvore da raiz para os elementos exteriores. Então, para cada elemento (*ReportVisualElement*) da árvore é gerado no *XtraReport* um controlo de acordo com o *ReportControl* que lhe está associado, e usando a informação de tamanho e posicionamento do respectivo elemento. A questão que se coloca agora é de que forma é gerado um controlo para adicionar ao *XtraReport* a partir de um *ReportControl*. Os detalhes da geração dos controlos para o *XtraReport* a partir de um *ReportControl* são descritos na secção seguinte.

4.5.1 Geração de controlos

Os componentes de geração de controlos são responsáveis por gerar os controlos para adicionar ao *XtraReport* quer a partir dos que são definidos no template quer no *ReportModel*. Os controlos a gerar podem ser por exemplo um simples campo de texto ou uma imagem com o logótipo da empresa. Por exemplo, no caso do *ReportModel*, se estiver definido um *SubReport*, este vai dar origem a um *subreport* que é gerado pelo respectivo gerador (*SubReportGenerator)*. No caso dos campos definidos no *ReportModel* (*ReportField* e *ReportColumn*), a forma de selecção do gerador depende de tipo (*Domain Type)* do campo, por exemplo, um campo do tipo texto ou numérico vai ser gerado pelo componente de geração de texto e um campo do tipo imagem é gerado pelo gerador de controlos do tipo imagem.

Na Fig 4.17 estão representados os componentes de geração de controlos:

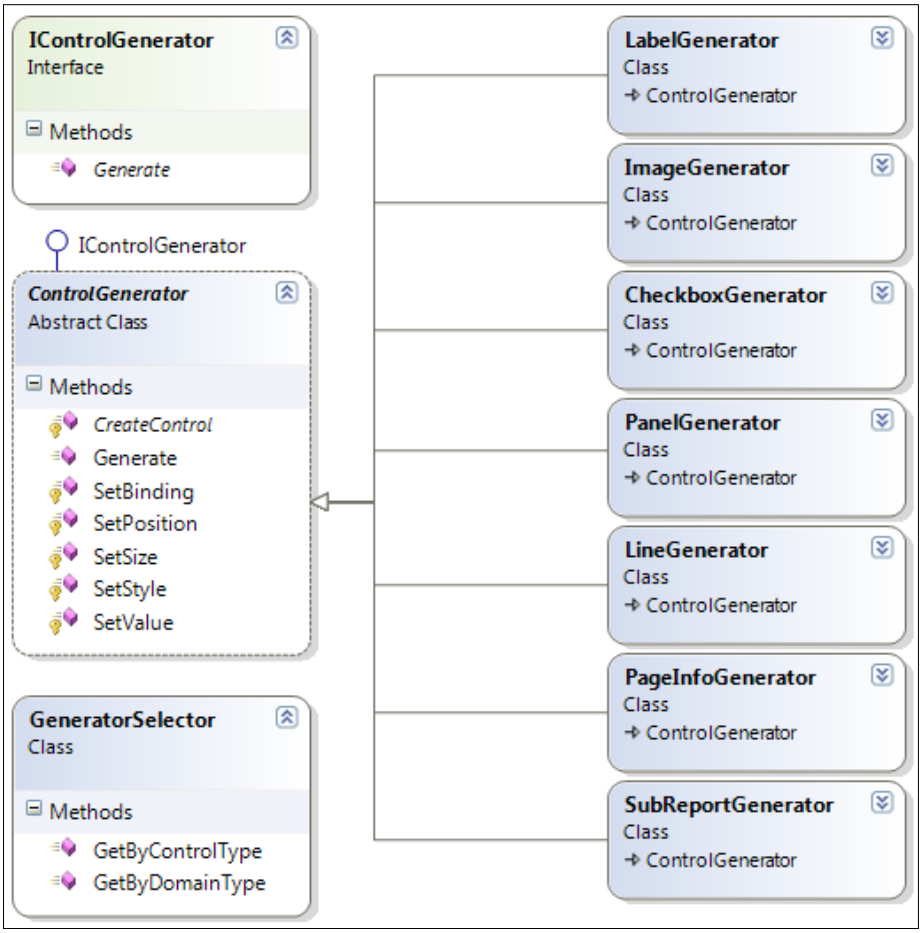


Fig 4.17 - Geradores de controlos

Componente	Descrição
IControlGenerator	Interface que deve ser implementada pelos componentes de geração de controlos.
ControlGenerator	Classe base (abstracta) para os componentes de geração de controlos. Define a lógica de geração comum a todos os gerados.
LabelGenerator	Responsável por gerar controlos do tipo texto. É este o tipo de controlo mais comum num report, sendo utilizado para a maioria dos campos num *ReportModel*.
ImageGenerator	Gera controlos do tipo imagem, como por exemplo o logótipo da empresa no cabeçalho de um report.
CheckboxGenerator	Responsável por gerar controlos do tipo *checkbox*, que representa os valores verdadeiro ou falso.
PanelGenerator	Gera controlos do tipo *panel/container* que tem por objectivo agrupar um conjunto de campos. Este tipo de controlo ter como "filhos" qualquer outro tipo de controlo incluindo outros *containers.*
LineGenerator	Responsável por gerar controlos do tipo linha.
PageInfoGenerator	Responsável por gerar o controlo de paginação num report.
SubReportGenerator	Gera controlos do tipo *subreport*, dando a possibilidade de num report poderem ser definidos report "filhos".
GeneratorSelector	Componente responsável por instanciar o gerador correcto de acordo com o *domain type* de um campo ou de acordo com o tipo de controlo do *ReportModel*.

Tabela 4.6 - Geradores de controlos

Os componentes de geração de controlos são instanciados pelo *ReportGenerator* através da classe *GeneratorSelector*, que o responsável por retornar o gerador correcto quer para os vários *domain types* quer para os *ReportControls* definidos no *ReportModel*. A execução dos geradores de controlos é feita através a *interface IReportGenerator* pelo método *Generate.*

Na Tabela 4.7 é apresentado o respectivo controlo gerado por cada um dos geradores de controlos:

Gerador	Controlo XtraReports
LabelGenerator	XRLabel
ImageGenerator	XRPictureBox
CheckboxGenerator	XRCheckBox
PanelGenerator	XRPanel
LineGenerator	XRLine
PageInfoGenerator	XRPageInfo
SubReportGenerator	XRSubreport

Tabela 4.7 - Geradores de controlos para XtraReports

Todos estes controlos do *XtraReports* têm uma base comum, a classe ***XRControl***, que possui propriedades como a posição do controlo no *report*, o tamanho, a cor de fundo e ainda as *borders* e a sua espessura. Dado isto, faz sentido existir para os geradores de controlos também uma base comum, que processe toda a lógica comum aos controlos. É este o objectivo da classe abstracta *ControlGenerator* ser a base de todos os geradores de controlos implementando toda a lógica de geração comum a todos. A implementação desta classe segue o padrão de desenho *Template Method* (4), definindo um algoritmo em que algumas das funções são redefinidas pelos geradores concretos.

Quando o método *Generate* é invocado, o *ControlGenerator* executa a seguinte sequência de passos:

1. **CreateControl**: método responsável por instanciar o controlo concreto do *XtraReports*.
2. **SetBinding**: associa o controlo a um campo da base de dados, sendo este método executado apenas para o *ReportField* e *ReportColumn*.
3. **SetValue**: atribui o valor ao respectivo controlo, sendo executado quando não está definida uma associação a um campo da base de dados.
4. **SetStyle**: aplica as formatações/configurações ao controlo de acordo com definições de um template.
5. **SetSize**: define o tamanho do controlo.
6. **SetPosition**: define a posição do controlo.

Implementando os geradores de controlos desta forma, criando uma classe base para processar toda a lógica comum, e depois delegar para os geradores concretos a lógica específica de cada tipo de controlo trás várias vantagens. Desde logo, esta divisão em vários componentes permite uma melhor organização do código produzido para cada um, sendo por isso mais fácil a sua manutenção e detecção/correcção de erros de geração. Permite também focar em cada um destes componentes detalhes específicos de cada controlo, tendo também a grande vantagem de poderem ser reutilizados em vários pontos da geração do *report*. É

também com grande facilidade que se podem adicionar novos componentes, quer utilizando como base o *ControlGenerator* ou então implementando a interface *IControlGenerator*.

Como foi referido na secção anterior, os controlos adicionados ao *XtraReport* podem ter duas origens distintas: o template e o *ReportModel* depois de calculada a *Report Visual Tree*. Por isso tornou-se necessário desenvolver um modelo de dados para os geradores que fosse independente de qualquer uma destas origens. A Fig 4.18 apresenta o modelo de dados que é passado como parâmetro para cada um dos geradores de controlos.

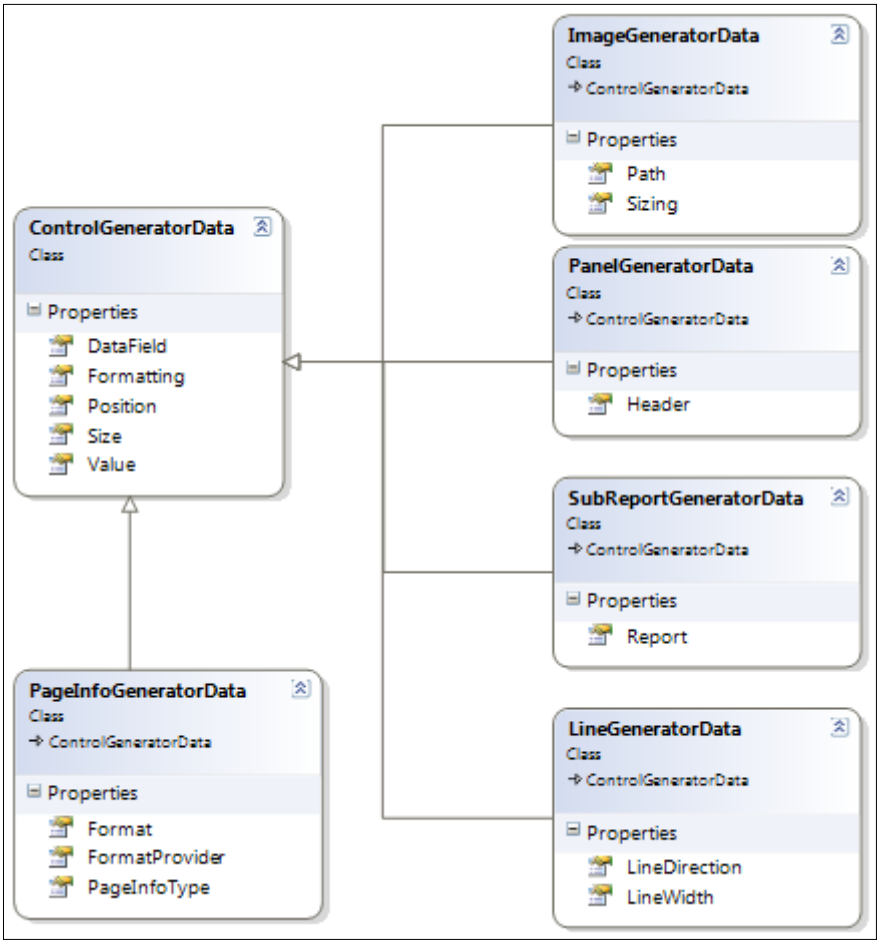


Fig 4.18 - Modelo de dados para os geradores de controlos

Na Tabela 4.8 é apresentada a descrição de cada um dos componentes que fazem parte do modelo apresentado na Fig 4.18.

