FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FEUP

Improving Database Reporting Processes with XML Technologies (Case Study of *Sage Next*)

Ricardo Miguel dos Santos Leandro

Report of Project

Master in Informatics and Computing Engineering

Supervisor: João Correia Lopes (Assistant Teacher)

3rd March, 2009

Improving Database Reporting Processes with XML Technologies (Case Study of *Sage Next*)

Ricardo Miguel dos Santos Leandro

Report of Project
Master in Informatics and Computing Engineering

Approved in oral examination by the committee:

Chair: Ana Paula Cunha da Rocha (Assistant Teacher)

External Examiner: José Carlos Leite Ramalho (Assistant Teacher)
Internal Examiner: João Correia Lopes (Assistant Teacher)

20^{st} March, 2009

Abstract

This project aims to prove that it is possible to use XML language specifications and associated technologies in the creation of reports. The current engine to generate reports owned by Sage was made more than 15 years ago and has been in use ever since. Due to its level of rudimentariness, an improvement was needed. The ability to export reports onto other commercial report generating platforms is a crucial step to Sage's ERP (*Sage Next*) evolution. Therefore, the use of an XML language to define reports will increase their portability.

To make a new reporting architecture, some XML technologies, a programing language and a reporting tool were chosen. After that, an XML language based in *Microsoft*'s Report Definition Language (RDL) was developed: the SRDL (Sage Report Definition Language). To prove that the created SRDL had the desired complexity it was made a conversion process from the reports of the previous reporting engine to the new architecture. After that, an application was developed using an API (Application Programming Interface) of a commercial reporting tool (*List & Label*) to output the converted reports.

To access the suitability of the components of the architecture developed in this work, four distinct reports were tested. All the results were successful, showing that with the created SRDL it is possible to create reports with a different tool, while maintaining the same visual look.

In the future, more tests will be made with the developing of a component, which will feed the new reporting engine with real data from several types of databases.

Resumo

Os objectivos deste projecto direccionam-se no sentido de provar que é possível usar especificações de linguagens XML e as tecnologias associadas ao XML na criação de relatórios. O actual motor de geração de relatórios da Sage foi criado há mais de 15 anos e tem sido utilizado desde então. Por ser rudimentar, mostrou-se necessária uma melhoria.

A capacidade de exportar relatórios para outras plataformas de geração de relatórios comerciais é um passo crucial para a evolução de um ERP da *Sage*: o (*Sage Next*). Assim, o recurso a uma linguagem XML para definir os relatórios irá aumentar a sua portabilidade.

Para fazer uma nova arquitectura de criação de relatórios foram escolhidas algumas tecnologias XML, uma linguagem de programação e uma ferramenta de geração de relatórios. Posteriormente, uma linguagem XML baseada na RDL (*Report Definition Language*) da Microsoft foi desenvolvida: a SRDL (*Sage Report Definition Language*).

Para provar que a SRDL criada tinha a complexidade desejada, foi feito um processo de conversão dos relatorios do antigo motor de geração para a nova plataforma. Em seguida, uma aplicação foi desenvolvida usando a API (*Application Programming Interface*) de uma ferramenta de criação de relatórios comercial para gerar os relatórios convertidos.

Após terminar a maioria dos componentes da arquitectura, foram testados quatro relatórios distintos. Todos os resultados foram bem sucedidos, provando que com o SRDL criado é possível criar relatórios com uma ferramenta diferente, mantendo a mesmo aspecto visual.

Futuramente, mais testes serão feitos com o desenvolvimento de um componente que irá alimentar o novo motor de geração de relatórios com dados reais de diversos tipos de bases de dados.

Acknowledgements

For all the support, advice and constant availability I must thank João Correia Lopes. For all the help integrating in *Sage* and for all support when developing new ideas and not giving up on them I must thank Jorge Morais. To Sage I must thank the opportunity they gave me to accomplish this project and to be a part of a big company.

To my girlfriend I thank all the inspiration, motivation and the encouragement to successfully finish this project. To my parents and my sister I thank the everlasting support they gave when I needed the most. To my friends I thank the courage and the moments of joy they gave me.

Ricardo Leandro

"Design is not just what it looks like and feels like. Design is how it works."

Steve Jobs

Contents

List of Figures

Abbreviations

XML	eXtensible Markup Language
XSLT	eXtensible Stylesheet Language Transformations
RDL	Report Definition Language
SRDL	Sage Report Definition Language
ERP	Enterprise Resource Planning
VB	Visual Basic
API	Application Programming Interface
POS	Point Of Sale

Chapter 1

Introduction

All around the world, companies are becoming more and more connected internally and with other companies. They want immediate responses to costumers when they make large orders or when shipments from suppliers are delayed. Moreover, managers want to know what impact these events will have on the business and how their business is at any point in time [Lau06].

This thesis presents a new reporting architecture model for Sage Portugal's applications. *Sage Next*, which is a ERP program for medium sized companies uses three types of report generating tools: *List & Label*, *Crystal Reports* and the third one, *Sage Map*, self-made. The first two reporting tools are different commercial solutions, which require using its proper report designer for creating of report layouts. The self-made reporting tool (*Sage Map*) has been in service for 15 years and, using it to create new reports or correcting old ones, is a highly demanding and time consuming task that *Sage Portugal* wants to improve. For costumers it means they cannot easily move reports between different reporting tools and the options available for choosing new tools, that work with their existing execution environments, are few [Cor05].

In order to create a new reporting process, a report description language in XML was needed so that it would serve as an intermediate between *Sage Next* program and the report renderer tools.

1.1 Scope

Sage Portugal is a business software oriented developer company that provides solutions to all kinds of companies: from individual and medium sized to big ones with a business volume higher than five million euro. Their products offer solutions intended for

POS (points of sale), CRM (Custumer Relationship Management), accounting and business planning. One of that solutions is *Sage Next* which is designed for medium sized companies. This product is a small ERP with an integrated system containing several applications that help companies by making daily operations easier and faster.

> "Informed decision making is important at all levels within an organization, and easy access to accurate information is also essential."[Say07]

Sage Next solution provides helpful information on the business welfare through reports containing either summarized or extensive data. There should be a tool with the ability to extract and summarize this data in order to allow key decision makers to have a better view of the business model and take companies in the best possible direction [Pec04].

As was said previously, reporting tools are essential in ERP programs so the reporting engine within the ERP needs to be a well designed and future proof engine.

1.2 Project

This project consists on creating a new reporting architecture model for Sage programs replacing the old one with improvements in its processes and adding new features. The new model will support exporting the report's layouts to XML and will enable their independence from *Sage Next*. The language of the XML files exported will have to be a well defined one containing all aspects of *Sage*'s programs reporting area. Finally, with the aid of a reporting tool's (*List & Label*) API, an application will be made to parse the created XML files and render the reports, maintaining their previous visual style. This application not only will prove that the created language contains all that is needed by Sage Next program, but will also aid the conversion process of all the old reports (more than 2000).

1.3 Objectives and Motivation

The goal of creating a Report Definition Language is to promote an interoperability of commercial reporting products. This can be done by defining a common schema allowing an interchange of report definitions [Cor05].

As it is said above, the Sage Report Definition Language project consisted on creating and RDL language in XML to be used on Sage programs as an intermediary between them and reporting tools.

After its creation, it was needed a proof that the created language was comprehensive enough to define all aspects of Sage programs's reports. A reporting tool like *List & Label* was chosen as a report renderer and the result had to resemble as closely as possible to the old *Sage Map* reports's visual style.

Developing this project is an important step on *Sage Next*'s life cycle, because the reporting model will be improved, clearing it from older technologies and enabling the use of different reporting tools in the future.

1.4 Report Structure

This chapter is devoted to the presentation of the document, its purpose and structure, seeking to guide the reader in order to give a better reading experience. In chapter 2 it will be presented a detailed study of state of the art reporting technologies as well the one owned by Sage. In chapter 3 it will be showed a more comprehensive study of the reporting solution that was designed describing not only the current architecture of Sage Next program but also the one intended by Sage. Then, on chapter 4, it will described the architecture that was implemented, explaining all its components and processes, as well as the created XML language for defining reports (*SRDL*) and its main features. Finally, in chapter 6 an analysis on all the work made is presented as well as the level of satisfaction of each accomplished objective. After that improvements and new features that could be added to the reporting process in the future will be showed.

Chapter 2

Bibliographic Review

In this chapter it will be explained all the investigation process behind the solution design and what technologies were involved in it. To begin with the current reporting tools used in *Sage Next* application will be explained. After that, it will be made an analysis on the other possible technologies to be used, their limitations and fitness for this project.

