FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FEUP

Operations Research Modeling Language for an ERP System

André Miguel Coelho de Oliveira Rodrigues

Project Report
Master in Informatics and Computing Engineering

Supervisor: Luis Paulo Gonçalves dos Reis (Auxiliar Professor)

2008, July

© André Rodrigues, 2008

Operations Research Modeling Language for an ERP System

André Miguel Coelho de Oliveira Rodrigues

Project Report

Master in Informatics and Computing Engineering

Approved in oral examination by the committee:

Chair: João Carlos Pascoal de Faria (Auxiliar Professor)

____

External Examiner: Per Vikkelsøe (Development Lead II)

Internal Examiner: Luis Paulo Gonçalves dos Reis (Auxiliar Professor)

31^{st} July, 2008

Abstract

Long are the times where businesses had to collect data from non-automated sources and lacked the computing resources necessary to analyze and make decisions based on it.
Enterprise Resource Planning systems integrate several data sources and process them, greatly improving what is commonly referred as Business Intelligence. Its purpose is to collect, integrate, analyze and present data in order to support better business decision-making.
Microsoft is currently on the ERP business with, among others, Microsoft Dynamics AX. The product covers a wide group of areas that range from Customer Relationship Management (CRM) to production, human resources and more.
Current ERP systems still, however, face some problems, being one of the most important ones the challenge of developing, deploying and maintaining customized solutions to customers. This is pointed out as one of the most important factors in these systems and having a flexible and easy to use way of doing it is increasingly becoming a key factor when it comes to choose one over another.
The usage of mathematical models to describe some of the most common optimization and decision-making problems has been, for long, used and current systems use state-of-the-art solvers to approach it.
Partners and end-user customers haven't, however, always been empowered with an easy way of accessing such technologies and further customize the system. Not only because of code openness and copyrights but also because there isn't usually any easy way of doing it.
Mathematical Modeling languages have been used for long and attempt to abstract away the implementation details of the underlying solver technologies and allow users to focus on the modeling part. There wasn't, however, a clear attempt to empower an ERP solution, right out-of-the-box, with such technologies, in a fully integrated way with its data sources and business logic.
This report introduces this problem that spawned a quest for a solution, one that this project encompassed for a concrete product - Microsoft Dynamics AX. It explores some of the various solver engine solutions and modeling languages available in the market today. It also goes beyond it by studying interchangeable document formats and overall ways of interacting with Microsoft Dynamics AX. It presents a new modeling environment, based on a modeling language, exposing the necessary requirements, while giving concrete examples of its usage in addressing real optimization problems in this system.
Design and implementation details are also given, with a special focus on the interpreter implementation, which made use of the MPLEX/MPPG tools, the Microsoft Math SDK, the OpenXML SDK and the Microsoft Solver Foundation, as well as, the services it provides and its relation with the IDE project which has been tightly coupled with this project.

It then evaluates the overall solution by presenting some methodologies and tests that ensured a good project outcome, within the defined scope and quality levels, and outlining how it addresses the previously problems. Some conclusions are then drawn on how the users perceive the system and how could it be further expanded and improved by adding mechanisms like debugging tools or solver independence.

Resumo

Longe vão os tempos em que empresas tinham de recolher dados de forma manual e não dispunham dos recursos computacionais necessários para os analizar e, com base nisso, tomar decisões.
Os sistemas ERP integram e processam diversas fontes de dados, introduzindo inúmeras melhorias no que se costuma designar de *Business Intelligence*. O seu propósito é recolher, integrar, analisar e apresentar informação de forma a suportar a tomada de melhores decisões de negócio.
A Microsoft está, actualmente, presente no negócio dos ERP com, entre outros, o Microsoft Dynamics AX. Este produto cobre um grupo alargado de áreas que vão desde gestão de clientes (CRM) até produção, passando pela gestão de recursos humanos e mais.
Os sistemas actuais, continuam, no entanto, a ter alguns problemas, sendo um dos mais importantes a necessidade de desenvolver, implementar e manter solucções costumizadas para os clientes. Este é, de facto, um dos mais importantes factores nestes sistemas e ter uma maneira fácil e fléxivel de o fazer está, gradualmente, a tornar-se num factor chave no momento da decisão por um em detrenimento de outro.
A utilização de modelos matemáticos para descrever alguns dos problemas de optimização e decisão mais comuns tem ganho crescente aceitação e alguns ERPs usam aplicações de optimização de ponta para os resolver.
Parceiros e clientes finais não têm, contudo, sempre sido empossados com uma maneira de aceder a tais tecnologias e extender, eles mesmos, a costumização do sistema. Não só por questões de abertura de código e legislação, mas também porque não existe, habitualmente, uma maneira de o fazer.
As linguagens de modelação matemática têm sido usadas, desde há muito, para abstrair os detalhes de implementação das tecnologias que lhes estao subjacentes permitindo aos utilizadores focarem-se na modelação. Não existiu, contudo, uma clara tentativa de implementar, de origem, numa solução ERP tais tecnologias de uma maneira totalmente integrada com as suas fontes de dados e lógica de negócio.
Este relatório introduz este problema e extende-se numa procura por uma solução, uma que esteja orientada a um produto especifico - o Microsoft Dynamics AX. Explora algumas das diferentes soluções existentes no mercado em aplicações de optimização e linguagens de modelação e vai mais alêm estudando formatos de representação de tais modelos e as diferentes maneiras de interair com o Microsoft Dynamics AX. Apresenta, de seguida, um novo ambiente de modelação, expondo os requisitos necessários e apresentando exemplos concretos da sua utilização em alguns dos probemas de optimzação do Dynamics AX.
Detalhes de implementação são também dados, com um especial focus na implementação do interpretador, que fez uso das ferramentas MPLEX/MPPG e da recente anunciada

Framework Microsoft Solver Foundation, assim como, dos serviços que implementa e na relacção que mantêm com o projecto de um IDE que está intimamente ligado a este projecto.
Avalia, de seguida, a solução apresentando algumas das metodologias e testes que foram usados para garantir o sucesso do projecto, dentro do seu âmbito e niveis de qualidade desejados, salientado como ataca os problemas referidos.
Algumas conclusões são, por fim, retiradas sobre a forma como os utilizadores percepcionam o sistema e como este poderia ser expandido e melhorado pela adicção de mecanismos como ferramentas de depuração e independência da tecnologia de resolução.