Componente	Descrição
ControlGeneratorData	Classe base para os dados dos componentes de geração de controlos. Contem propriedades que definem o tamanho (*Size*) e posição (*Position*) do controlo, o valor (*Value*) a ser atribuído ou o respectivo campo numa tabela da base de dados (*DataField*). Contém ainda informação para ser aplicada como formatação condicional (*Formatting*). Esta classe é utilizada como principal em alguns dos geradores específicos, uma vez que a informação nela contida é suficiente para esses geradores. É o caso do *LabelGenerator* e do

	CheckboxGenerator.
ImageGeneratorData	Define as propriedades para o gerador de controlos do tipo imagem, como o tipo de ajuste do tamanho da imagem (*Sizing*), o caminho (*Path*) para uma imagem guardada em disco, ou ainda a própria imagem através da propriedade *Value* herdada do *ControlGeneratorData*.
PanelGeneratorData	Usada pelo gerador de *containers*, onde pode ser definido para além das propriedades da classe base, o título (*Header*) do *container*.
PageInfoGeneratorData	Usada pelo *PageInfoGenerator* e define as propriedades que permitem especificar que informação deve ser apresentada no controlo (data, paginação), e ainda definir o formato da data.
SubReportGeneratorData	Classe que define as propriedades para o gerador de *subreports*. Define a propriedade *Report* que é o *XtraReport* gerado para ser colocado no *subreport*.
LineGeneratorData	Usado pelo *LineGenerator*, definindo a espessura e direcção (vertical/horizontal) da linha.

Tabela 4.8 - Modelo de dados para os geradores de controlos

O *ReportGenerator* instância os geradores de controlos e passa-lhes como parâmetro os dados de input quer estes tenham origem nos controlos definidos no template, quer sejam definidos no *ReportModel*.

4.6 Custom Reports

Como já foi referido anteriormente, existem determinados *reports*, em que devido é sua complexidade quer de layout quer no cálculo de determinados campos, não é possível a sua geração automática. É neste contexto que surgem os *custom reports*, que tem por objectivo permitir a construção de *reports* com a total liberdade que um report designer pode oferecer. A grande vantagem deste tipo de *reports* está na elevada liberdade de personalização dos mesmos. A desvantagem evidente está no elevado custo de desenvolvimento e manutenção comparativamente com os *reports* gerados.

A necessidade de construção de *custom reports* está dependente das necessidades de reporting do produto onde a Framework de Reporting está a ser utilizada. Num produto equivalente ao ERP PRIMAVERA, como já vimos anteriormente, existem *reports* de elevada complexidade, sendo por isso necessário recorrer ao uso de *custom reports*. Nestes casos, os *custom reports* terão necessariamente de ser construídos em tempo de desenvolvimento do produto, sendo disponibilizados no produto final.

Pode também ser requisito de um produto dar ao utilizador final a possibilidade de criar os seus próprios *custom reports*, sendo que neste caso o *Report Designer* tem obrigatoriamente de ser distribuído com o produto final. Isto permitirá ao utilizador final do produto ter o controlo de personalizar todo o tipo de reports do produto, uma funcionalidade que já está presente no ERP PRIMAVERA.

Um *custom report* passa necessariamente por duas fases distintas. A primeira passa pela sua construção ou desenho no Report Designer, e a segunda pela sua aplicação na Framework de Reporting, dando origem a um documento que o utilizador final tem acesso.

4.6.1 Report Designer

A Fig 4.19 mostra o exemplo da construção de um report, neste caso de uma factura.

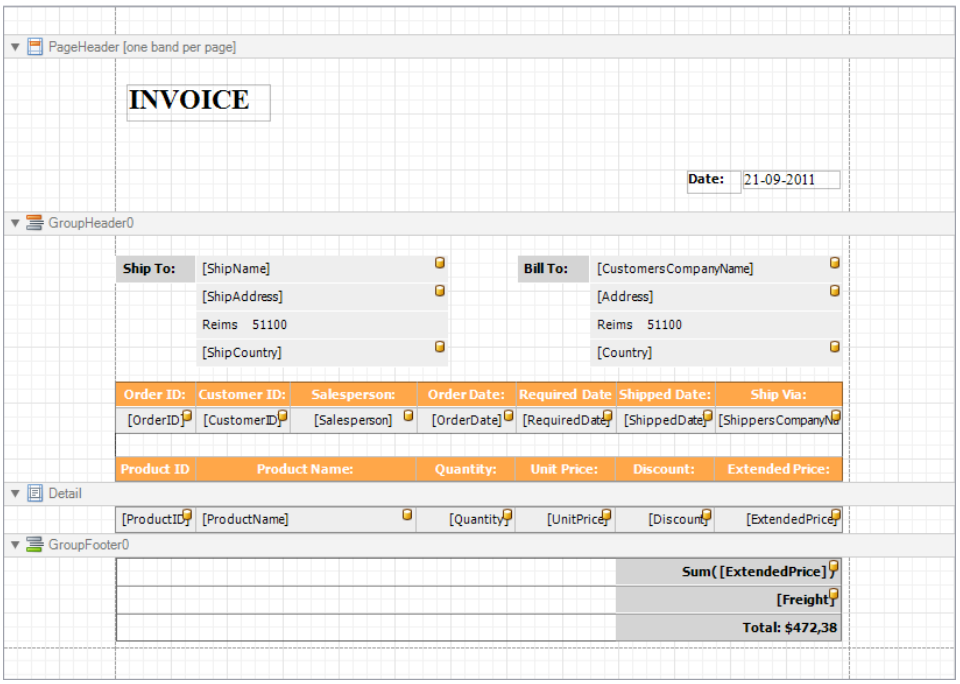
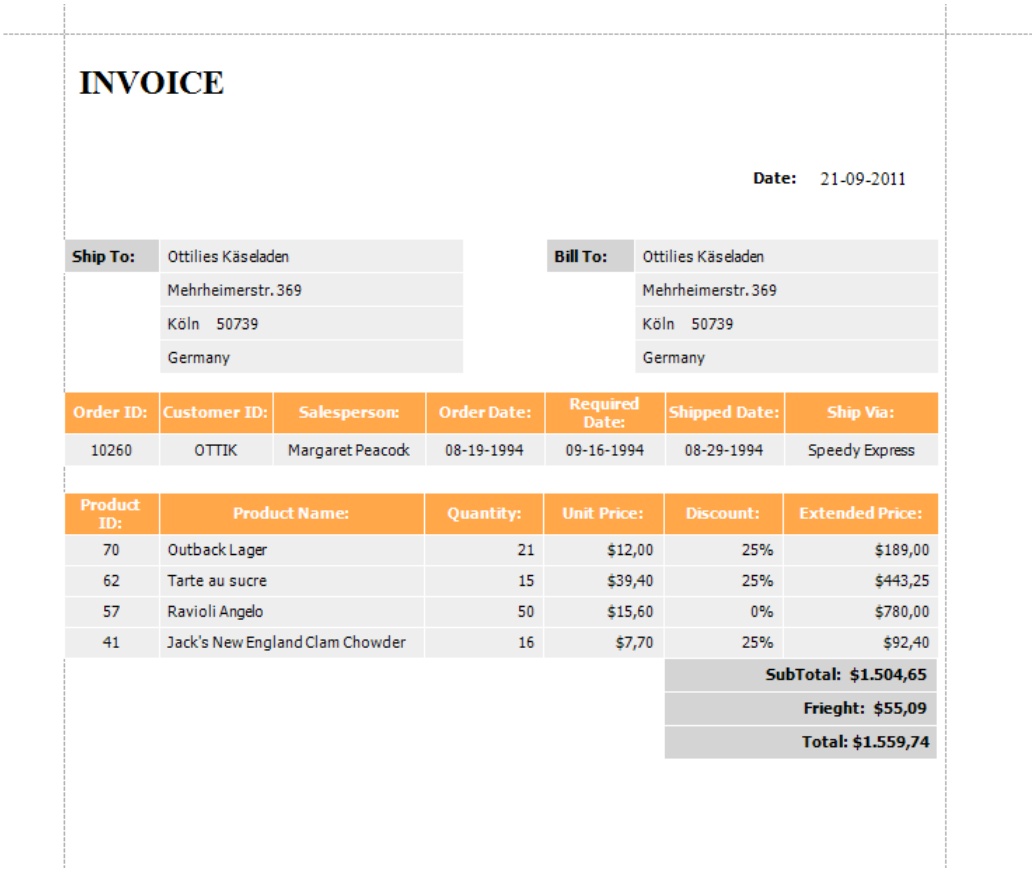


Fig 4.19 - Report Designer (Factura)

O resultado pode ser visto na Fig 4.20:



INVOICE

Date: 21-09-2011

Ship To:	Bill To:
Ottilies Käseladen	Ottilies Käseladen
Mehrheimerstr. 369	Mehrheimerstr. 369
Köln 50739	Köln 50739
Germany	Germany

Order ID:	Customer ID:	Salesperson:	Order Date:	Required Date:	Shipped Date:	Ship Via:
10260	OTTIK	Margaret Peacock	08-19-1994	09-16-1994	08-29-1994	Speedy Express

Product ID:	Product Name:	Quantity:	Unit Price:	Discount:	Extended Price:
70	Outback Lager	21	$12,00	25%	$189,00
62	Tarte au sucre	15	$39,40	25%	$443,25
57	Ravioli Angelo	50	$15,60	0%	$780,00
41	Jack's New England Clam Chowder	16	$7,70	25%	$92,40
				SubTotal:	$1.504,65
				Frieght:	$55,09
				Total:	$1.559,74

Fig 4.20 - Preview (Factura)

4.6.2 Geração de Custom Reports

A principal desvantagem no uso de *custom reports* é o elevado custo de desenvolvimento e manutenção. Dado isto, tornou-se necessário incluir na Framework de Reporting componentes capazes de aplicar algum tipo de tratamento nestes *reports*, fazendo com que fosse possível pelo menos minimizar de alguma forma estes custos, adicionando, como nos *reports* gerados automaticamente, algum tipo de configuração, adicionando alguma flexibilidade aos *custom reports*. Assim como nos *reports* gerados, a estratégia seguida foi a de utilizar os templates para definir as configurações para este tipo de reports.