2.1 Introduction

This project was developed in order to evolve the reporting model in *Sage*'s programs replacing older technologies. The main problems with the older model were the absence of a reporting language to define database reports and a very restrictive report design tool that is attached to *Sage Next* program.

The design tool had lack of functionalities that are of the most importance when creating or modifying reports: lack of a conditional language and variables to write formulas, lack of debugging help, lack of an undo functionality and absence of colors for formatting purposes. Finding existing solutions for creating a Report Definition Language was not an easy task, since only one existing possible solution was found. *Microsoft's Report Definition Language* specification was a well designed implementation of the concept and it contained all that was needed for creating *Sage Next* reports layouts. Far more complex than the RDL language desired *Sage*, only a subset of this specification was used for creating Sage's RDL.

2.2 Technology Review

In this section, the technologies involved in the SRDL project will be explored for a better understanding of *Sage Next* and its problems.

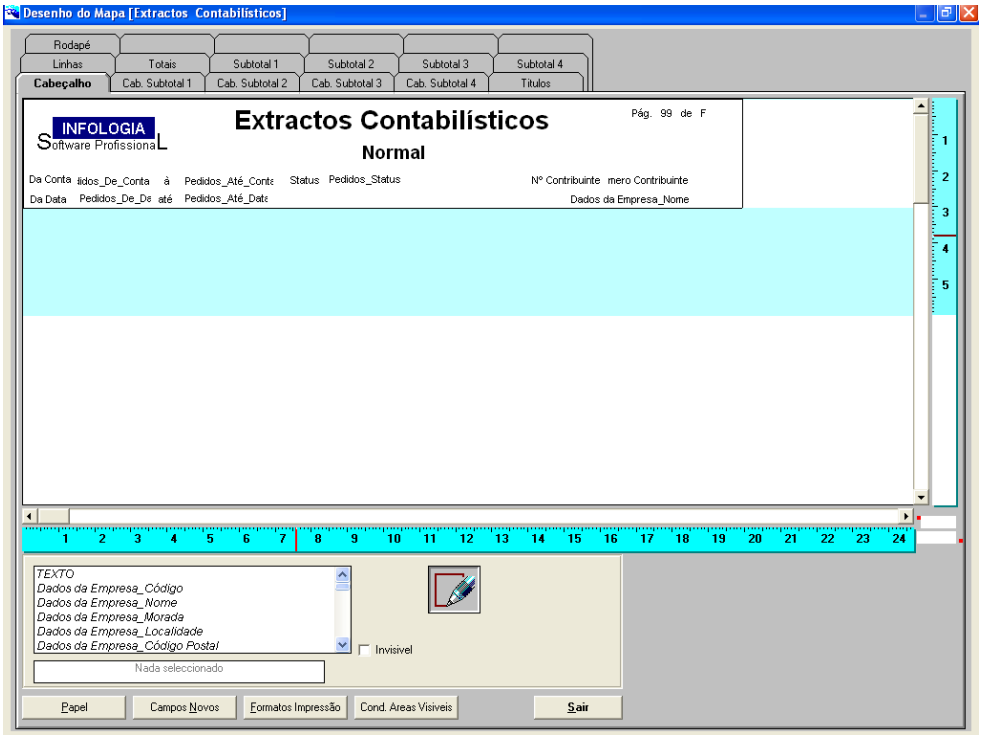

Figure 2.1: Sage Map Designer

2.2.1 Sage Next's Current Reporting Technologies

As already explained (see section 2.1),Sage Next is one application that uses a proprietary format for the definition of a report. In this section *Sage Map* as well as the other three existent technologies that are used for reporting, will be analyzed,

2.2.1.1 Sage Map

Sage Map is a reporting tool formed by a report template designer and report generating application embedded in *Sage Next* (see figure 2.1). The template designer does not have a WYSIWYG [1] editor and lacks the existence of a conditional language to write formulas. The layout visual editor is not an easy tool to work with. Most of the time, tricks and workarounds are needed in order for the reporting engine to output what is intended. Despite having some mathematical functions to write simple formulas, they do not have sufficient complexity to make all the needed procedures in the reporting area.

2.2.1.2 Crystal Reports

> "Crystal Reports remains the market leader and de facto standard for business and corporate report writing."[Pec04]

[1] What you see is what you get

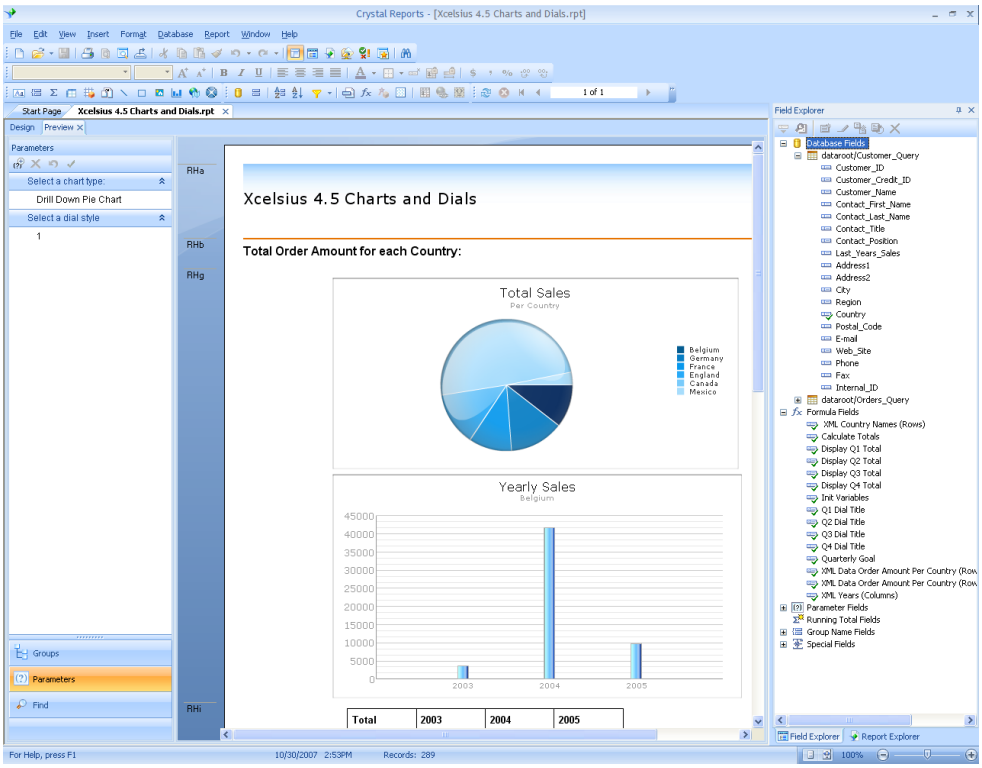

Figure 2.2: Crystal Reports Designer

Crystal Reports is, as shown before, the number one in data base reporting and this happens because it is not only a result of a "marriage of two former competitors"[Pec04] but also the most powerful application in the area. It supports: sorting and grouping in reports, creating geographic maps, formulas, custom functions, a variety of tools to make reports visually appealing, cross-tabs, charts, sub-reports, reporting from SQL databases and exporting reports to different file formats like XML among others [CRY09]. Despite being the most complete reporting solution, *Sage*'s costumers complained about its complex interface when trying to customize their reports.

2.2.1.3 List & Label

List & Label is a reporting tool made by *combit Software GmbH* and it was the last one added to *Sage Next*'s reporting architecture. *List & Label* is a powerful tool, with which anyone can use, quickly and effectively, to fulfill all the reporting needs. It has a freely distributable designer, available in 15 languages, causing impression on customers with the final product. [cSG09]

In figure 2.3, an example of the designer working can be seen. With *combit Software*'s *List & Label* reporting solutions, any developer can add a reporting engine to it's own applications easily, and quickly create reports assisted with useful functions. To integrate