Acknowledgements

First and foremost I would like to thank everyone at Microsoft for making this traineeship possible. Special thanks go to Hans Jorgen Skovgaard, Per Vikkelsøe, Laurent Ricci and all the other guys who interviewed me and actually opened me the gates. I would also like to express my gratitude to André Lamego, João Magalhães and Tiago Silva for guiding me during the first months in the company - you guys rock!
Thanks, also, to IAESTE Denmark for the help with the logistical and legal aspects of the reallocation.
From FEUP I would like to thank Raul Vidal and AlumniLEIC for creating the bridge between Microsoft and the University that made the interviewing process, which culminated with this traineeship, possible. I would also like to thank Luis Paulo Gonçalves dos Reis, my project supervisor and João Pascoal Faria for their guidance, advices and great share of knowledge throughout the project but, most of all, for their interest and time devotion to this project. A word of gratitude also goes to António Augusto Sousa, my program's director, for all the help, support and interest in all the burocratic process that allowed us to do the project.
A special mention to my friends in Delft, where I've spent, before coming here, 5 months studying through the Erasmus program, and with whom I've experienced my first period of living abroad and who tough me so much.
My thanks also go to Raquel Cristóvão, with whom I've shared my life during most of my faculty years and that helped me through all the way. For this, and so much more, you'll always be in my heart.
Last, but not least, I wish to thank my parents Antides Santo and Maria dos Anjos Dixe and my (best of the world) sister Irene Dixe for their every-day love and support, even when separated by thousands of kilometers.

André Rodrigues

Contents

1 Introduction **1**
1.1 About Microsoft . . . 1
1.1.1 The Corporation . . . 1
1.1.2 Microsoft Dynamics . . . 2
1.1.3 Microsoft Development Center Copenhagen . . . 3
1.2 Microsoft Dynamics AX . . . 3
1.2.1 Optimization Problems in Dynamics AX . . . 3
1.3 Operations Research . . . 4
1.4 Motivation and Objectives . . . 5
1.5 Report Overview . . . 6

2 Problem Description **7**
2.1 Declarative Optimization Language for an ERP System . . . 7
2.2 Target Personas . . . 8
2.3 Optimization Problems in Microsoft Dynamics AX . . . 9
2.3.1 Traveling Salesman . . . 9
2.3.2 Warehouse Picking Routes . . . 10
2.3.3 Production Scheduling . . . 12
2.4 Project Requirements . . . 12
2.4.1 ORML Interpreter . . . 14
2.4.2 Common Libraries . . . 15
2.5 Schedule and Deliverables . . . 16
2.6 Summary . . . 18

3 State of the Art **21**
3.1 Microsoft Dynamics AX . . . 21
3.1.1 Application Model Layering . . . 22
3.1.2 The X++ Programming Language . . . 23
3.1.3 Data Sources . . . 23
3.1.4 Integration . . . 24
3.2 Optimization Problems . . . 25
3.2.1 Linear Programming . . . 25
3.2.2 Integer Programming . . . 26
3.2.3 Constraint Satisfaction Programming . . . 26
3.3 Optimization Engines . . . 26
3.3.1 ILog CPLEX . . . 26
3.3.2 Microsoft Solver Foundation . . . 27

3.4 Mathematical Modeling Languages 28
3.4.1 OPL 6.0 29
3.4.2 MPL 4.2 30
3.4.3 AIMMS 3.8 33
3.4.4 AMPL 35
3.4.5 Language Comparison 36
3.5 Optimization Problems Formulation 37
3.5.1 Traveling Salesman 37
3.5.2 Warehouse Picking Routes 38
3.5.3 Production Scheduling 39
3.6 Mathematical Document Formats 41
3.6.1 Mathematical Markup Language 41
3.6.1.1 Microsoft Math 41
3.6.2 Office Math Markup Language 42
3.6.2.1 Open XML Formats SDK 43
3.7 .NET Framework 43
3.7.1 Programming Languages 44
3.8 Language Processors 45
3.8.1 Compiler Compiler Tools 47
3.9 Summary 49

4 Solution Specification **51**
4.1 Operations Research Modeling Language Specification 51
4.1.1 Language Overview 51
4.1.2 Grammars 53
4.1.2.1 Lexical Grammar 54
4.1.2.2 Grammar Rules 55
4.2 Modeling Optimization Problems in Microsoft Dynamics AX 61
4.2.1 Traveling Salesman 62
4.2.2 Warehouse Picking Routes 63
4.3 Summary 64

5 Design and Implementation **67**
5.1 Design 67
5.1.1 ORML Interpreter 68
5.1.1.1 Logical View 69
5.1.1.2 Development View 76
5.1.1.3 Process View 77
5.1.1.4 Physical View 79
5.1.2 ORML Interpreter Services 79
5.1.2.1 Syntax Highlight Service 79
5.1.2.2 Autocomplete Service 80
5.1.2.3 Export to MathML and OMML Service 80
5.1.3 Common Libraries 81
5.1.3.1 Common Data Layer 81
5.1.3.2 Model Management Layer 82
5.2 Implementation 83