A Framework de Reporting utiliza informação do contexto de impressão para fazer algumas parametrizações no *report* original. Um exemplo deste comportamento, é a possibilidade de quer o cabeçalho quer o rodapé do report poderem ser adicionados em tempo de execução.

Um *custom report* é então processado pela Framework de Reporting passando pelas mesmas etapas de geração de um *report* gerado automaticamente. A diferença para os *reports* gerados automaticamente, é que a secção dos detalhes (dos dados) é ignorada no processamento do *custom report*, não sendo aplicada qualquer tipo de alteração, uma vez que esta é a parte que realmente é personalizada no Report Desinger. Significa isto, que apesar de ser possível o desenho de um report na sua totalidade (editando todas as suas secções), uma vez que a Framework de Reporting é capaz de fazer personalizações com base num template, a única secção que deve realmente ser desenhada é o detalhe. Aplicando esta regra ao exemplo da factura apresentado anteriormente [Fig 4.19], este pode agora ser desenhado da forma apresentada na Fig 4.21:

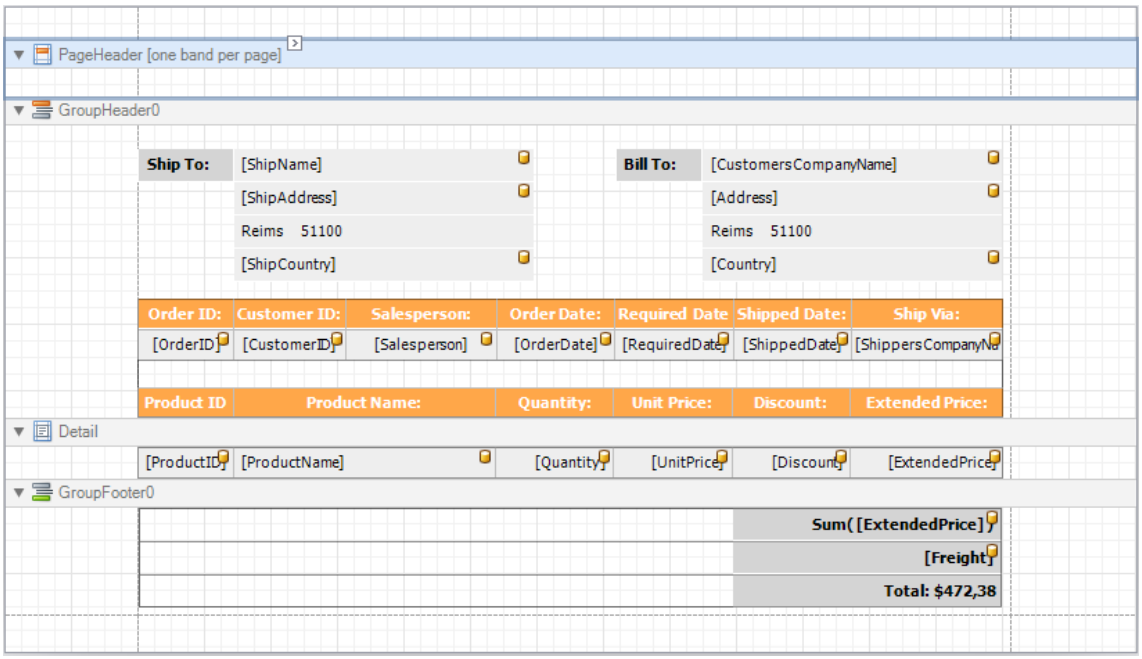
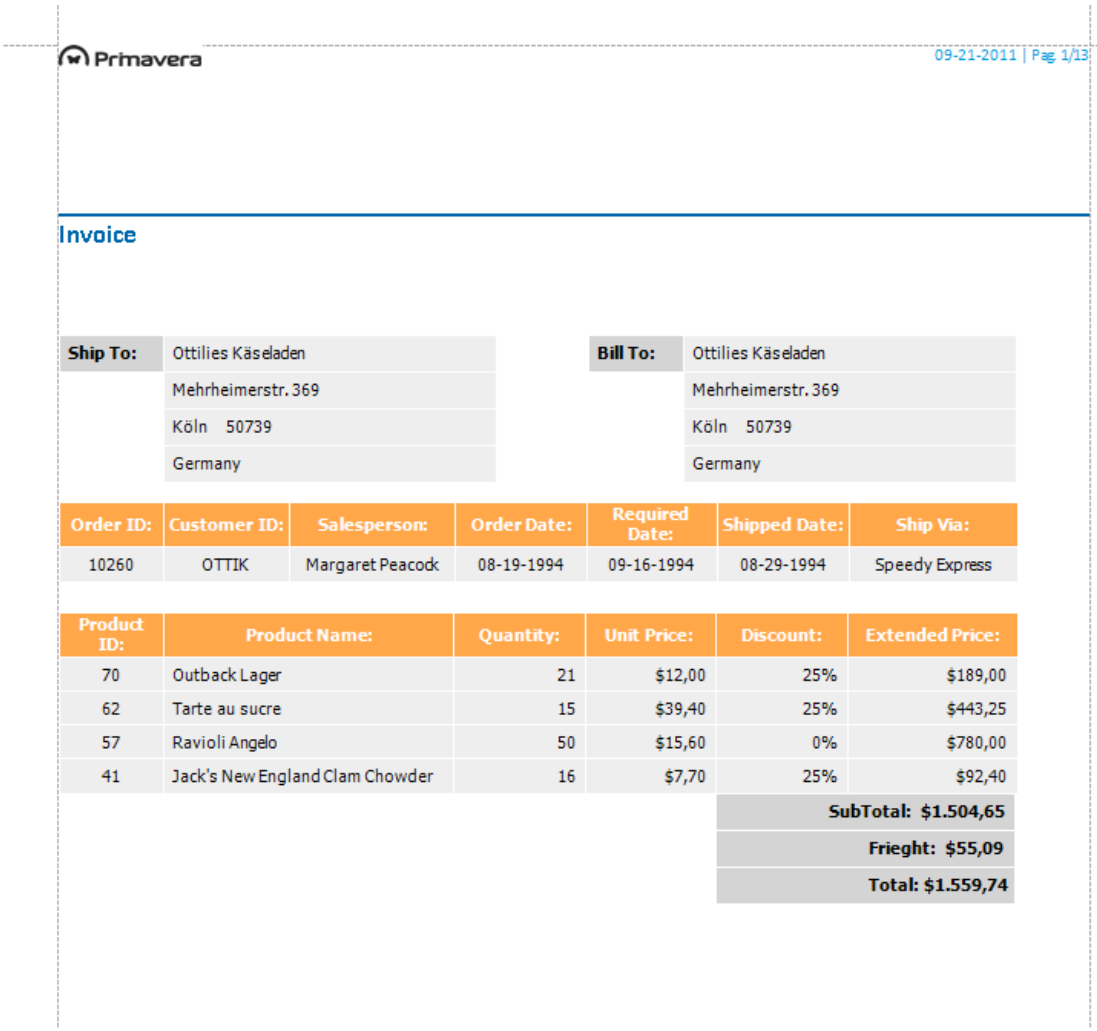


Fig 4.21 - Report Designer (Factura 2)

De notar que neste caso, no *custom report* não foi incluído qualquer tipo de cabeçalho. O resultado do processamento do *custom report* pelo Framework de Reporting, tendo sido aplicado o template apresentado anteriormente, é apresentado na Fig 4.22:



Primavera

09-21-2011 | Pag. 1/13

Invoice

Ship To:	Ottilies Käseladen	**Bill To:**	Ottilies Käseladen
	Mehrheimerstr. 369		Mehrheimerstr. 369
	Köln 50739		Köln 50739
	Germany		Germany

Order ID:	Customer ID:	Salesperson:	Order Date:	Required Date:	Shipped Date:	Ship Via:
10260	OTTIK	Margaret Peacock	08-19-1994	09-16-1994	08-29-1994	Speedy Express

Product ID:	Product Name:	Quantity:	Unit Price:	Discount:	Extended Price:
70	Outback Lager	21	$12,00	25%	$189,00
62	Tarte au sucre	15	$39,40	25%	$443,25
57	Ravioli Angelo	50	$15,60	0%	$780,00
41	Jack's New England Clam Chowder	16	$7,70	25%	$92,40
				SubTotal:	**$1.504,65**
				Frieght:	**$55,09**
				Total:	**$1.559,74**

Fig 4.22 - Preview (Factura 2)

Como podemos observar, o cabeçalho foi gerado com base no Printing Context (onde se inclui o template).

Esta capacidade de manipulação dos *custom reports* por parte da Framework de Reporting, tem como grande vantagem a uniformização de todos os *reports* de um produto com base no mesmo template.

Capítulo 5

5 Módulo de impressão

O módulo de impressão disponibiliza as funcionalizadas de impressão da Framework de Reporting a um produto desenvolvido sob a tecnologia Athena. Neste módulo ficarão todos os dados essenciais à impressão de documentos, como por exemplo os templates, os *custom reports* e ainda o ***Report Model*** das entidades, e que serão utilizados pela Framework de Reporting para gerar os respectivos *reports*.

Os módulos construídos usando a Framework Athena seguem uma arquitectura *multi-tier* (Fig 5.1), possibilitando desta forma dividir a aplicação em camadas lógicas distintas.

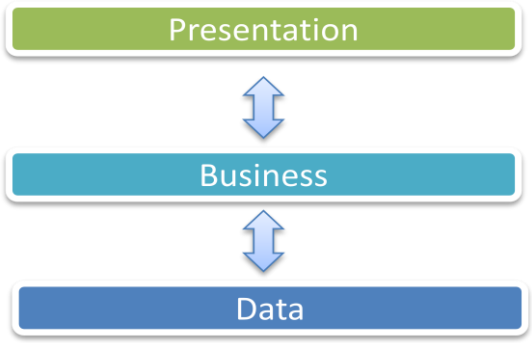


Fig 5.1 - Arquitectura multi-tier

- ***Apresentação***: esta camada fornece a interface de utilizador da aplicação.
- ***Negócio***: implementa as funcionalidades de negócio da aplicação, servindo de intermediário entre as camadas de apresentação e de dados. A camada de apresentação nunca tem acesso directo aos dados.
- ***Dados***: esta camada é responsável pelos dados da aplicação, tratando por exemplo da sua persistência em base de dados.