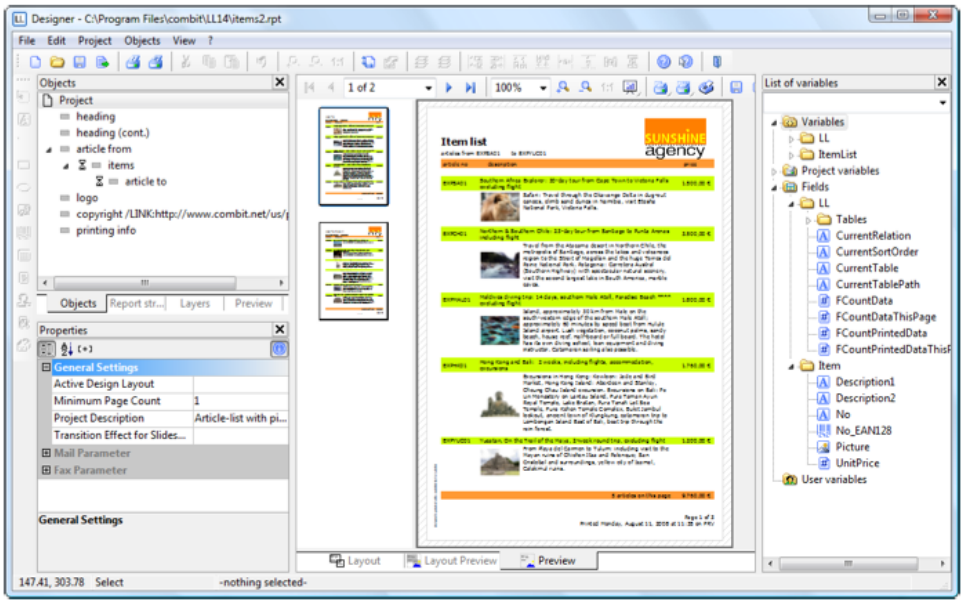

Figure 2.3: List & Label Designer

the report generator into an existing application, a large variety of programming languages like Java, Visual Basic, Clarion, Cobol, Delphi and C#.NET are available.

2.2.2 State of the art: Technologies and their Limitations

In this section the possible technologies as well their features will be analyzed to determine their importance to this project.

> "In today's database reporting market, most vendor applications use a proprietary format for the definition of a report. In addition, vendors that provide a report execution usually only support their own design tools." [Cor05]

In other words, proprietary formats for defining reports are used in most vendor applications like the ones used in *Sage Next*. Using different formats is not efficient because reports made with one tool can not be used by other tools for report generation. Furthermore when it is needed to upgrade reports from one format to another the report has to be done from scratch. A research was made to find out if there was a solution to this problem.

2.2.2.1 Microsoft RDL

Microsoft tried to solve problems in the reporting area by creating a RDL language. Their objectives for the RDL development were very clear: "The goal of Report Definition

Language (RDL) is to promote the interoperability of commercial reporting products by defining a common schema that allows interchange of report definitions." [Cor05]

Microsoft RDL development started before 2003 and, since then, they launched three versions of the RDL's specification. The version used was the 2005 version because when the project started the 2008 version was not available yet.

This technology is intended to promote the interoperability of commercial reporting tools and allow the interchange of report definitions. RDL is also meant to be fully encapsulated. This means that when successfully interpreting an RDL document, it should not be required any understanding of the source application. Also, with RDL it should be possible to output a variety of formats like web pages, PDF or XML.[Cor05].

This technology consists of an XML schema containing all the parts that describe the three kinds of information that a report can have: data, layout and properties. Data represents the informations on how to obtain the real data, like SQL queries, as well the structure of the data. The Layout part is where it is described, how the data will be presented and where the formatting information is provided. The last part is where properties, such as author, parameters and images within the report, are.

2.2.2.2 fyiReporting

After analyzing *Microsoft*'s RDL, it was found a project using it. *fyiReporting* is an open-source RDL project that has a powerful reporting tool. This tool has support for charts tables and free forms and outputs in several formats. The output formats can be: HTML, HTML, PDF, Excel, RTF, XML, .Net Control, Web Archive, and print. This project is released under the terms of the Apache License Version 2 which allows the use of the RDL Project 4 libraries and programs in either open source or commercial applications as long as they credit *fyiReporting Software, LLC*. *fyiReporting* project includes a WYSIWYG designer which allows the creation of reports without any knowledge of RDL. Also, wizards are available for creating new reports and for inserting new Tables, Matrices, and Charts into existing reports.[fS09] In figure 2.4 it can be seen the fyiReport designer.

Although this was a complete tool, *Sage* wanted to use *List & Label* because of commercial interests.

2.2.2.3 Sage 50 Accounts 2009

After the beginning of this project (during the RDL implementation), *Sage Portugal* was informed that another division of the company in the United Kingdom had already solved similar problems in the reporting area. The tool, which is embedded in Sage's program *Sage 50 Accounts 2009*, is called *Sage Report Designer* and has a powerful desktop application for designing reports with a WYSIWYG editor. This tool enables the creation

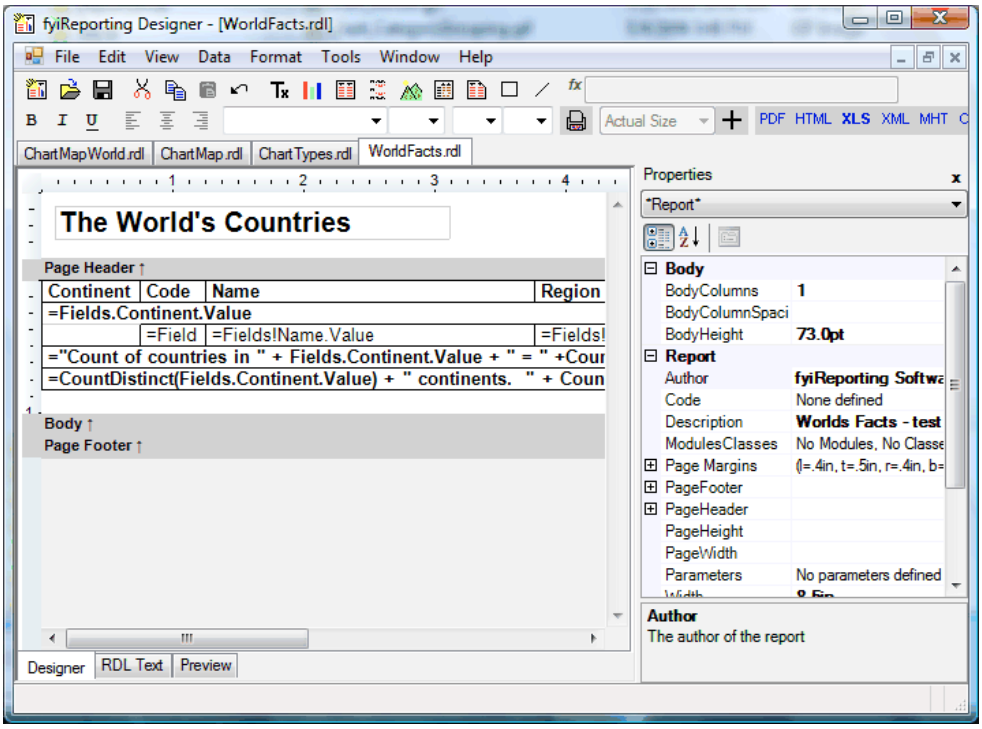

Figure 2.4: fyiReport Designer

of summaries or in-depth reports of key business information (i.e. balance sheets). It supports generating reports in a variety of formats including PDF, HTML, XML and BMP. It also has a *Query Engine* which provides access to databases and a number of data providers. The *Query Engine* includes a rich, extensible expression language which can be supplemented with application-specific functions. In figure 2.5 it can be seen the report designer.

2.2.3 Technologies Used in the SRDL Project

In this section, the technologies chosen for this project will be reviewed and their contribution to this project will be explained.

2.2.3.1 XML technologies

XML Schema is an XML technology standard developed by W3C and released in 2004. This technology has a single modeling language, which is very flexible and powerful, yet relatively simple to implement.. The XML Schema standard is namespace-sensitive, as it can be used to process documents with elements and attributes from different namespaces. Elements are used to define elements. These elements can have simple content or have a more complex one containing several elements [Bra02]. The main reason for using this