5.2.1 ORML Interpreter . . . 83
5.2.1.1 Structures . . . 84
5.2.1.2 Lexical and Syntax Analyzers . . . 86
5.2.1.3 Semantical Analyzer and Interpretation Engine . . . 87
5.2.1.4 Interpreter Interface . . . 89
5.2.1.5 Interpreter AxWrapper . . . 90
5.2.2 ORML Interpreter Services . . . 90
5.2.2.1 Syntax Highlight Service . . . 90
5.2.2.2 Autocomplete Service . . . 91
5.2.2.3 Export to MathML and OMML Service . . . 91
5.2.3 Common Libraries . . . 92
5.2.3.1 Common Data Layer . . . 92
5.2.3.2 Model Management Layer . . . 92
5.3 Development Methodologies . . . 93
5.4 Summary . . . 96

6 Evaluation of the Solution 97
6.1 Testing . . . 97
6.1.1 Test Scope . . . 97
6.1.1.1 ORML Interpreter . . . 98
6.1.1.2 ORML Interpreter Services . . . 99
6.1.1.3 Common Libraries . . . 99
6.1.2 Test Strategy . . . 99
6.1.2.1 Testing Procedures . . . 99
6.1.2.2 Test Tools . . . 100
6.1.3 Test Resources . . . 100
6.1.4 Test Results . . . 101
6.1.5 ORML Interpreter . . . 101
6.1.6 ORML Interpreter Services . . . 103
6.1.7 Common Libraries . . . 103
6.2 Market Requirements Analysis . . . 103
6.3 Modeling Experience . . . 104
6.4 Summary . . . 104

7 Conclusions and Future Work 107
7.1 Success Evaluation . . . 107
7.2 Conclusions . . . 108
7.3 Originalities . . . 109
7.4 Limitations . . . 109
7.5 Project Continuum . . . 110

A Survey Questions 119

B Project Requirements 127

C Project Schedule 131

D Microsoft Math Supported Formats 133

E Language Definition **135**

F Modeling Optimization Problems in Microsoft Dynamics AX **143**
F.1 Traveling Salesman 143
F.2 Warehouse Picking Routes 144

G Computer Specifications **145**
G.1 Quad-core processor desktop 145
G.2 Single-core processor laptop 145

H ORML Models **147**
H.1 Zebra 147
H.2 Boeing 148
H.3 PetroChem 149
H.4 WycoDoors 150
H.5 Traveling Salesman 151
H.6 Warehouse Picking Routes 153

List of Figures

2.1 TSP Entity-Relationship Diagram 10
2.2 Warehouse Environment 10
2.3 Warehouse Entity-Relationship Diagram 11
2.4 Interpreter Interface - Use Cases 14
2.5 Console Interpreter - Use Cases 15

3.1 Dynamics AX Rich Client 22
3.2 Dynamics AX Application Model Layers 22
3.3 Sample X++ Code 23
3.4 MSF High Level Architecture 27
3.5 OPL - Spreadsheet Reading Example 30
3.6 OPL - Database Reading Example 30
3.7 MPL - Model Example 32
3.8 MPL - Text File Reading Example 32
3.9 MPL - Spreadsheet Reading Example 33
3.10 MPL - Database Reading Example 33
3.11 MPL - Spreadsheet Reading Example with Filtering 33
3.12 AIMMS - Spreadsheet Reading Example 35
3.13 AIMMS - Database Reading Example 35
3.14 AMPL - Text File Reading Example 36
3.15 AMPL - Spreadsheet Reading Example 36
3.16 AMPL - Database Reading Example 36
3.17 Microsoft Math 42
3.18 .NET Framework 3.5 Architecture 44
3.19 Compiler Front End Model 47
3.20 Lex File Structure 48
3.21 Yacc File Structure 48

4.1 ORML Model - Structure 56
4.2 ORML Model - Index Section Example 57
4.3 ORML Model - Input Section Example 58
4.4 ORML Model - Variables Section Example 59
4.5 ORML Model - Functions Section Example 59
4.6 ORML Model - Constraints Section Example 60
4.7 ORML Model - Outputs Section Example 60
4.8 ORML Model - Call Expression Example 60
4.9 ORML Model - Boolean Expression Example 61

4.10 ORML Model - Comparison Expression Example 61
4.11 ORML Model - Arithmetic Expression Example 61

5.1 4+1 model (adapted from [85]) 68
5.2 ORML Interpreter - Namespaces Diagram 70
5.3 ASTNode Inheritance - Class Diagram 70
5.4 Example of ORML Statement Nodes - Class Diagram 71
5.5 Example of ORML Expression Nodes - Class Diagram 71
5.6 Example of a coefficient matrix 74
5.7 ORML Interpreter - Interface - Class Diagram 75
5.8 ORML Interpreter - Component Distribution 76
5.9 ORML Interpreter - Activities 77
5.10 ORML Model - Example 78
5.11 Figure 5.10 corresponding Arithmetic Abstract Syntax Tree 78
5.12 Dynamics AX Deployment Scenario 79
5.13 Autocomplete Situation Example 80
5.14 ORML Interpreter - Export Activities 81
5.15 Common Data Layer - Class Diagram 82
5.16 Model Management Layer - Class Diagram 82
5.17 Model and Model Instances Entity-Relationship Diagram 83
5.18 ORML Interpreter - Semantic Analyzer - Check Example 88

6.1 Testing Categories 100

A.1 Question 1 answers 121
A.2 Question 2 answers 122
A.3 Question 3 answers 122
A.4 Question 4 answers 123
A.5 Question 5 answers 123
A.6 Question 5b answers 124
A.7 Question 5c answers 124
A.8 Question 5d answers 125
A.9 Question 6 answers 125
A.10 Question 7 answers 126

C.1 Project Schedule 131

F.1 ORML Model - Traveling Salesman 143
F.2 ORML Model - Warehouse Picking Routes 144

H.1 ORML Model - Zebra 147
H.2 ORML Model - PetroChem 149
H.3 ORML Model - WycoDoors 150
H.4 ORML Model - Traveling Salesman 151
H.5 ORML Model - Warehouse Picking Routes 153