Este tipo de arquitectura permite que sejam feitas modificações ao nível de cada camada sem que estas tenham qualquer impacto nas restantes camadas. Isto é conseguido uma vez que a comunicação entre camadas é feita através de *interfaces*, e se as alterações não ocorrerem a este nível, então cada uma das camadas é independente das restantes.

O módulo de impressão encapsula a Framework de Reporting, sendo as funcionalidades de impressão disponibilizadas como um serviço do módulo. Desta forma a Framework de Reporting não está directamente acessível, sendo por isso possível a sua substituição sem que isso afecte a implementação da lógica de impressão nos restantes módulos. Para a construção do módulo de impressão foram utilizadas ferramentas disponibilizadas pela Framework Athena, como o ***Entities Designer*** e o ***Presentation Designer***. A utilização de cada uma destas ferramentas é contextualizada de forma detalhada nas secções seguintes.

5.1 Modelo

Nesta secção é detalhado o modelo de entidades definido para o módulo de impressão. As entidades necessárias são modeladas recorrendo ao ***Entities Designer***, a partir do qual é gerado todo o código necessário para a manipulação das mesmas, como por exemplo os serviços de CRUD. A Fig 5.2 apresenta o modelo de entidades desenhado para o módulo de impressão:

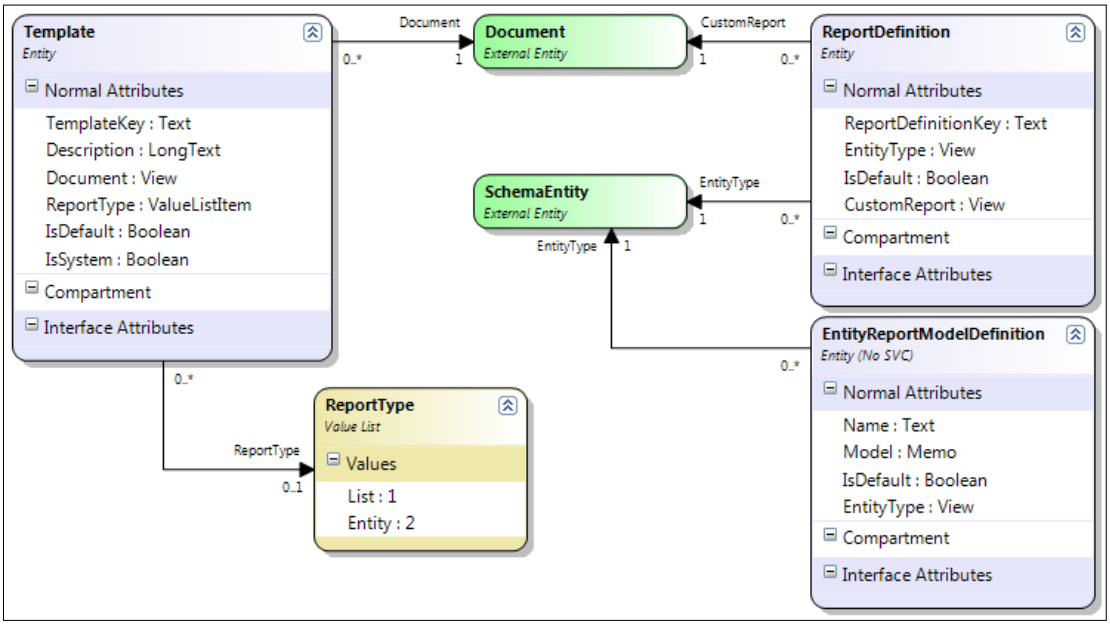


Fig 5.2 - Modelo de classes do módulo de impressão

Na Tabela 5.1 é apresentada a descrição de cada um dos componentes que fazem parte do modelo de entidades do módulo de impressão.

Componente	Descrição
Template	Representa a definição do template para um determinado tipo de *report*. O template propriamente dito, isto é, o ficheiro xml fica na tabela de documentos. Pode ser definido o tipo de *report* (***ReportType***) para qual o template vai estar disponível, listas ou entidades.
ReportDefinition	Representa a definição de um *custom report* para uma determinada entidade (***EntityType***), sendo possível criar para cada tipo de entidade vários *custom reports.* Mais uma vez, é utilizada a tabela de documentos para guardar o ficheiro do *custom report*.
EntityReportModelDefinition	Define os *ReportModels* para as entidades de um produto. O ReportModel está serializado em formato xml na coluna ***Model***, e a associação para a tabela ***SchemaEntity*** define a que entidade pertence o modelo. Também aqui podem ser definidos vários *ReportModels* para a mesma entidade.
ReportType	Representa o tipo de *report* para o qual está definido o template, listas ou entidades. Apesar de na prática um template poder ser utilizado tanto para entidades como para listas, esta distinção serve apenas para se poder agrupar os templates pelo contexto em que podem ser utilizados.

Document	Entidade externa ao módulo de impressão, disponibilizada por um dos módulos core da Framework Athena, que tem como objectivo guardar documentos como os templates e *custom reports.*
SchemaEntity	Entidade externa ao módulo de impressão, disponibilizada por um dos módulos core da Framework Athena, que contem todas as entidades de um produto.

Tabela 5.1 - Modelo de classes do módulo de impressão

Com este modelo é então possível definir vários templates no módulo de impressão, quer para listas quer para entidades. Para a impressão de entidades, os respectivos *ReportModels* são definidos na tabela *EntityReportModelDefinition* sendo também possível definir mais do que um para cada tipo de entidade. Relativamente aos *custom reports,* estes ficam definidos na tabela *ReportDefinitions*, sendo possível no limite ter um *custom report* para cada entidade do produto.

5.2 Serviço de impressão

Como foi já referido, o módulo de impressão não expõe directamente a Framework de Reporting, mas sim um conjunto de serviços que permitem utilizar as funcionalidades disponíveis nesta Framework. A Framework Athena disponibiliza um "contentor" centralizado de serviços que é implementado seguindo o padrão ***Service Locator*** (22), e que tem como objectivo permitir aos vários módulos o registo de serviços. Desta forma, o módulo de impressão regista neste "contentor" o serviço de impressão, que pode ser depois utilizado por outros módulos para executar as funcionalidades de impressão. Na Fig 5.3 é apresentada a *interface* do serviço de impressão:

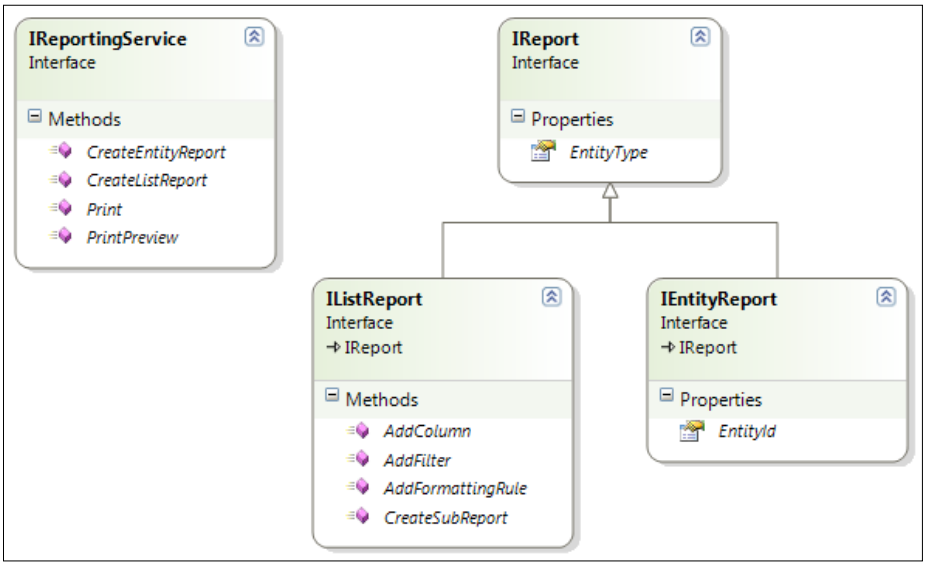


Fig 5.3 - Serviço de impressão

Na Tabela 5.2 é apresentada a descrição de cada um dos componentes que fazem parte da *interface* do serviço de impressão.

Componente	Descrição
IReportingService	Serviço de impressão que disponibiliza as funcionalidades de impressão para outros módulos.

IReport	*Interface* base de um *report*.
IListReport	*Interface* para os *reports* do tipo lista.
IEntityReport	*Interface* para os *reports* de entidades.

Tabela 5.2 - Serviço de impressão

O serviço de impressão disponibiliza os seguintes métodos:

- **CreateEntityReport**: cria um *report* do tipo entidade (***IEntityReport***).
- **CreateListReport**: cria um *report* do tipo lista (***IListReport***).
- **Print**: executa o serviço de impressão para um *report* (***IReport***).
- **PrintPreview**: executa o serviço de pré-visualização para um *report* (***IReport***).

Como se pode observar na Fig 5.3, juntamente com o serviço de impressão são disponibilizadas as *interfaces* para cada um dos tipos de *reports* (listas e entidades), tendo cada uma destes os métodos concretos para cada um dos tipos. Significa isto que o serviço de impressão propriamente dito apenas disponibiliza os métodos de criação e impressão dos *reports* sendo que os métodos específicos para cada um dos tipos de *reports* são disponibilizados pelas *interfaces* respectivas. A implementação concreta de cada um dos tipos de *reports* (***IEntityReport*** e ***IListReport***) é feita no módulo de impressão.

5.2.1 Listas

O componente responsável pela gestão e exploração (apresentação) de listas, dá pelo nome de **Query Builder**. Uma vez que este componente permite a manipulação de listas em "*runtime*", isto é, o utilizador pode adicionar/remover colunas, criar filtros, adicionar formatações condicionais, o contexto de impressão muda de acordo com as configurações da lista. Isto significa que o Query Builder tem de construir o contexto de impressão de acordo com as definições da lista que está a ser apresentada, recorrendo ao serviço de impressão para criar um *report* do tipo lista (***IListReport***) e depois utilizando os métodos por este tipo de *report* para adicionar colunas, filtros e formatações condicionais. Neste tipo de *reports* é também possível criar *sub-reports* também do tipo lista, isto para que seja possível a impressão de listas hierárquicas.

Na Fig 5.4 é apresentada a lista de armazéns, apenas estando visíveis duas colunas: a chave e a descrição do armazém.