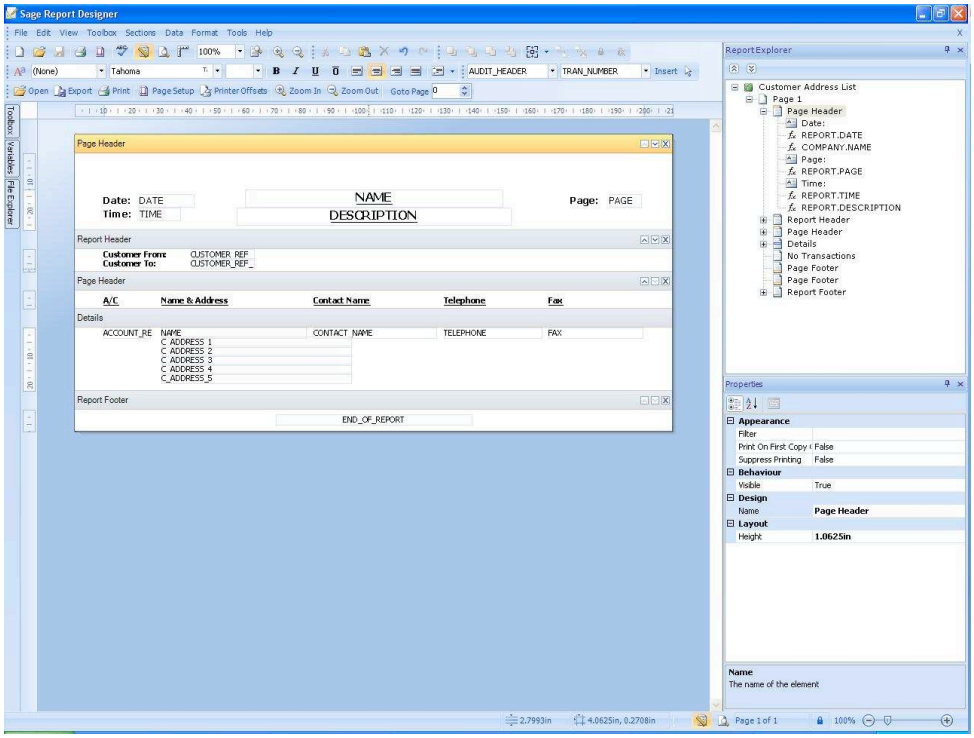

Figure 2.5: Visual Report from Sage Accouts

technology was that one of the most important objectives of this project is creating an XML language with a complex structure to define reports.

XSLT is a specification of a language for transforming XML documents into other XML documents. It is designed to be used as part of XSL, which is a style-sheet language for XML with a vocabulary for specifying formatting. "XSL specifies the styling of an XML document by using XSLT to describe how the document is transformed into another XML document that uses the formatting vocabulary."[W3C99b] These technologies were used because it was needed a process to convert from one language to another with a totally different structure.

2.2.3.2 C#.NET

C# is an objective oriented programing paradigm, adequate to the development of this project. This technology has a fast learning curve, so it was chosen due to the small amount of available time to finish this project (five months).

2.3 Conclusions

There are several technologies on the reporting area, each of them containing different designer and different file formats. After this investigation a decision was made to create a subset of *Micrsoft*'s RDL specification. Moreover, a process to convert *Sage Next*'s proprietary report definition file format to SRDL (Sage RDL), was going to be created. For printing or generating the reports, *List & Label* tool was chosen by *Sage*. Not only for commercial reasons but also because it had API functions, which could help the development of the reporting engine.

Sage Portugal intended to use *SAGE 50 Accounts 2009* reporting engine and designer, and convert *Sage Next*'s report definition files to this tool's file format. Unfortunately, there was not any documentation available about it and this tool was only found two months after the beginning of the project. Despite the fact that *Sage Portugal*'s intentions, it was decided to continue with the original planning.

Chapter 3

Report Making Architectures

In this chapter, the main architectural aspects of the project are going to be explained. To begin with, it will be described the current reporting architecture of *Sage Next*. Then, Sage's intended reporting process will be analyzed in order to identify its requirements and possible problems. Finally, it will be explained the possible reporting solution that was implemented.

3.1 Problem Analysis

Sage Next's current reporting model uses three different programs to generate reports. Each program needs a file containing the report's template and a connection with the database in order to generate the report. The first report generator used was Sage Map (section 2.2.1.1) that was created internally and was sufficient to Sage demands despite its problems and inefficiencies. With the appearing of *Crystal Reports* technologies, Sage started to create new report templates with it. The conversion of all the older report templates created with *Sage Map* was not an easy task due to huge differences between technologies. So, the only way would be to create from scratch the more than two thousand report templates.

With the lack of manpower and time to convert all the older reports, only the new ones were created, but it got worse when complaints from clients started to emerge. When clients wanted to change the template they had to use the *Crystal Reports* designer tool. Because the tool was not an user friendly application, a new reporting tool was chosen. *List & Label*'s solution had, in Sage's opinion, a much more user friendly design tool, so new report templates were now created with this tool.

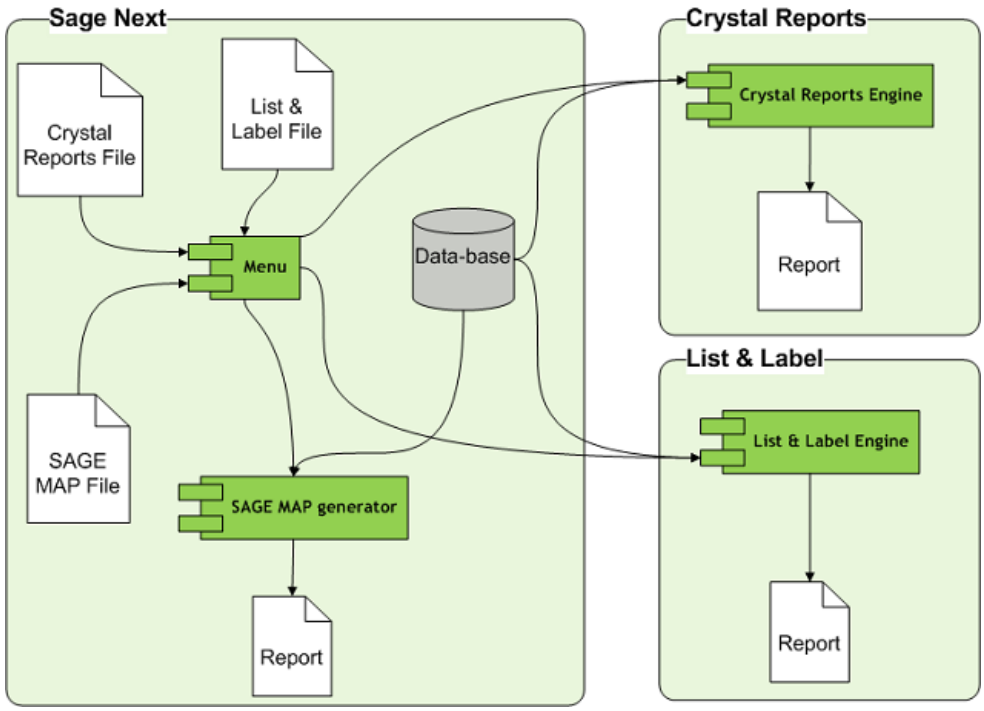

Figure 3.1: SAGE Next's Current Reporting Architecture

At this point, in *Sage Next* coexisted three different technologies for the same purpose and a new model for reporting was needed, which was the reason to start the project described in this dissertation.

3.1.1 Sage Next's current reporting architecture

As seen in figure 3.1, in the current architecture all three kinds of report templates formats are used to generate reports. The decision of what reporting tool to use when running *Sage Next*, is hard-coded in the application's menu. When a report is triggered by the menu, it calls the reporting engine to generate the report. Then, the engine will connect to the database and to get input data, and then output the generated report.

3.1.2 Intended Reporting Architecture

In order to unify the process, it is needed an intermediary between all three kinds of file formats and a reporting tool. The new reporting architecture (as seen in figure 3.2) contains a new file format for report templates (SRDL file). This new format consists of an XML file with a proper language capable of containing all the definitions for report creation. Due to *Sage*'s interests the data connection options were left aside from this new format. The new file is obtained through a conversion process by another component of the architecture using also XML technologies.