List of Tables

2.1 Project Priorities 17

3.1 Language Data Binding Comparison 37

4.1 ORML Data Types 52
4.2 ORML Operators 53

5.1 Data Type Mappings 75

6.1 Execution Times Tests 102

D.1 Microsoft Math Engine Supported Formats 134

LIST OF TABLES

Glossary

A Mathematical Programming Language (AMPL) High-level programming language for describing and solving high complexity problems for large scale mathematical computation. 35

Abstract Syntax Tree Tree representation of the syntax of some source code. 17

ActiveX Component object model (COM) developed by Microsoft. 24

ADO.NET Set of computer software components that can be used by programmers to access data and data services. ADO.NET is sometimes considered an evolution of ActiveX Data Objects (ADO) technology but was changed so extensively that it can be conceived of as an entirely new product. 15, 24, 74, 81

Advanced Interactive Mathematical Modeling Software Advanced development environment for building optimization based decision support applications and advanced planning systems. 33

Application Programming Interface Set of declarations of the functions (or procedures) that an operating system library or service provides to support requests made by computer programs. 43

Enterprise Resource Planning software systems that are used for operational planning and administration and for optimizing internal business processes. 2, 3, 5

Framework Re-usable design for a software system that may include support programs code libraries or other software to help develop and glue together the different components of a software project. 7

Graphical User Interface Type of user interface which allows people to interact with electronic devices like computers or hand-held devices. 35

Just-in-Time Also known as dynamic translation JIT is a technique for improving the runtime performance of a computer program. It converts code at runtime prior to executing it natively for example bytecode into native machine code. 68

Linear Programming Problem involving the optimization of a linear objective function subject to linear equality and inequality constraints. 25

Microsoft Development Center Copenhagen Microsoft's largest development center in Europe and outside of the USA that is dedicated to the development of Business Solutions. 3

Open Database Connectivity Standard software API for using database management systems. 34

Open Office XML The Office Open XML format was originally developed by Microsoft as a successor to its binary Microsoft Office file formats. The specification was later handed over to Ecma International to be developed as the Ecma 376 standard. It was published in December 2006 and can be freely downloaded from Ecma international. 42

Unix Operating System Computer operating system originally developed in 1969 by a group of AT&T employees at Bell Labs. 48

Yet Another Compiler Compiler Parser generator developed by Stephen C. Johnson at AT&T for the Unix operating system. 48

Acronyms

AIMMS - Advanced Interactive Mathematical Modeling Software. 33–35, 49

AMPL - A Mathematical Programming Language. 35, 36, 49

ANTLR - ANother Tool for Language Recognition. 47

AOS - Application Object Server. 23

API - Application Programming Interface. 27, 28, 43, 73, 74

ASCII - American Standard Code for Information Interchange. 34

AST - Abstract Syntax Tree. 17, 48, 69–73, 77, 78, 80, 84, 85, 87, 88, 90, 96, 98, 119, 120

BCL - Base Class Library. 44

BIP - Binary Integer Programming. 26

BNF - Backus-Naur Form. 53

CDL - Common Data Layer. 92, 96

CLI - Common Language Infrastructure. 44

CLR - Common Language Runtime. 24, 44

COM - Component Object Model. 24

CPU - Central Processing Unit. 1

CRUD - Create/Read/Update/Delete. 16, 74, 99, 100

CSP - Constraint Satisfaction Programming. 4, 26, 102

CTS - Common Type System. 44

DBMS - Database Management System. 23, 74

DEV - Developer. 8

DLL - Dynamic Link Library. 76, 81

DSL - Domain-specific language. 8, 45, 48

ERP - Enterprise Resource Planning. 3, 5–7, 9, 11, 12, 21, 24, 25, 109

FP - Functional Programming. 45

GNU - GNU's Not Unix. 47

GPLEX - Gardens Point Scanner Generator. 49

GPPG - Gardens Point Parser Generator. 49

GUI - Graphical User Interface. 35

IDE - Integrated Development Environment. 13, 14, 18, 29, 49, 67, 74, 91, 109, 121

IP - Integer Programming. 26, 51

IR - Intermediate Representation. 48

JIT - Just-In-Time. 68

JSSP - Job-shop scheduling problem. 39

LALR - Lookahead left-right. 48, 49

LINQ - Language Integrated Query. 24, 74

LISP - List Processing Language. 46

LP - Linear Programming. 4, 25, 26, 59, 73, 102

MathML - Mathematical Markup Language. 14, 17, 18, 41–43, 50, 76, 80, 91, 96, 99, 103, 108, 121

MDCC - Microsoft Development Center Copenhagen. 3, 7

MIP - Mixed Integer Programming. 26, 59, 102

MIT - Massachusetts Institute of Technology. 46

MPL - Mathematical Programming Language. 30–33, 49

MPLEX - Managed Package LEX. 49, 72, 83, 86, 87, 96

MPPG - Managed Package Parser Generator. 49, 72, 83, 87, 96

MSFT - Microsoft. 1

MSN - Microsoft Network. 2

ODBC - Open Database Connectivity. 34–36

OML - Optimization Modeling Language. 28

OMML - Office Math Markup Language. 14, 17, 18, 41–43, 50, 76, 80, 91, 96, 99, 100, 103, 108, 121

OOXML - Office Open XML. 42, 43, 80, 91

OPL - Optimization Programming Language. 29, 49

ORML - Operations Research Modeling Language. 6, 16, 18, 45, 49, 51–64, 68, 70, 73, 77, 80, 82, 85–87, 103, 108–110

PC - Personal Computer. 1

PM - Program Manager. 8, 113

POSIX - Portable Operating System Interface. 47

SDK - Software Development Kit. 42, 43, 48, 49

SFS - Solver Foundation Services. 27, 73

SQL - Structured Query Language. 32

TDD - Test-Driven Development. 17, 93, 94, 98, 101

TSP - Traveling Salesman Problem. 9, 38, 62, 64

VCS - Version Control System. 21

VES - Virtual Execution System. 44

VS - Visual Studio. 49

VSS - Visual Source Safe. 95

WPF - Windows Presentation Foundation. 18

WWW - World Wide Web. 41

XML - Extensible Markup Language. 34, 41

XSLT - Extensible Stylesheet Language Transformations. 43, 80, 121

YACC - Yet Another Compiler Compiler. 45, 47, 48, 87

Chapter 1

Introduction

No great discovery was ever made without a bold guess

Isaac Newton[1]

This chapter briefly contextualizes this internship project, presenting the company where it was developed, the development center and the product division where it is carried out.