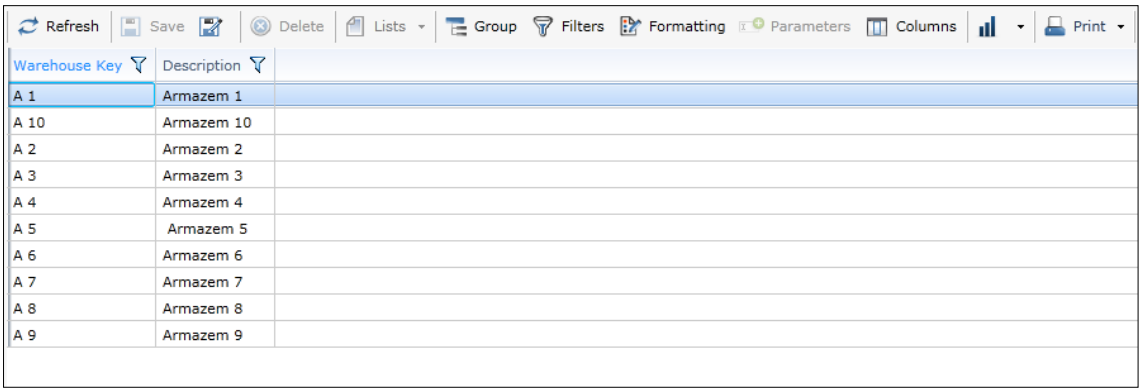


Fig 5.4 - Lista de armazéns

Na Fig 5.5 é apresentada a impressão da lista anterior (Fig 5.4):

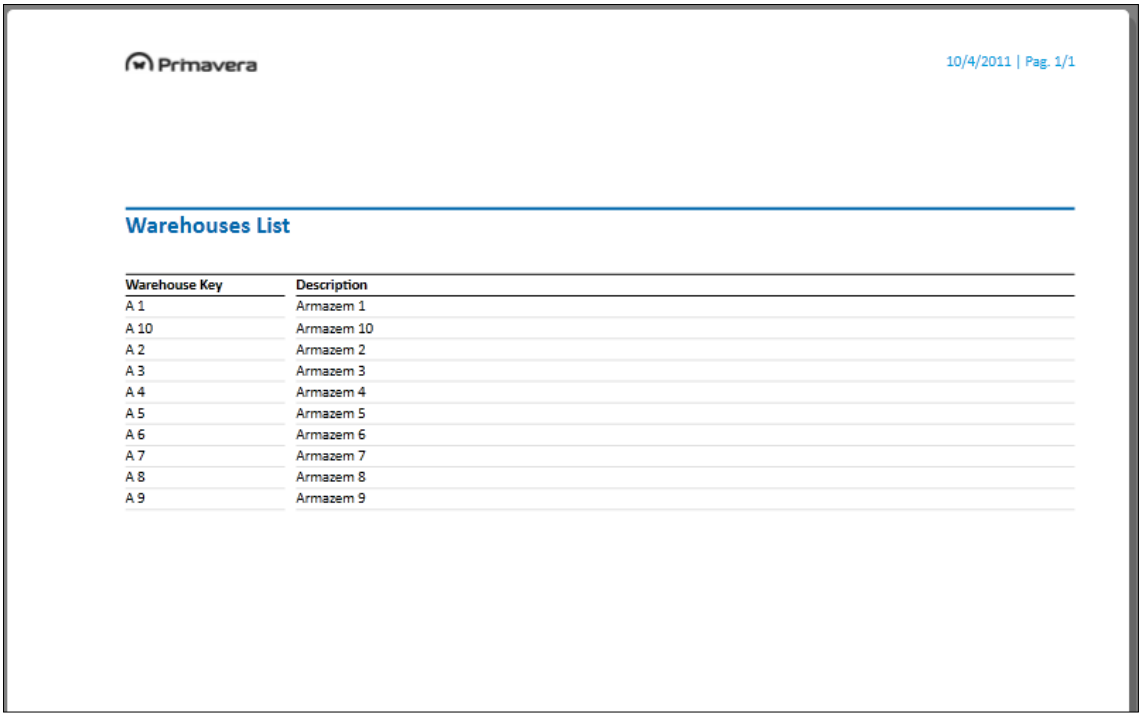
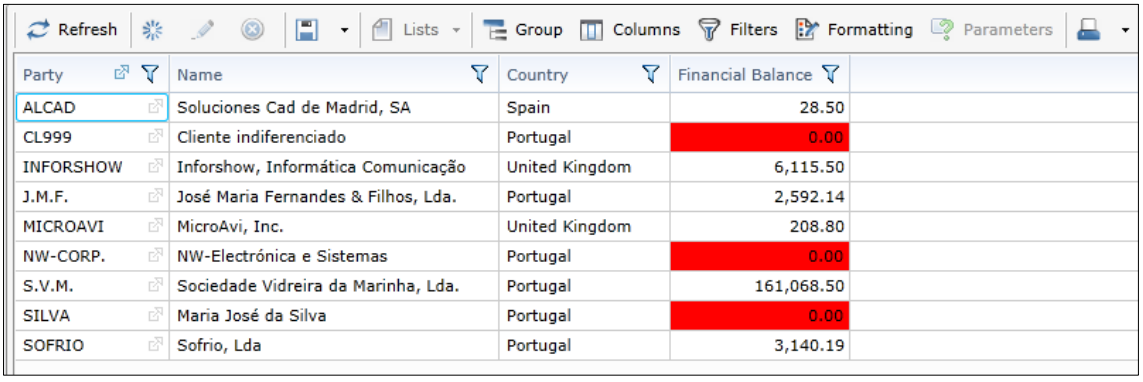


Fig 5.5 - Impressão da lista de armazéns

Outro exemplo de impressão de listas é a lista de clientes (Fig 5.6). Nesta lista estão definidas quatro colunas: a chave (*Party*), o nome (*Name*), o país de origem (*Country*) e o saldo financeiro (*Financial Balance*). A esta lista está aplicada uma formatação condicional, que coloca a coluna referente ao saldo financeiro com o fundo a vermelho para os clientes em que o saldo é igual a zero.



Party	Name	Country	Financial Balance
ALCAD	Soluciones Cad de Madrid, SA	Spain	28.50
CL999	Cliente indiferenciado	Portugal	0.00
INFORSHOW	Inforshow, Informática Comunicação	United Kingdom	6,115.50
J.M.F.	José Maria Fernandes & Filhos, Lda.	Portugal	2,592.14
MICROAVI	MicroAvi, Inc.	United Kingdom	208.80
NW-CORP.	NW-Electrónica e Sistemas	Portugal	0.00
S.V.M.	Sociedade Vidreira da Marinha, Lda.	Portugal	161,068.50
SILVA	Maria José da Silva	Portugal	0.00
SOFRIO	Sofrio, Lda	Portugal	3,140.19

Fig 5.6 - Lista de clientes

Na Fig 5.7 é apresentada a impressão da lista de clientes (Fig 5.6), e como se pode observar pela imagem, na impressão está também reflectida a formação condicional que estava aplicada à lista.

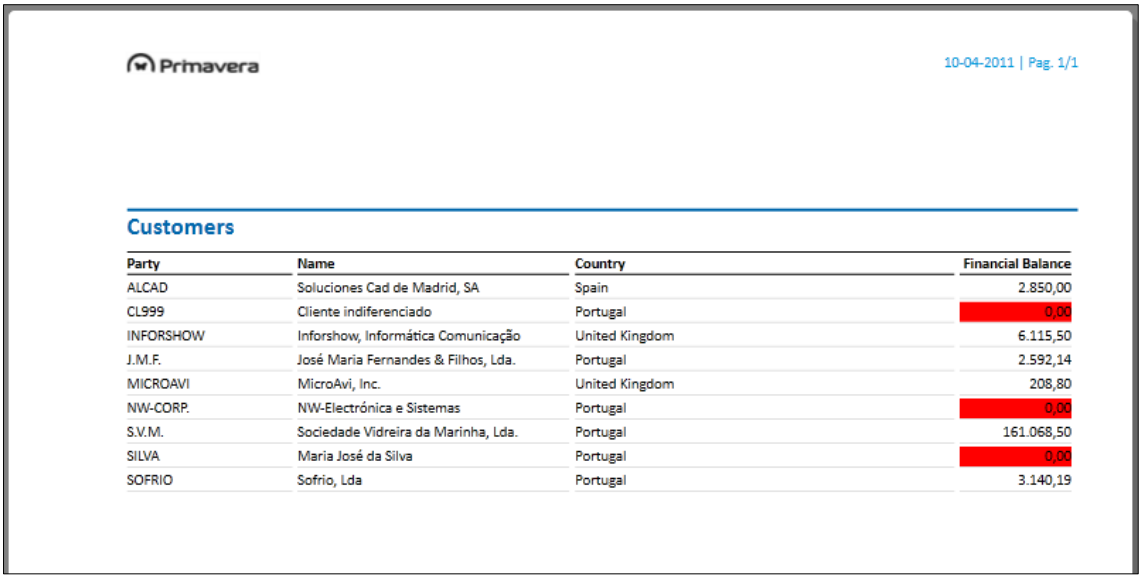


Fig 5.7 - Impressão da lista de clientes

5.2.2 Entidades

No caso da impressão de entidades, a abordagem é um pouco diferente da seguida nas listas, em que o *ReportModel* é dinâmico, uma vez que depende das configurações da lista (colunas, filtros, formatações). No caso das entidades o *ReportModel* é estático para a mesma entidade. Assim sendo, o *ReportModel* para as entidades que suportam impressão é gerado em tempo de desenvolvimento do produto, ficando registado na tabela ***EntityReporModelDefinition***. Depois, quando é pedido uma impressão para uma determinada entidade, apenas é necessário passar para o módulo de impressão qual o tipo da entidade (ex: Cliente) e o identificador do registo a imprimir (ex: a chave do cliente). Apenas com base nestes parâmetros, o módulo de impressão é responsável por carregar o *ReportModel* para o tipo de entidade respectivo, e de o passar para a Framework de Reporting para a respectiva geração.

O *ReportModel* para cada das entidades que suportam impressão é gerado com base no modelo do ***Presentation Designer***. O ***Presentation Designer*** é mais uma das ferramentas disponibilizadas pela Framework Athena, e que tem como função a modelação dos formulários para as entidades de cada módulo de um produto Athena. Até ao processo de geração do *ReportModel* são dados os seguintes passos:

1. Modelação das entidades no Modelo de Entidades (***Entities Designer***).
2. Modelação dos formulários (***Presentation Designer***).
3. Geração do *ReportModel* com base no modelo ***Presentation Designer***.