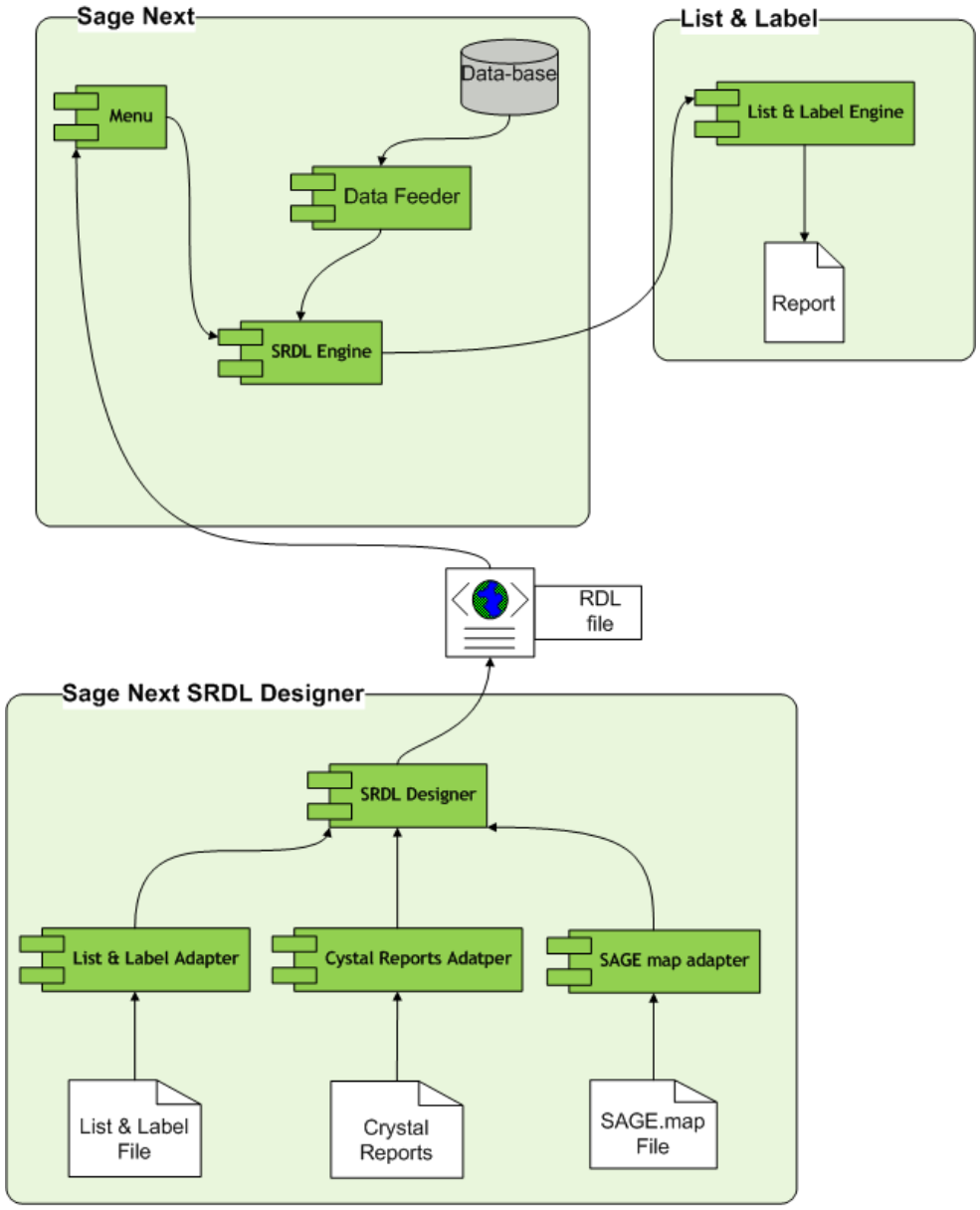

Figure 3.2: Intended Reporting Architecture

3.1.3 Requirements

In order for this new reporting architecture to successfully create reports, maintain the visual style of the old ones, some requirements were identified.

- An XML Language capable of defining all aspects of SAGE Next's Reports;
- A conversion process from the old format to the XML Language;
- An application to render the report.

The XML Language designed was based on *Microsft*'s RDL and will be described on the next chapter. For the language creation a XML Schema was developed for purposes of XML validation and to help creating new reports in the future. In this project only the oldest of the file formats (*Sage Map*) was chosen to convert to the new XML one. This was done due to time limitations and *Sage* priorities.

3.2 Methodology to be applied

In this section the major steps of the intended architecture and the technologies that will support them will be explained.

3.2.1 Converting to SRDL

The *Sage Map* file is filled with binary data structures so its contents are unreadable to ordinary people. To overcome this problem, a co-worker made a tool that could write the data in plain text. Now, with the data in a readable way, it is easier to process it and convert it into other formats. A possible converting process from a text file to an XML one with a totally different structure is to first convert to an XML file, maintaining the logical structure, and then change the language of the XML file. This process takes advantages of XSLT technology due to its capabilities to describe how an XML document is transformed into another one with different vocabulary or even changes in structure . The conversion process to SRDL will start with the creation of an XML schema based on Microsoft's RDL and then develop a XSLT file. The schema will contain all the elements necessary to generate reports similar to *Sage Next*'s ones. The XSLT must define all the rules to convert the first XML file to SRDL. To make this process possible an application is needed, like *Saxon*, to apply the transformation between XML languages.

3.2.2 Generating the Report

The initial part of the process consisted in parsing the generated SRDL file, now it must be used a technology capable of generating reports. To do this, it is needed a report generating tool (like *List & Label*'s) with an API, and a programable language (like C#.NET)

to develop an application. In this application, the API functions will open the report tool's designer and then the final report. While the generating application is being developed, some changes will have to be made to the SRDL in order to maintain the former report's visual style. The testing phase will start without accessing any database, since the database component will only be made after the generating component is done by a *Sage* coworker.

3.3 Summary and conclusions

This chapter, it can be verified that *Sage Portugal* had major problems in *Sage Next*'s reporting processes. Sage wanted to unify the reporting technologies and in this process get rid of *Sage Map*.

To improve reporting processes it is needed a RDL (Report Definition Language) and a reporting tool with a usable API. The technologies that are going to be used are: XML technologies (XSLT and XML schema), C#.NET and *List & Label*. A new architecture will be developed to meet Sage's demands and accurately improve the reporting processes. Next chapters will explain it in detail.

Chapter 4

Implementation

In chapter 4 a deepen study of the developed architecture will be presented. First, all of the components of the architecture will be described and how they were integrated. After that, the main features of the created RDL (SRDL) will be analyzed.

4.1 Project's Architecture

In figure 4.1 it can be seen the final architecture that was implemented to improve the report generating process. This architecture is similar to the desired one by sage as seen in figure 3.2. The Sage Map adapter, seen on figure 3.2, is composed of two components on figure 4.1 (Report Defintion Data Dumper and Text to XML) wich will be described later on this chapter. The SRDL Designer seen on figure 3.2 is composed by the XML to SRDL component wich is seen on figure 4.1. The components SRDL Engine and List & Label engine are the same on both figures 3.2 and 4.1.

The developed architecture is composed by six components: four applications, a report generating tool and a XML transformation tool. Part of these components were made by a Sage coworker which can be seen on the left side of figure 4.1 (Report Definition Data Dumper and the Data Feeder). On the right side of the picture it can be seen the other main components of the project that will later be analyzed: Text to XML, XML to RDL, SRDL Engine, *List & Label* Engine. As seen on figure 4.1, on the new reporting engine the Report Definition Data Dumper outputs a text file which will be then parsed and transformed to XML. The Saxon application will then transform it into SRDL. Finally the SRDL engine will parse the SRDL file and receive data from the Data Feeder component. After that, it will output the report with *List & Label*'s API functions. Now all the components of figure 4.1 will explained one by one.