1.1 About Microsoft

1.1.1 The Corporation

Microsoft Corporation [2] (NASDAQ: MSFT) is an American Multinational Computer Technology Corporation headquartered on Redmond, Washington USA, that focuses on the development of Software. The company was founded by the young entrepreneurs 'Bill' (William Henry) Gates III and Paul Gardner Allen, on the 4th of April 1975 in Albuquerque, New Mexico, to develop and sell BASIC [3] interpreters for the Altair 8800 [4], a microcomputer based on the Intel Corporation [5] 8080A Central Processing Unit (CPU) produced by Micro Instrumentation and Telemetry Systems [6].
A couple of years later, in August 1980, ten years before Linux was even an idea in Linus Torvalds head [7] [8], Microsoft announced its first Operating System, the Xenix [9], as a port of Unix for various 16-bit microprocessors. Later, in the mid-1980s, Microsoft rose to dominate the home computer operating system market with the Microsoft Disk Operating System - MS-DOS [10].
In 1985, Microsoft started to explore the emerging graphical capabilities of the PC and released Windows [11] version 1.0, but due to its lack of functionality it achieved very little popularity among the users. This was not Microsoft's final word on the subject and its following versions (3.1, NT, 95/98, ME, 2000, XP and Vista) have had rising acceptance and lead Microsoft to its leading position in the desktop computer market.

As of June 30, 2007, Microsoft had about 79,000 employees and a declared 2007 Net Revenue of about 51.12 Billion US Dollars, making it one of the largest software companies in the world [12] [13].
Having Windows and Office as the company's flagship products, it is highly successful in most of the industry areas. As of September 20, 2005, Microsoft is structured in three major divisions, each one with its one president:

- Platform Products and Services - Windows Client, Server & Tools and MSN;
- Business - Information Worker and Microsoft Business Solutions;
- Entertainment & Devices - Mobile & Embedded Devices and Home & Entertainment.

Microsoft's current mission statement is to "enable people and businesses throughout the world to realize their full potential".

1.1.2 Microsoft Dynamics

Microsoft Dynamics [14] is the line of business software made by Microsoft and it replaces the previous family of Microsoft Business Solutions. It currently comprises the following software:

- Customer relationship management
  - Microsoft Dynamics CRM
- Enterprise Resource Planning
  - Microsoft Dynamics AX (formerly Axapta)
  - Microsoft Dynamics GP (formerly Great Plains Software)
  - Microsoft Dynamics NAV (formerly Navision)
  - Microsoft Dynamics SL (formerly Solomon IV)
- Retail management
  - Microsoft Retail Management System (formerly QuickSell)

Microsoft business applications compete with the similar products from Oracle and SAP but tend to appeal to small and medium-sized businesses.
To extend their functionality, as it is usual with other Microsoft products, third party or other Microsoft software such as Windows Server, SQL Server and Exchange Server can be used together. By making use of its *stack*[1] [15] of class leading products that embrace

[1] Stack is mentioned in the solution's stack sense as a set of software subsystems or components needed to deliver a fully functional solution e.g. product or service.

the concept of doing more with less, Microsoft tends to offer real attractive solutions to its customers.

1.1.3 Microsoft Development Center Copenhagen

Microsoft Development Center Copenhagen (MDCC) is Microsoft's biggest development center in Europe and outside the United States [16] employing over 900 people from more than 40 countries. Established in 2002, following the acquisition of the Danish company Navision and mainly focused in [17] Enterprise Resource Planning (ERP) Solutions (Microsoft Dynamics AX and Microsoft Dynamics NAV), its goal is "becoming the world's leading software development center for business solutions".

1.2 Microsoft Dynamics AX

Microsoft Dynamics AX [18] is one of the ERP systems currently developed by Microsoft. Previously called Axapta, it was originally developed by the Danish Company Damgaard Data A/S with its beginnings in 1983. In 2000, the company was merged with Navision Software A/S (also from Denmark) and, together, they formed Navision-Damgaard. The company had 1200 employees and 30 offices, mostly in Europe, with 130000 customers and a reported $181 million in revenue in the earlier year.
In July, 11th, 2002 the company was acquired by Microsoft [19], and the product joined the previously acquired Great Plains (now Dynamics GP) and Solomon (now Dynamics SL) products, reinforcing the position of the Microsoft Business Solutions group in the ERP market for small and mid-market businesses, especially in Europe.
Microsoft Dynamics AX, in its 2009 version, is a "comprehensive business management solution for mid-sized and larger organizations that works like and with familiar Microsoft software to help your people improve productivity".
It was developed in 3 sites [18]: Fargo, USA (former Great Plains Software); Redmond, USA (Microsoft headquarters); and Vedbæk, Denmark (former NavisionDamgaard), where this project has been carried out.

1.2.1 Optimization Problems in Dynamics AX

Being an ERP system, Dynamics AX deals with huge chunks of data and should be able to make the most out of them giving its users as much good information as possible within time. It should also offer its users, again, within time and quality, answers to some of the most common problems. Examples are: production planning, service technician scheduling, warehouse picking routes or product modeling.
Currently, in Dynamics AX, most of these problems are solved using customized algorithms coded in regular imperative programming languages, like C# or X++, which are

hard to maintain and debug and usually have a low performance. This is a clear barrier to study, maintain, improve or replace models.
Ongoing investigation supports the use of mathematical models (CSP, LP, etc) to improve result accuracy and efficiency. In-code mathematical models do, however, loose almost all of the simplicity that is intended in them and since Dynamics AX is an extendable platform, where the user can, hopefully, easily add new functionality that is specific to his business this means that the used mathematical models should be easy to add or change.