De forma a se perceber melhor os passos envolvidos até à impressão de uma entidade, é apresentado o seguinte exemplo (Fig 5.8) em detalhe:

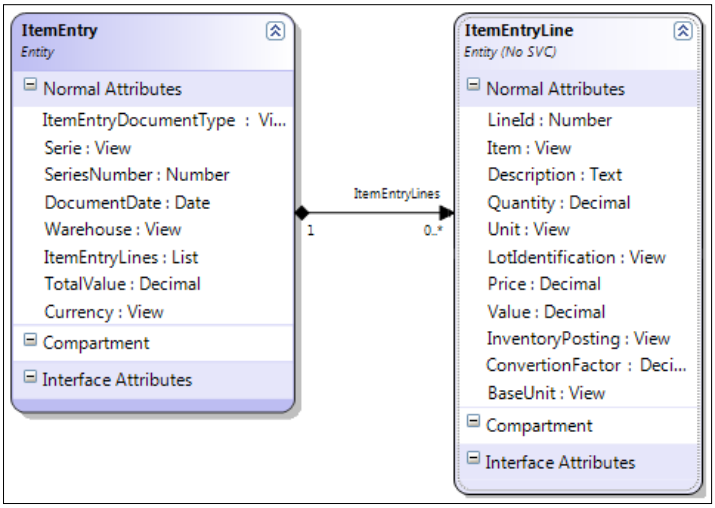


Fig 5.8 - Modelo de entidades: entrada em stock

Neste exemplo estão definidas as principais entidades que representam o documento de entrada de artigos em stock. A entidade ***ItemEntry*** tem os dados mestre do documento, como o tipo de documento, a serie e número de serie, a data de entrada, o armazém para o qual vai ser dada a entrada de artigos, e ainda o valor total e respectiva moeda da entrada. Uma entrada (***ItemEntry***) tem a lista de artigos definida pela associação ***ItemEntryLines***, que é definida pela entidade ***ItemEntryLine*** e que representa os artigos numa entrada de stock. Nesta entidade estão definidos dados como o artigo, a descrição, a quantidade e a unidade, a identificação do lote, o preço unitário e o valor total da respectiva linha.

Depois de modeladas as entidades envolvidas na entrada de artigos em stock, segue-se a modelação do respectivo formulário no ***Presentation Designer***:

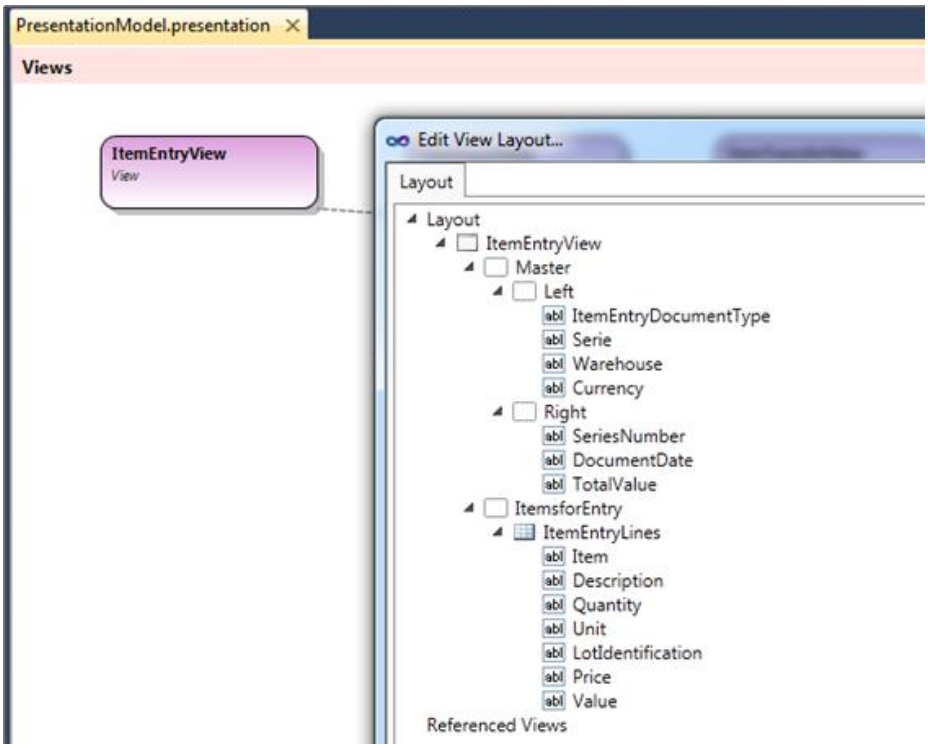


Fig 5.9 - Modelação do formulário de entrada em stock

Para o formulário de entrada de artigos em stock são definidos dois *containers* principais, em que no primeiro ficam os dados mestre da entrada (***ItemEntry***) e no segundo a respectiva lista de artigos (***ItemEntryLines***). A partir deste modelo é gerado o respectivo formulário e também o *ReportModel* a utilizar na impressão dos documentos de entrada de artigos em stock.

A Fig 5.10 apresenta o formulário gerado a partir do ***Presentation Designer***:

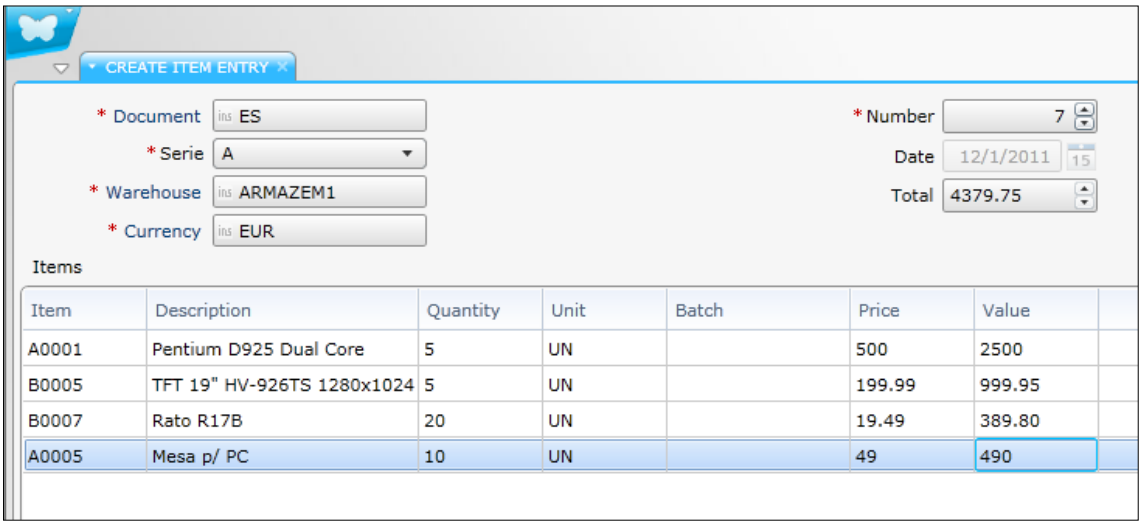


Fig 5.10 - Formulário de entrada em stock

A impressão do documento de entrada de artigos em stock é apresentada na Fig 5.11:

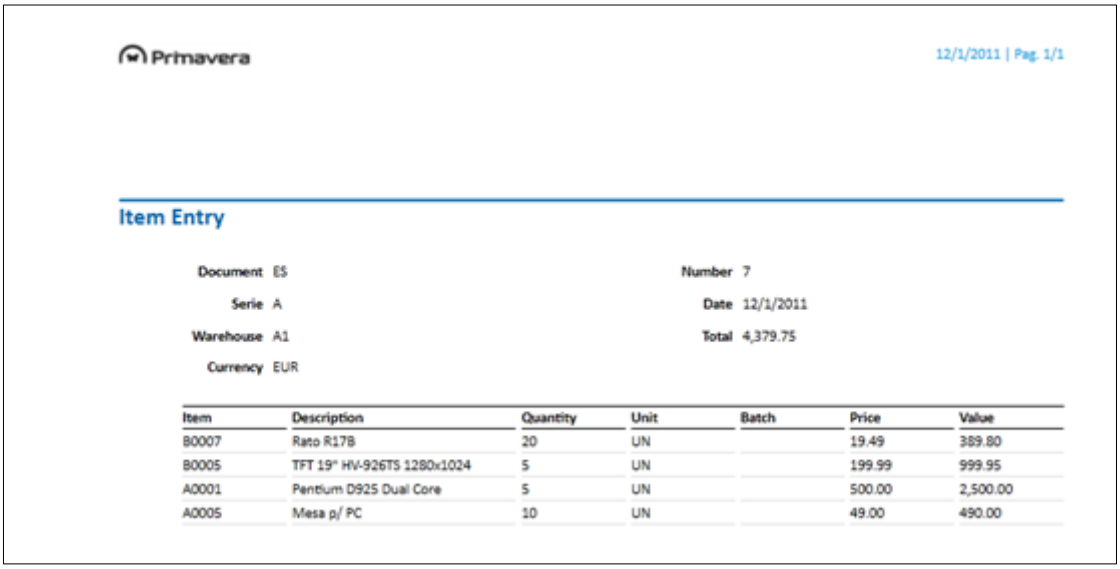


Fig 5.11 - Impressão do documento de entrada em stock

E desta forma, com base no modelo do ***Presentation Model*** é gerado o *ReportModel* para cada uma das entidades. As vantagens desta abordagem são várias, desde logo o facto de não existir intervenção directa por parte de quem desenvolve no *ReportModel*, diminuindo consequentemente o número de erros que possam surgir. Outra vantagem pode ser verificada no caso de existirem alterações à própria definição do *ReportModel*, sendo apenas necessário alterar a lógica de geração do mesmo.

Capítulo 6

6 Conclusões e trabalho futuro

Este trabalho surgiu da necessidade de eliminar algumas das limitações existentes no actual componente de *reporting* utilizado pelo ERP PRIMAVERA. Estas limitações passam principalmente pela necessidade de construção de um elevado número de mapas sem que seja possível partilhar qualquer tipo de configuração que possa ser comum aos vários relatórios. Isto leva a que num produto como o ERP PRIMAVERA seja necessário a construção de cerca de 2000 mapas para responder a todas as necessidades de *reporting.* O custo associado ao desenvolvimento e manutenção destes mapas é bastante elevado, sendo por isso importante encontrar uma nova solução de *reporting* para integrar em futuros produtos. Este trabalho tinha então como principal objectivo o desenvolvimento de um componente de *reporting* para ser integrado num módulo da Framework Athena. Este componente será o responsável por disponibilizar aos produtos desenvolvidos sob esta tecnologia as funcionalidades de impressão. O resultado é a Framework de Reporting e o respectivo módulo de impressão no qual vai ser integrada. Os principais requisitos são a geração automática dos *reports* a partir dos modelos das aplicações e a capacidade de poderem ser definidas determinadas configurações nos mesmos, sendo aqui o objectivo reduzir/eliminar o processo manual de construção de *reports*.