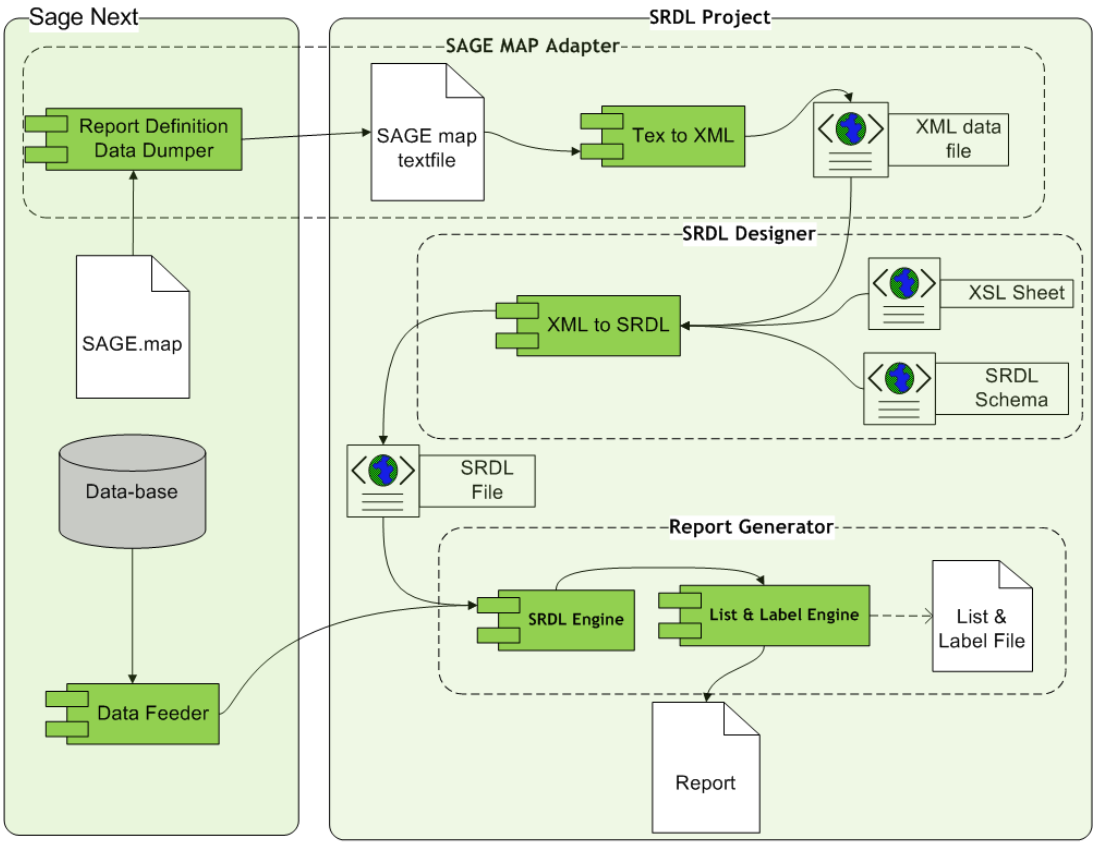

Figure 4.1: SRDL Project Architecture

4.1.1 Report Description Data Dumper

This component, made by a *Sage* co-worker in VB (Visual Basic), is embedded in *Sage Next* and generates text files from the *Sage Map* files. These text files contain the description of all the composing parts of reports: general settings, parameters, headers, footers, tables and text-boxes. The text file does not contain any type of data or its location, it only has enough information to generate the report's layout. The data definitions were not included and they were only managed by the Data Feeder component. This was done because *Sage Next* uses different types of databases and connections and dealing with them directly on the report's templates usually lead to connection errors.

4.1.2 Text to XML

This component is an application made in C#.Net that parses the text file generated by the Report Description Data Dumper, embedded in *Sage Next*, and converts it into an XML file, maintaining its logical structure as it can be seen below.

The text file's structure example:

```
[Geral]
Altura=0
AlturaLinhas=0
ComoOrdenar=0
CondAcumular=
CondArea 1=
...
[Pedidos]
[Pedido 1]
Campo=1
ComoPedir=0
...
[Pedido 2]
Campo=76
ComoPedir=0
...
```

The XML file's structure example

```
<Report>
  <Geral>
    <Altura>0</Altura>
    <AlturaLinhas>0</AlturaLinhas>
```

```
    <ComoOrdenar>0</ComoOrdenar>
      ...
  </Geral>
  <Pedidos>
    <Pedido id="1" >
      <Campo>1</Campo>
      <ComoPedir>0</ComoPedir>
    ...
    </Pedido>
    <Pedido id="2" >
      <Campo>76</Campo>
      <ComoPedir>0</ComoPedir>
        ...
    </Pedido>
   ...
  </Pedidos>
 ...
</Report>
```

This was done to enable a more comprehensive study of *Sage Map* file structure and to allow the use of XML technologies, such as XSL which transforms XML documents from one dialect to another.

4.1.3 XML to SRDL with Saxon

This component is a batch file with instructions to call an application named Saxon for the transformation process. This process consists on transforming the first XML file into the XML file in SRDL, using the created XSL sheet and the SRDL schema. The SRDL schema is an XML file which describes the structure of an RDL and it will be analysed further in this chapter. For the transformation between XML languages, the Open Source SAXON XSLT processor, was chosen instead of C# because the it was not found a class implementing support xQuery functions. These functions are used in the transformation process and are crucial to its functioning.

4.1.4 SRDL Engine/Report Generator

This C# .Net based application parses de XML file, calls *List & Label* API functions and opens its report designer with the layout already created .

The C# .Net technology was used because *List & Label* has an API written in C # and the technology has a large library of pre-coded solutions to common programming problems like working with linked lists or memory management.

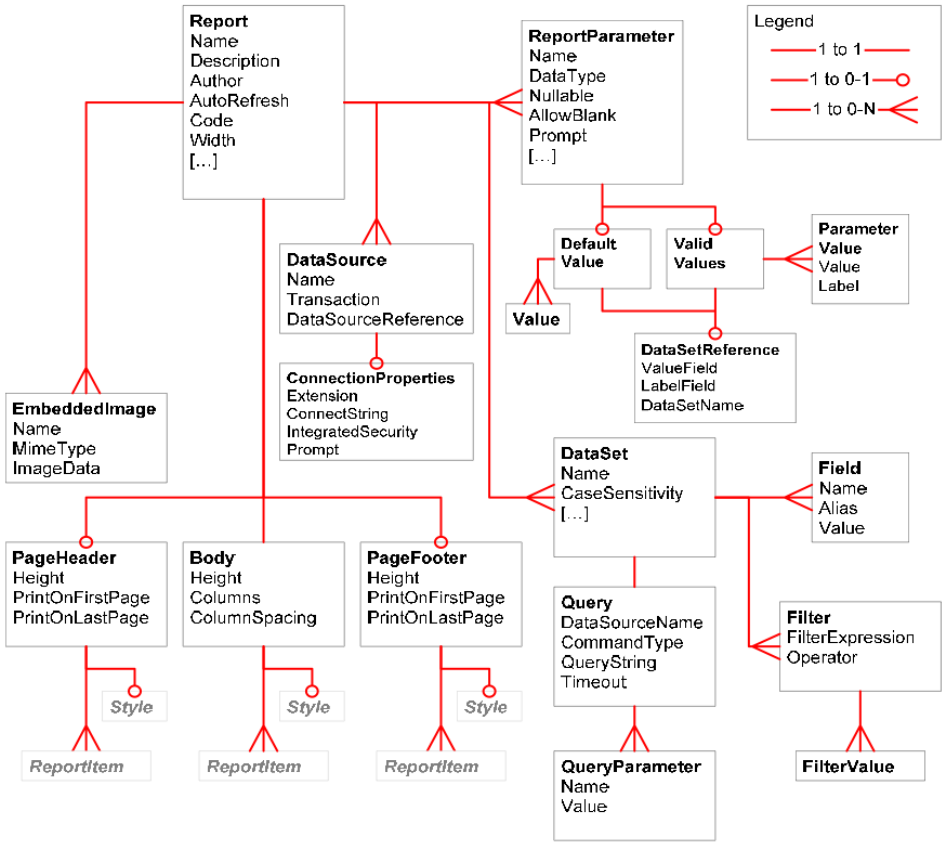

Figure 4.2: SRDL Schema structure

For the final part of the report generating process, data from *Sage Next*'s database needs to be injected into the report layout in order to fill the report tables with information. This is done by the Data Feeder.

4.1.5 Data Feeder

The Data Feeder works as an interface between *Sage Next*'s database and the SRDL Engine. This part of the project was not finished due to *Sage*'s business priorities. Data sets were manually created for the case study to feed the SRDL Engine.

4.2 SRDL's Architecture

Like said before, the SRDL was based on *Microsoft*'s RDL and the schema can be seen on figure 4.2.

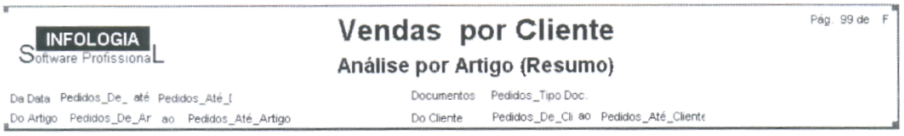

Figure 4.3: Header Example

4.2.1 SRDL Schema Structure

A report is mainly composed by three parts: header, body and footer. The headers and the footers contain informations about the report like the title, the parameters, the name of the company and page count. The body contains a table with columns on which the data will be shown.

In RDL there is an important element which is the *ReportItem*. Each report item is used to describe every atomic part of reports as they can specify a rectangle, a text box, a line, an image or a data region.Besides that a report item can have another report item inside, like a rectangle that has several text boxes or images.