1.3 Operations Research

Operations research[2] [20] is an interdisciplinary branch of applied mathematics which uses methods like mathematical modeling, statistics and algorithms to achieve optimal or good decisions in complex problems.
Some of the primary tools used by operations researches are statistics, optimization, stochastic, queuing theory, game theory, graph theory, decision analysis, simulation and, because of the computational nature of these fields it is also tied to computer science.
Although the modern field of operations research arose during World War II, according to some [21][22], Charles Babbage (1791-1871) is the "father of operations research" because of his research into the cost of transportation and sorting of mail which led to England's universal "Penny Post" in 1840.
During that time Babbage and his friend Colonel Colby were carrying out studies on the postal system [23]. Although Rowland Hill, later the creator of the penny post, probably have heard before about his studies, this was probably the first occasion on which he encountered an operational research approach to the postal system. It would also have been his first encounter with the concept of a uniform postal rate, an immediate corollary of Babbage's theory of the cost of verifying prices, a theory which he was later to discuss in his book "On the Economy of Machinery and Manufactures" [24].
Although the Penny Post was original from 1680 [25], when a merchant named William Dockwra organized the London Penny Post, which delivered mail anywhere in London for a penny, it was absorbed into the Post Office and, from then on, the charges gradually increased. These increases were particularly noticeable during, the almost continuous, wars with France because each time more money was needed, the cost of postage was increased. Because of these high costs, many frauds became common.
Rowland published a pamphlet entitled "post office reform" in which he proposed an uniform postage rate of 1d, - one penny - which would lead to an increase in correspondence and the virtual abolition of attempts to evade the postage.

[2] Also called Operational Research in the UK.

1.4 Motivation and Objectives

Studies support that Enterprise Resource Planning Systems implementations have always faced the "plain vanilla versus customer satisfaction" problem [26]. How to balance schedule and budget goals against the benefits of customizing an ERP software has been, since ever, puzzling project managers minds. Truth is, customization does tend to make users happier.
The flexibility provided, out off the box, by the newer generation ERP systems can preventive help to address this issue and, thus, reduce the stress in a company's ERP system implementation by simplifying the required customization.
In many, older-generation ERP system, most of the system's configuration values were "hard coded" making them hard to change and maintain each time a new release or upgrade was installed. New generation ERP systems have, however, become increasingly rules-based, meaning that the majority of these values, limits and other parameters are now entered and maintained in the system through the use of tables that can be accessed and maintained by the authorized, principal users of the system.
This project's modeling environment takes customization further by proposing something quite new in the industry by making use of mathematical models, with an easy to describe language, which allows some "algorithms", and not just the parameters, to also be easily configured, stored and maintained. It has, thus, been faced as a proof-of-concept to gather new ideas, study technologies and provide some feedback on user experience and requirements on the tools it uses.
These are its objectives:

1. Research the state of the art on methodologies and technologies regarding modeling and solving this type of problems;

2. Define a modeling language that can address, in an easy-to-use and simple way, these problems while providing data-binding capabilities with the Dynamics AX platform. Different approaches can be studied to carry this out: from extending existing technology to creating our own, more flexible, language. Integrate with Microsoft Dynamics AX database;

3. Identify 1-2 optimization problems in the Dynamics AX platform;

4. Design models, using the defined language, to solve the problems identified under 3;

5. Design and implement the *ORML Interpreter*;

6. Provide Integration with Microsoft Dynamics AX database, both in terms of data and meta data;

As secondary, but nonetheless also important, objectives it is also possible to refer:

1. Provide integration with the *ORML IDE* project;
2. Use state-of-the art and in-house ("house" being Microsoft) technologies when possible.

1.5 Report Overview

The rest of this report is organized as follows:

Chapter 2 - "Problem Description" starts by giving a deeper understanding of the problem and what this project aims to accomplish. It will then present the project's priorities, schedule and the deliverables.

Chapter 3 - "State of the Art" reviews the current concepts regarding relevant optimization problems, solvers and its usage in ERP systems. It also details the frameworks and tools used for the project's development and how they were chosen.

Chapter 4 - "Solution Specification" presents, on a high level, the proposed solution to the previously identified problem. It starts by presenting the Operations Research Modeling Language (ORML) language and goes on presenting approaches to some of the identified optimization problems in the Dynamics AX system.

Chapter 5 - "Implementation" gives the reader important knowledge about the way the system has been designed and implemented, stating the relevant decisions, as well as, the ways of interacting with it.

Chapter 6 - "Solution Evaluation" evaluates the solution, starting by presenting different techniques that were put in practice during the project to, preventively, ensure a good project outcome. It then describes the tests that were done to ensure that the developed product complies with its requirements with the expected quality levels. It finally draws some conclusions regarding some user opinions collected about the modeling experience.

Chapter 7 - "Conclusion and Future Work" reviews the project, draws the necessary conclusions, achievements and ends up by pointing out some ideas concerning future work that could be done.

Chapter 2

Problem Description

Things should be as simple as possible, but not simpler

Albert Einstein[27]

This chapter presents the requirements for the project and what it aims to achieve. It then presents its roadmap, including priorities and milestones.