Para desenvolver este trabalho foi necessário analisar o componente de *reporting* utilizado actualmente no ERP PRIMAVERA, sendo o objectivo recolher informação acerca das necessidades em termos de funcionalidades e também das principais limitações existentes. Esta análise abrangeu também os relatórios existentes no ERP, com o objectivo de agrupar os vários mapas de acordo com as suas características e definindo as regras de construção para cada um. Desta análise surgiram 3 categorias de mapas: tipo 1, tipo 2 e tipo 3. No tipo 1 estão incluídos os mapas de listas, como por exemplo a lista de artigos ou a lista de clientes. Os mapas do tipo 2 são relativos à impressão de entidades simples, com layout e regras de negócio simples como por exemplo a ficha de artigo ou do cliente. No tipo 3 estão incluídos os mapas mais complexos, quer no processamento quer na sua estrutura, como por exemplo uma factura ou um extracto de conta.

A Framework de Reporting foi desenvolvida tendo em conta esta análise, sempre com o objectivo de automatizar a construção do maior número possível de *reports*. Foi definido como objectivo principal a automatização de todos os *reports* dos tipos 1 e 2. Para atingir este objectivo foram desenvolvidos vários componentes, entre os quais está o ***ReportModel***, que é o modelo a partir do qual a framework constrói os *reports*. Este modelo é gerado com base no ***Presentation Model*** em tempo de desenvolvimento de um produto no caso da impressão de entidades, e construído dinamicamente pelo componente de exploração de listas (***Query Builder***) no caso da impressão de listas. Desta forma é possível obter a totalidade dos *reports* gerados automaticamente, não existindo necessidade de construir *custom reports* para nenhum mapa do tipo 1 ou 2. Relativamente aos mapas do tipo 3, que são os documentos mais complexos como facturas ou extracto de contas, é necessária a construção de *custom reports*, devido à complexidade dos cálculos envolvidos e do respectivo layout.

Relativamente às configurações de *layout* e formatação dos *reports* foi desenvolvido na Framework de Reporting o componente de Templates. Nestes templates podem ser definidas formatações dos controlos, definições relativas às fontes a aplicar no *report* e ainda e definição de controlos como por exemplo a imagem com o logótipo da empresa. A grande vantagem na aplicação de templates é o facto de as definições de layout e formatação dos *reports* passarem a estar acessíveis num único local. Por isso, quando é necessário alterar por exemplo o tipo de fonte ou o logótipo dos *reports*, basta alterar a definição dos templates. De notar também que mesmo no caso dos *custom reports* as configurações dos templates são aplicadas, possibilitando desta forma ter uma parte, ainda que reduzida, que é configurável neste tipo de *reports*.

Analisando os resultados conseguidos com o desenvolvimento deste trabalho, pode concluir-se que o principal objectivo foi atingido, isto é, a totalidade dos mapas do tipo 1 e tipo 2 são gerados automaticamente. Consequentemente, os custos associados ao desenvolvimento e manutenção de mapas deverá diminuir, já que todo o processo é automático e baseado nos modelos das aplicações em questão, e quando for necessário efectuar algum tipo de intervenção (adicionar novas funcionalidades; correcção de erros), esta será feita apenas ao nível da Framework de Reporting, e não em cada um dos mapas da aplicação como acontece actualmente. Um outro ponto positivo está relacionado com o facto de apesar da Framework de Reporting ter sido desenvolvida sempre tendo em conta o contexto do projecto Athena, e de que teria de ser construído um módulo para ser integrada nos produtos Athena, a Framework em si é independente da Framework Athena, podendo por isso ser aplicada noutro tipo de projectos com necessidades de impressão.

Trabalho futuro

Um ponto onde é necessário apostar numa futura versão da Framework de Reporting é a questão dos *custom reports*. Apesar de nesta versão existir já alguma automatização neste tipo de *reports*, em que a framework é capaz de fazer personalizações com base num template, estes exigem ainda um elevado custo no seu desenvolvimento. Torna-se por isso necessário voltar a olhar para este tipo de *reports* e identificar que tipo de características os distinguem, agrupando-os se necessário em tipos mais específicos para que seja possível de alguma forma introduzir na Framework de Reporting novos componentes que sejam capazes automatizar a sua construção.

Bibliografia

1. **PRIMAVERA BSS.** [Online] http://www.primaverabss.com.

2. **Swithinbank, Peter, et al., et al.** *Patterns: Model-Driven Development Using IBM Rational Software Architect.* 2005.

3. **Crystal Reports.** Crystal Reports. [Online] http://www.crystalreports.com/.

4. **Gamma, Erich, et al., et al.** *Design Patterns: Elements of Reusable Object-Oriented Software.* 1994.

5. **Beck, Kent e Johnson, Ralph.** Patterns Generate Architectures.

6. **Riehle, Dirk.** Framework Design: A Role Modeling Approach. 2000.

7. Building Object-Oriented Frameworks. [Online] http://lhcb-comp.web.cern.ch/lhcb-comp/Components/postscript/buildingoo.pdf.

8. *Applying Patterns and Frameworks to Develop Object-Oriented Communication Software.* **Schmidt, Douglas C.** 1997.

9. **Fayad, Mohamed e Schmidt, Douglas C.** Object-Oriented Application Frameworks. 1997.

10. **Foote, Brian.** Designing to facilitate change with object-oriented frameworks. 1988.

11. *Domain-Specific Language Design Requires Feature Descriptions.* **van Deursen, A. e Klint, P.** 2002, Journal of Computing and Information Technology, pp. 1-8.

12. **Primavera BSS.** *Project Orion: Vision/Scope - Initial vision and scope of Project Orion.* 2009.

13. **Ben-Avraham, Dee e Reilly, Kevin.** Model-Driven Development.

14. **Selic, Bran.** The Pragmatics of Model-Driven Development. 2003.

15. *Model-driven development: The good, the bad, and the ugly.* **Hailpern, B. e Tarr, P.** 2006, IBM Systems Journal, pp. 451-461.

16. **Martin, Robert C.** Design Principles and Design Patterns. 2000.

17. **Chen , Xin.** *Developing Application Frameworks in .NET.* 2004.

18. Report. *Wikipédia.* [Online] [Citação: 26 de 11 de 2009.] http://en.wikipedia.org/wiki/Report.

19. **Data Dynamics.** ActiveReports. [Online] http://www.datadynamics.com/Products/ActiveReports/.

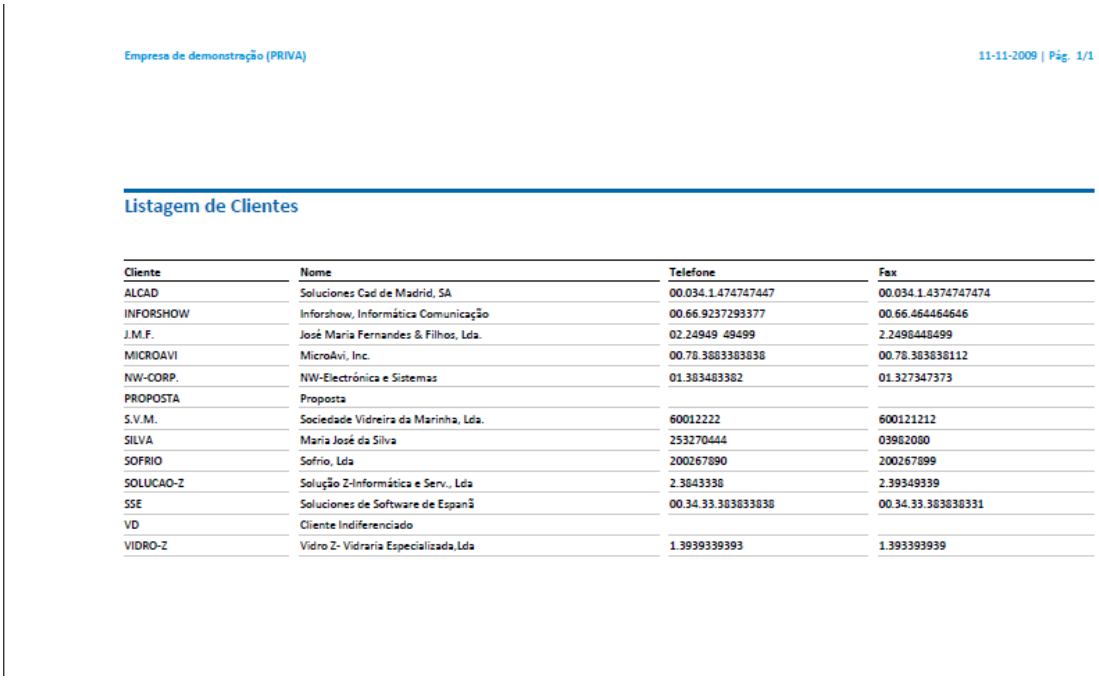
20. **Stimulsoft.** Stimulsoft. [Online] http://www.stimulsoft.com/.

21. **DevExpress.** DevExpress XtraReports. [Online] http://www.devexpress.com/Products/NET/Reporting/.

22. **Fowler, Martin.** Inversion of Control Containers and the Dependency Injection pattern. [Online] 2004. http://martinfowler.com/articles/injection.html.

23. *Active Guidance of Framework Development.* **Pree, Wolfgang, et al., et al.** 1995.