On figure 4.3, an example of a report header with one image an some text boxes can be seen. In RDL the header would look like the sample code below.

```
<PageHeader>
    <Height>1725 px</Height>
    <ReportItems>
      <Rectangle>
        <Name>PageHeader</Name>
         (...)
         <ReportItems>
          <TextBox>
            <Name>0,1</Name>
            <Top>105 px</Top>
            <Left>90 px</Left>
            <Height>915 px</Height>
            <Width>2325 px</Width>
            <Style>
              (...)
              <BorderStyle>
                <Left>None</Left>
                <Right>None</Right>
                <Top>None</Top>
                <Bottom>None</Bottom>
```

```
                </BorderStyle>
                <TextAlign>Center</TextAlign>
              </Style>
              <Value>@IINF</Value>
            </TextBox>
            (...)
             <TextBox>
              (...)
              <Value>Dados da Empresa_Nome</Value>
            </TextBox>
          </ReportItems>
        </Rectangle>
      </ReportItems>
    </PageHeader>
```

4.2.2 SRDL XSL transformation process

The XSL Sheet is an XML file with instructions to transform the XML file generated from the Text to XML component to an XML file with the SRDL language. The process involves using xPath queries to navigate the XML file and XSL functions for string comparison. An example of choosing the report's orientation by querying the first XML file and comparing string, can be seen below.

```
<Report  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="SRDL.xsd">
       <Description>
       <xsl:value-of select="Report/Geral/Titulo/node()"/>
       <xsl:text> -
       </xsl:text><xsl:value-of select="Report/Geral/SubTitulo/node()
       </xsl:value-of>
       </Description>
       <Orientation>
       <xsl:choose><xsl:when test="Report/Geral/Portrait=0 ">
       Landscape
       </xsl:when>
           <xsl:otherwise>Portrait</xsl:otherwise>
           </xsl:choose>
       </Orientation>
       <ReportParameters>
       (...)
```

```
        </ReportParameters>
        (...)
</Report>
```

4.3 Conclusions

In this chapter it was seen how the components of the architecture were integrated and implemented. It can be concluded that the developed RDL can potentially define all aspects of reports. Furthermore, this architecture will provide a successful migration of the old reports on an automated way and in less time, without losing their visual style. In the next chapter, the proof of concept will be presented with an example of a conversion of an old report to a new one.

Chapter 5

Case Study

In this chapter it will be presented an example of a conversion and generation of a report. Then, it will be verified the level of satisfaction of the obtained results.

5.1 Input Data

To test the developed architecture, four reports were converted and generated. In figure 5.1 it can be seen one of them generated by *Sage Map* reporting application.

To convert one report to SRDL, first is necessary to get the data that defines the report's template layout from *Sage Next*. To do this, it is used one component from the created architecture: the Report Definition Data Dumper. This component creates a text file with that data in a text format, so it can be read and used by other components. The first lines of the created file can be seen on the next page (the full file has more than 5000 lines).

sage Next Prime/Vision

Balancete Razão

Pág. 1 de 1

Período

Exercício 2007 — Nº Contribuinte 599999999

Período Setembro — Tipo de Saldo Saldo das Somas — Gestão Comercial/Administrativa - Demonstração

Conta	Descrição	Período		Saldos	
		Débito	Crédito	Débito	Crédito
11	CAIXA	6.112,00	20.001,00		13.889,00
21	CLIENTES	2.768,44		2.768,44	
22	FORNECEDORES	12.500,00		12.500,00	
24	ESTADO E OUTROS ENTES PÚBLICOS		2.649,73		2.649,73
42	IMOBILIZAÇÕES CORPÓREAS	15.000,00	11.441,58	3.558,42	
71	VENDAS		2.288,13		2.288,13
Total		**36.380,44**	**36.380,44**	**18.826,86**	**18.826,86**

Figure 5.1: Sage Next Report

```
[Geral]
(...)
SubTitulo=Período
Titulo=Balancete Razão
(...)
[Areas]
[Area 0]
B:
   Baixo=1
   Cima=1
   Direita=1
   Esquerda=1
BDasLinhas=-2
CorL=0
Invisivel=0
Fonte=1
Q:
   x1=0
   x2=10845
   y1=15
   y2=1755
Quantos=9
[Area 1]
(...)
[Area 2]
(...)
```

Now this input data will pass through all the steps of the transformation to generate a report.

5.2 Transformation Process

The transformation process begins with converting the text file to XML and then to SRDL. This is done by two components (Text to XML and XML to SRDL) in a consecutively way outputting an SRDL file ready to be used to generate a report or to be saved for further use. In figure 5.2 there can be seen some lines of the generated SRDL file.

```
− <Report>
    <Description>Balancete Razão - Periodo</Description>
    <Orientation>Portrait</Orientation>
  + <ReportParameters></ReportParameters>
  + <DataSets></DataSets>
  + <PageHeader></PageHeader>
  − <Body>
    − <ReportItems>
      + <Rectangle></Rectangle>
      − <Rectangle>
          <Name>Details</Name>
          <Top>75 px</Top>
          <Left>0 px</Left>
          <Height>225 px</Height>
          <Width>11145 px</Width>
        + <Visibility></Visibility>
        + <Style></Style>
        − <ReportItems>
          − <Matrix>
              <Name>Details</Name>
            + <MatrixRows></MatrixRows>
            + <MatrixColumns></MatrixColumns>
            </Matrix>
          </ReportItems>
        </Rectangle>
      + <Rectangle></Rectangle>
      </ReportItems>
    </Body>
  + <PageFooter></PageFooter>
  </Report>
```

Figure 5.2: SRDL file

To generate the final output, data is needed to fill the report. The Data Feeder component was not finished, so data sets were manually created in XML to feed data to the report generator. In figure 5.3 it can be seen some lines of the file with the data. Having the data and the RDL file ready it can now be tested one generation of a report.

5.3 Output results

In figure 5.4 it can be seen the developed application's interface that can output report. First it is selected a SRDL file and then, clicking the "Design/Print" button, the report will appear in *List & Label* viewer. The data sets containing the report's data must be in the same directory of the SRDL file. In figure 5.5 the viewer with the generated report can be seen. The *List & Label* viewer allows exporting the report in several formats like PDF or HTML.

5.4 Conclusions

Comparing both figures 5.1 and 5.6, a significant resemblance can be seen only with minor differences. These differences were expected due to the use of different reporting technologies, but they are insignificant because report's layout is kept the same. It can be concluded that the change of technologies did not affect the report, also the conversion process to the new format is a lot faster than manually creating reports from scratch.