2.1 Declarative Optimization Language for an ERP System

As new generation ERP systems evolve, new ways of developing, configuring and deploying them are constantly developed. Reducing the complexity of deploying and further customizing such systems is a goal to every major development company in this market.
The usage of new optimization technologies and of modeling languages to access them are some of these new approaches, for instance, in product configuration modules.
Microsoft currently has a team that is working in a new .NET-based system that provides model-driven planning, scheduling and optimization capabilities (Microsoft Solver Foundation) [28] which is designed to contribute to an integrated business and optimization platform. Besides this framework, there are some other, smaller, projects like a constraint solver developed in 2006 by two Portuguese trainee's who also did their internship in the MDCC [29] [30]. It would be interesting to study the use of some of these technologies to solve some of the problems that Microsoft's Dynamics AX ERP System currently tackles.
The Microsoft Solver Foundation Framework, in its current development state, doesn't, however, provide an easy to use way to model and, specially, bind the problem definition and data. This becomes increasingly important when it is considered that the Dynamics Ax product isn't built on the usual Microsoft *Stack* technologies which makes it even harder to integrate.
A solution to ease the modeling experience with tight integration, by providing data binding mechanisms, with the Dynamics AX system is, thus, an interesting challenge.
Instead of defining a solution, mathematical modeling languages, which can usually be

described as declarative Domain-specific languages (DSLs) describe a problem. This means that, in contrast with imperative programming languages, where serial orders are given to a computer stating the *how*, a declarative language focus on the *what* giving the programmer a higher abstraction level on the problem. This clearly makes the code shorter and easier to understand and maintain.

2.2 Target Personas

This project is targeting a wide variety of users. Being a proof-of-concept, it will also evaluate its acceptance by the users and give some feedback on further requirements and improvements. This project targets the following types of users:

1. Microsoft developers (Program Manager (PM)s and Developer (DEV)s) that currently have to put great efforts in translating operations research mathematical model to code and, further, in maintaining them afterwards;

2. Partners that do customizations for clients;

3. Customers that may want themselves to customize their solution or simply do little adjustments in the model.

Based on research and input from business customers, Microsoft has developed the Microsoft Dynamics Customer Model [14]. This model, acts as a tool to document, capture, visualize and share how people work within departments, and how that drives performance across organizations. The model visually represents people in different roles, or *personas*, which are based on real user data, and create a common language to guide Microsoft Dynamics product design.
After a quick survey (Appendix A), within the organization, some of the target personas, within this model, for this project, were identified:

- Emil - as a product designer, Emil has to be able to specify new products. The rules which specify if a product is valid or not, in a configuration environment, can be specified through the use of a constraint programming model. He also searches for the least expensive components. To do this, some criteria should be matched and a mathematical model which could describe the selection process may be developed.

- Ellen - as a warehouse manager, Ellen ensures that inventory levels are accurate and that periodic physical inventory counts occur. She also optimizes the warehouse and focuses on turnover rate reduction which means that she wants to know all sort of different information about the warehouse and use this information to optimize it. In the search of information some models may be useful.

- Sammy / Ted - They work in the logistics area and, mostly, try to optimize transportation and shipping tasks. To find the optimal routes to do this, some models could be used.

- Simon - As a partner system implementer/consultant, Simon analyzes customers' needs, writes the specification and puts together a customized solution. He may use models to easily customize and extend the ERP system.

2.3 Optimization Problems in Microsoft Dynamics AX

Optimization problems are common in ERP systems. The need for assisted decision mechanism creates the need to formulate models and develop algorithms that can address them in an automated way, supporting the user while doing it fast.
This section, presents two of these problems and adds a third one that was used in an exploratory way - the Traveling Salesman Problem (TSP) problem.

2.3.1 Traveling Salesman

The TSP problem is one of the most widely studied integer programming problems. It is a problem in discrete and combinatorial optimization and can easily be stated as [31]:

Problem 1. *A salesman is required to visit each of n cities, indexed by* $1,\ldots,n$. *He leaves from a "base city" indexed by 0, visits each of the n other cities exactly once, and returns to city 0, During his travels he must return to 0 exactly t times, including his final return (here t may be allowed to vary), and he must visit no more than p cities in one tour (By a tour it is meant a succession of visits to cities without stopping at city 0).*
It is required to find the itinerary which minimizes the total distance traveled by the salesman.

If t is fixed, then for the problem to have a solution we must have $tp >= n$. For $t = 1$, $p >= n$, we have the standard traveling salesman problem.
Although the TSP is conceptually simple, it is difficult to obtain an optimal solution since in a n-city situation, any permutation of n cities yields a possible solution, thus, creating, $m!$ possible tours that must be evaluated in the search space.
The study, within this project, was defined as being interesting because route optimization problems are quite common in ERP systems and because this is actually a well defined and studied model which can easily be modeled to trial the developed solution.
To do this, custom tables were created in a common AX system installation to support the data on which one could apply such model.

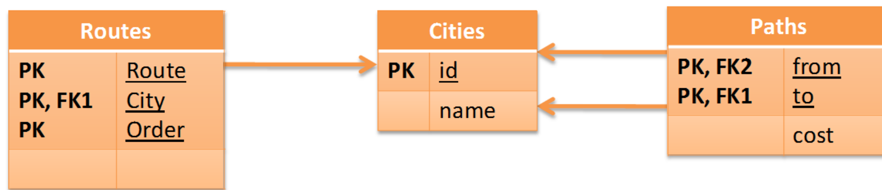

Figure 2.1: TSP Entity-Relationship Diagram

2.3.2 Warehouse Picking Routes

Picking, also known as order picking, consists in collecting products from a warehouse, to fulfill a client's request. The picking activity usually reduces the average request cycle time (time since the client places the order until it is fulfilled) although it may introduce an overhead to the warehouse personal costs. Some strategies may, however be used to optimize the picking operations, mainly, in three areas: product placement, operations and documentation/information [32]. The main overhead in these operations is the time that

Figure 2.2: Warehouse Environment

the operators spend in movements to pick the products and the crucial factors in defining a strategy to optimize these times are:

- Operators for request
- Products for request
- Request scheduling

With these factors in mind, one can present four, of the most known, strategies [33]:

Discrete picking - One operator does the complete collect. It is the process that is easier to operate and with the lower error rate associated. The overhead with movements between collects originates, however, is the biggest.

Picking by zone - The warehouse is divided in zones, according to product types and each zone has an assigned operator. To fulfill an order, the operator collects the products in his areas and places them in a common place. With this strategy, multiple operators can work for the same order.