Anexo A



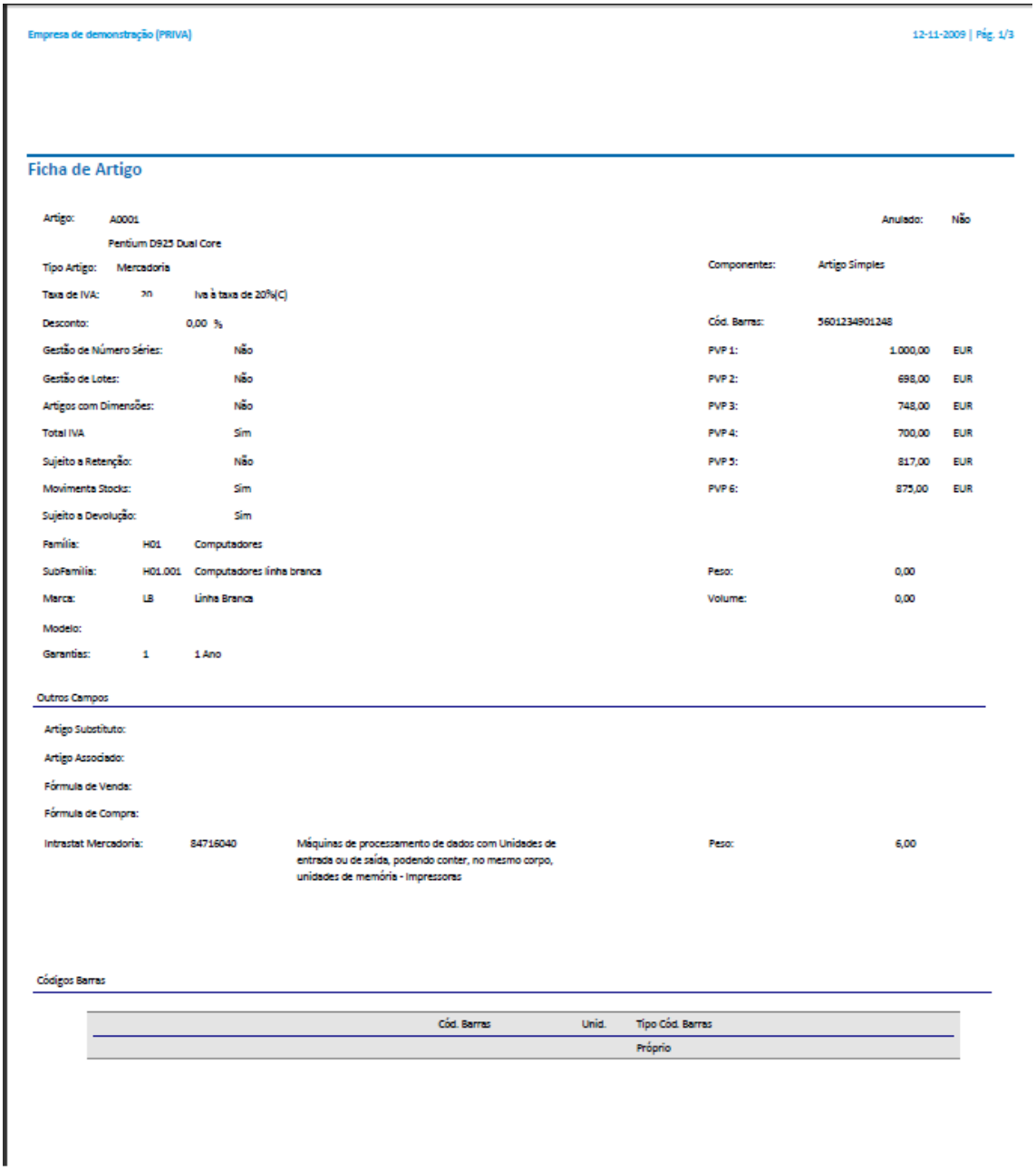
Empresa de demonstração (PRIVA) 11-11-2009 | Pág. 1/1

Listagem de Clientes

Cliente	Nome	Telefone	Fax
ALCAD	Soluciones Cad de Madrid, SA	00.034.1.474747447	00.034.1.4374747474
INFORSHOW	Inforshow, Informática Comunicação	00.66.9237293377	00.66.464464646
J.M.F.	José Maria Fernandes & Filhos, Lda.	02.24949 49499	2.2498448499
MICROAVI	MicroAvi, Inc.	00.78.3883383838	00.78.383838112
NW-CORP.	NW-Electrónica e Sistemas	01.383483382	01.327347373
PROPOSTA	Proposta		
S.V.M.	Sociedade Vidreira da Marinha, Lda.	60012222	600121212
SILVA	Maria José da Silva	253270444	03982080
SOFRIO	Sofrio, Lda	200267890	200267899
SOLUCAO-Z	Solução Z-Informática e Serv., Lda	2.3843338	2.39349339
SSE	Soluciones de Software de Espanã	00.34.33.383833838	00.34.33.383838331
VD	Cliente Indiferenciado		
VIDRO-Z	Vidro Z- Vidraria Especializada,Lda	1.3939339393	1.393393939

Relatório do tipo 1: lista de clientes

Anexo B



Empresa de demonstração (PRIVA) 12-11-2009 | Pág. 1/3

Ficha de Artigo

Artigo:	A0001			Anulado:	Não
	Pentium D925 Dual Core				
Tipo Artigo:	Mercadoria		Componentes:	Artigo Simples	
Taxa de IVA:	20	Iva à taxa de 20%(C)			
Desconto:	0,00 %		Cód. Barras:	5601234901248	
Gestão de Número Séries:	Não		PVP 1:	1.000,00	EUR
Gestão de Lotes:	Não		PVP 2:	698,00	EUR
Artigos com Dimensões:	Não		PVP 3:	748,00	EUR
Total IVA	Sim		PVP 4:	700,00	EUR
Sujeito a Retenção:	Não		PVP 5:	817,00	EUR
Movimenta Stocks:	Sim		PVP 6:	875,00	EUR
Sujeito a Devolução:	Sim				
Família:	H01	Computadores			
SubFamília:	H01.001	Computadores linha branca	Peso:	0,00	
Marca:	LB	Linha Branca	Volume:	0,00	
Modelo:					
Garantias:	1	1 Ano			

Outros Campos

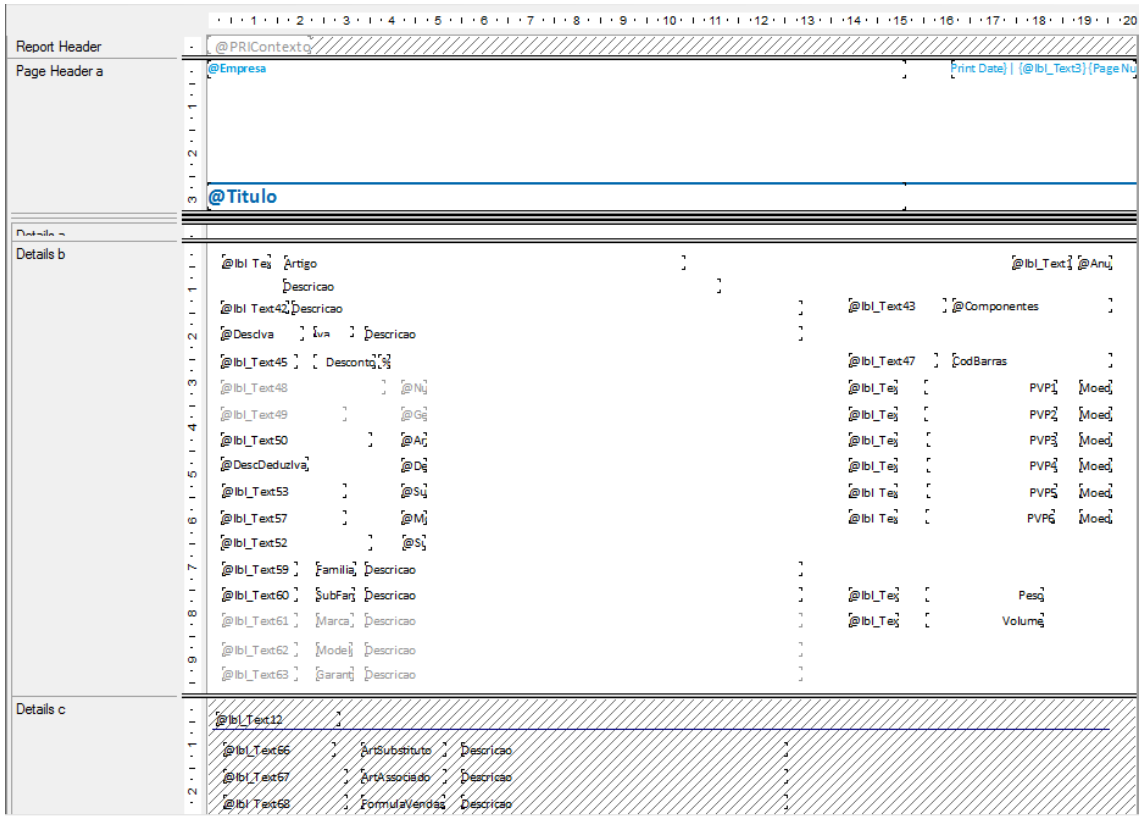
Artigo Substituto:				
Artigo Associado:				
Fórmula de Venda:				
Fórmula de Compra:				
Intrastat Mercadoria:	84716040	Máquinas de processamento de dados com Unidades de entrada ou de saída, podendo conter, no mesmo corpo, unidades de memória - Impressoras	Peso:	6,00

Códigos Barras

Cód. Barras	Unid.	Tipo Cód. Barras
		Próprio

Relatório tipo 2: ficha de artigo

Anexo C



Crystal Reports Designer - Relatório tipo 2: ficha de arigo

Anexo D

Telef. Fax.

Capital Social 0,00
Cons. Reg. Com.
Matrícula Nº

Exmo.(s) Sr.(s)
Maria José da Silva
Rua de S. Geraldo Nº 41

Cividade
4700.024 Braga

Original

Factura Nº 22/A

V/Nº Contrib.	Requisição	Moeda	Câmbio	Data
		EUR	1,000000	30-04-2009

Desc. Cli.	Desc. Fin.	Vencimento	Condição Pagamento
10,00	0,00	30-05-2009	Factura 30 dias

Artigo	Descrição	Quant.	Un	Pr. Unitário	Desc.	IVA	Total Líquido
GAR001	Grarrafa de Whisky 15 anos B&A	1,00	CXS	216,00	0,00	20,00	194,40
GAR009	Garrafe Bagaceira Regional - Caves Altas	2,00	UN	21,40	0,00	20,00	38,52
GAR027	Vinho do Porto Vintage/1994 - Gold Grapes	4,00	UN	67,00	0,00	20,00	241,20
	Documentos Originais:						
	GR A/2 de 27-04-2009						

Quadro Resumo do IVA			
Taxa	Incidência	Total IVA	Motivo Isenção
20,00	506,72	101,34	

Mercadoria/Serviços	526,80
Descontos Comerciais	-52,68
Desconto Financeiro	0,00
Portes	0,00
Outros Serviços	0,00
Adiantamentos	0,00
Ecovalor	0,00
IEC	32,60
IVA	101,34
Acerto	0,00

Local de Carga	Carga	Modo de Expedição
N/ Morada	26-01-2009 / 18:38	N/ Viatura
Local de Descarga	**Descarga**	**Matrícula**
Rua José Inácio Peixoto Nº 68 Sé 4700.892 Braga		

Total (EUR) 608,06

Documento Processado por Computador / © PRIMAVERA BSS /

Relatório tipo 3: factura