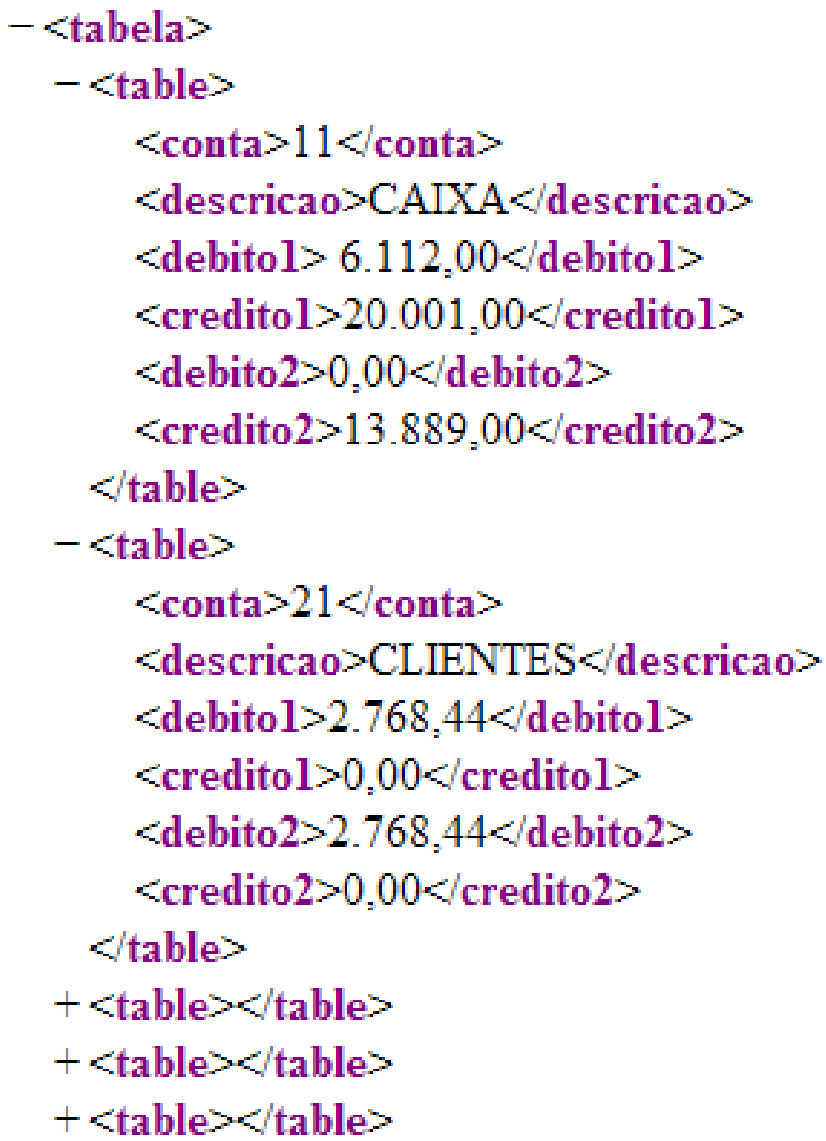

Figure 5.3: XML dataset

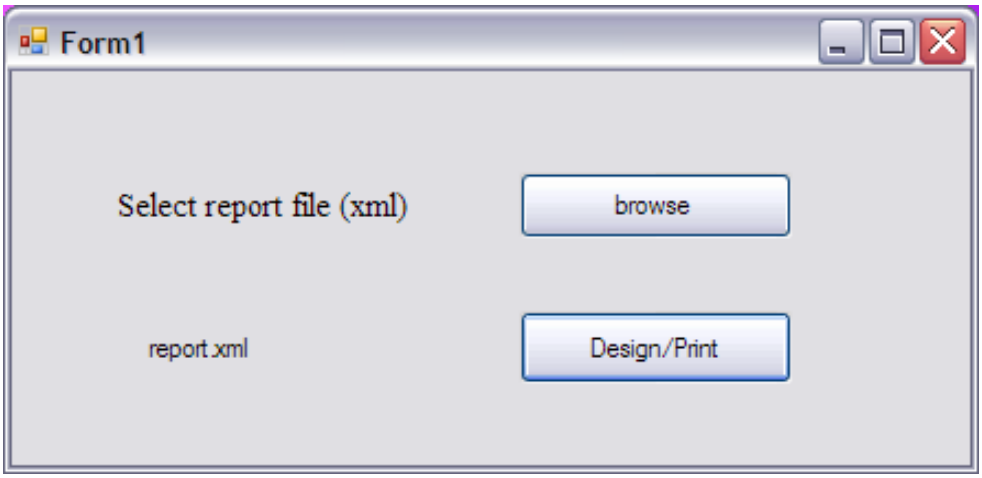

Figure 5.4: SRDL Engine Interface

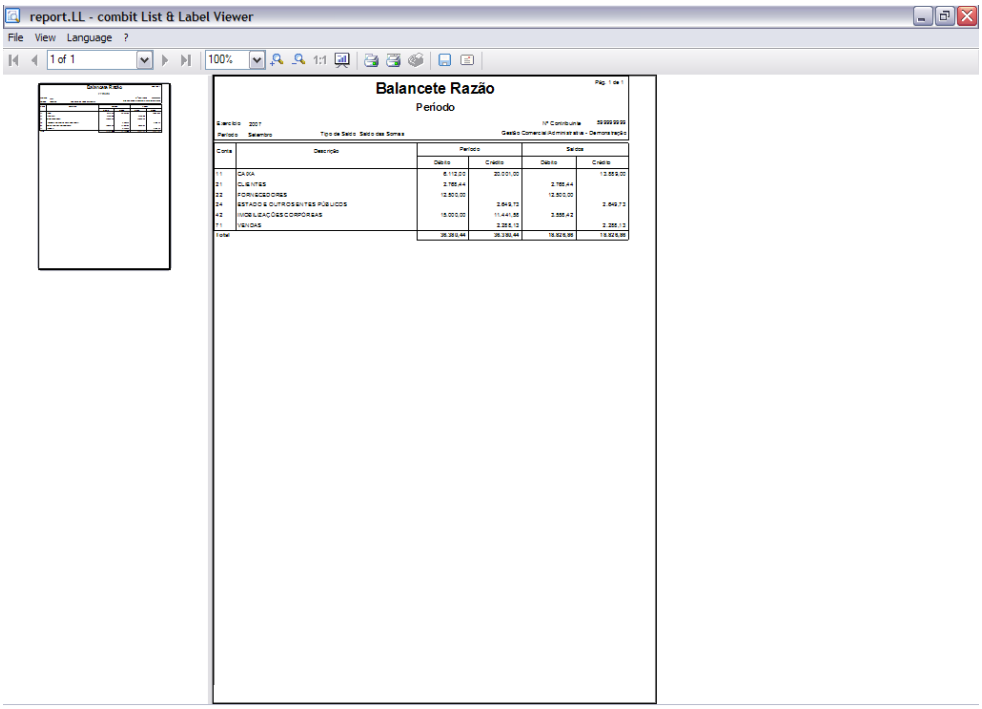

Figure 5.5: List & Label Viewer

Pág. 1 de 1

Balancete Razão

Período

Exercício 2007 — Nº Contribuinte 599999999

Período Setembro — Tipo de Saldo Saldo das Somas — Gestão Comercial/Administrativa - Demonstração

Conta	Descrição	Período		Saldos	
		Débito	Crédito	Débito	Crédito
11	CAIXA	6.112,00	20.001,00		13.889,00
21	CLIENTES	2.768,44		2.768,44	
22	FORNECEDORES	12.500,00		12.500,00	
24	ESTADO E OUTROS ENTES PÚBLICOS		2.649,73		2.649,73
42	IMOBILIZAÇÕES CORPÓREAS	15.000,00	11.441,58	3.558,42	
71	VENDAS		2.288,13		2.288,13
Total		**36.380,44**	**36.380,44**	**18.826,86**	**18.826,86**

Figure 5.6: SRDL Report

Chapter 6

Conclusions and Future Work

This project showed that it is possible to successfully use an XML language and associated technologies to describe database reports. The use of XML technologies will provide several improvements in *Sage Next* application. In this chapter all the conclusions obtained during the development of this project will be shown and explained. First, an analysis on the objectives's accomplishment will be made. The parts of this project which were not finished will be explained, showing what future work this project will need.

6.1 Project Objectives Accomplishment

The main objective of this project was to create an XML language capable to define reports. To prove this, an old internal file format of one of *Sage* was chosen to covert to XML. To prove the successful conversion, and that the XML language was complete, a reporting tool was chosen to generate the reports.

The development of the XML language (called SRDL) was the first part of this project to be considered finished. A complete XML schema based on *Microsoft*'s RDL was successfully developed and ready for testing. The next part of the project to be concluded was the process in which the files from *Sage Map* application were converted to SRDL using XSLT transformation sheets. The XSLT was finished and tested successfully in the four conversions that were accomplished. The SRDL report generator component was developed and the four previous SRDL files were used successfully with manually created test data. The Data Feeder component will feed the report generator with real data and then more tests will be performed.

In all four reports the results showed successful results. Old and new reports had great resemblance, only showing minor differences which could only be noticed by a trained eye.

6.2 Future Work

This project provided *Sage* with a new reporting architecture and process. Furthermore the company will have now the ability to have its reports represented in a format inter-operable, allowing for future usages. The Data Feeder component which can be seen on figure 4.1 has not been completely finished due to *Sage Portugal*'s commercial priorities, as said before in section 4.1.5. The next step in this project is to conclude this component in order to make more testing.

In the future this project will be integrated in *Sage Next* architecture enabling the ability to test the SRDL Engine with real data, and stop using of *Sage Map* old technology to output reports.

References

[Bra02] N. Bradley. *The Xml Companion*. Addison-Wesley Professional, 2002.

[Cor05] Microsoft Corporation. Report Definition Language Specification, November 2005. Available in http://download.microsoft.com/download/6/5/7/6575f1c8-4607-48d2-941d-c69622e11c32/RDL_spec_08.pdf, Last accessed in February 2009.

[CRY09] CRYSTALREPORTS. Crystalreports.com, 2009. Available in http://www.crystalreports.com/, Last accessed in February 2009.

[cSG09] combit Software GmbH. Reporting tool list & label, 2009. Available in http://www.combit.net/en/reporting-tool/report-generator-List-Label, Last accessed in February 2009.

[fS09] fyiReporting Software. fyireporting software, 2009. Available in http://www.fyireporting.com, Last accessed in February 2009.

[Kay08] Michael Kay. Saxon the xslt and xquery processor, December 2008. Available in http://saxon.sourceforge.net, Last accessed in February 2009.

[Lau06] Laudon & Laudon. *Management Information Systems - The Digital Firm*. Prentice Hall, 2006.

[Pec04] George Peck. *Crystal Reports 10 The Complete Reference*. McGraw-Hill/Osborne, 2004.

[Say07] Asif Sayed. *Client-Side Reporting with Visual Studio in C Sharp*. Apress, 2007.

[W3C99a] W3C. Xml path language, November 1999. Available in http://www.w3.org/TR/xpath, Last accessed in February 2009.

[W3C99b] W3C. Xsl transformations, November 1999. Available in http://www.w3.org/TR/xslt, Last accessed in February 2009.

[W3C04] W3C. Xml schema, October 2004. Available in http://www.w3.org/TR/xmlschema-0/, Last accessed in February 2009.