Picking by lot - There is order accumulation. When an operator goes to collect the orders he can bring a greater quantity of that product thus improving its productivity.

Picking by wave - This process is similar to discrete picking but with order scheduling in turns. This means some products are only picked during certain times in a day. This introduces some advantages concerning reception and expedition of products. This strategy can be easily merged with other strategies like the picking for zone or lot.

Dynamics AX, like most ERPs, deals with order shipments. To process an order, one must pick the items from a warehouse that is arranged in aisles and racks. This procedure can be optimized using one of the previously presented strategies.
Currently, the product does not include an effective optimization algorithm for this operation and since the platform this project aims to develop can address these kinds of problems, it is an interesting problem to study.
In order to do so, the available relevant information in the current database was identified. After this, the additional required information, which would require the minimal changes to the database schema, was defined and a new table implemented according to figure 2.3. This involved the usage of a **discrete picking** strategy.

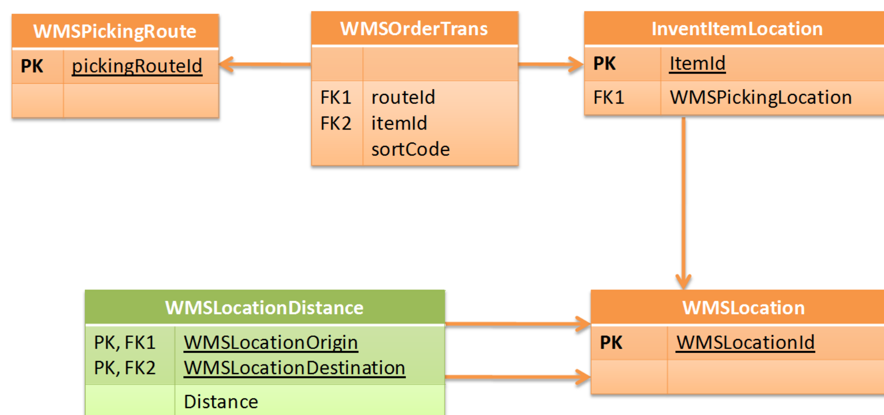

Figure 2.3: Warehouse Entity-Relationship Diagram

2.3.3 Production Scheduling

Scheduling is an important part of manufacturing and engineering by being able to have a major impact on a process's productivity. Its goal is to minimize the production time and costs by telling a production facility what to make, when, with which staff and on which equipment among many other possible variables. Production scheduling tools currently greatly outperform older manual scheduling methods. Two main types of scheduling can be identified:

Forward scheduling - planning the tasks from the date resources become available to determine the shipping date or the due date.

Backward scheduling - planning the tasks from the due date or required-by date to determine the start date and/or any changes in capacity require.

In the Dynamics AX system, the data that is needed to solve this kind of problems is currently being constructed in run time by a pre-processor which should be always used to access data. Because it is still in the initial phases of development all the details concerning the necessary data were nott possible to acquire by the time of this report. The work carried out for this problem will, however, be briefly presented as a way of introducing another problem type that is quite common in ERP systems.

2.4 Project Requirements

A clear definition of the requirements, is a necessary step in assuring a good project outcome [34] [35]. To do so, it is necessary to understand the market situation and the real needs of the product within its scope. Much work has been done in the study and definition of Mathematical Modeling languages. Some key points where identified as being of the most importance when creating such languages [36] [37].

- Syntax - The syntax should be complete, easy-to-learn and easy-read. Complete means that it should, ideally, cover all the features and structures that are supported by the state-of-the-art solvers (i.e. nonlinear functions, all sorts of variables, but also ordered sets).
- Solver Suite - For difficult combinatorial problems, different solvers may demonstrate very different behaviors. As such, it is important to allow the user to use his one solver of choice or even, to use its own implemented solvers.
- Infeasibility tracing - During the early modeling stages, one might see the solver returning infeasible solutions statements. Sometimes, it is really hard to identify the reasons for this and it can be of great help to support the analysis in this phase.

- Platform independence - A modeling system should not only support multiple platforms but should also be completely independent of the platforms and the operating systems.

- Open design - This is important not only to support the different types of solvers but also to support the great number of ways of storing data ranges that go from pure plain text files to huge supply chain management systems databases as Dynamics AX or SAP.

- Indexing - Real world problems, especially supply chain optimization models, easily require ten or more indices. The software needs to allow that many indices, but it also needs to handle the resulting data structures efficiently.

- Scalability - The language should support millions of constraints/variables. It's important that the modeling language has been designed in such a way to guarantee linear scale-up.

- Memory Management - The model generation phase can be very demanding in terms of memory management.

- Performance - The model generation phase should be very, very fast.

- Robustness - Most modeling languages are around for a long while. This means that if a new language is to come, it has to have an extra level of robustness or users won't adopt it.

Taking into account these criteria, a new language can be designed. Based on this language it was the goal of this project, to develop an interpreter which could execute on it. This interpreter should, among other things, provide the services to be used from third party applications like an Integrated Development Environment (IDE)[1] A good, and early, definition of the requirements is of the greatest importance and can directly influence the project outcome.
This section presents the functional requirements that were identified and that capture the intended behavior of the interpreter system. This behavior is expressed as services, tasks and functions that the system is required to perform. To do this, use case diagrams[2] supported by a features list (Appendix B) will be used.

[1]This project will be carried out in parallel with another project which aims to develop an IDE for the language of this project.
[2]Use cases have quickly become a widespread practice for capturing functional requirements by capturing the who (actor), the what (interaction) and the purpose (goal) without dealing with system internals

2.4.1 ORML Interpreter

The interpreter is the core component of all the system and it provides the tools to process and interpret the model source code and output the results. It also provides the services like syntax highlight or auto completion to build a rich edition environment. To use these functionalities, the interactions had to be studied and the connecting points identified.
To improve user's experience some other features, which are supported by the interpreter, have also been taken into account. These include model exporting capabilities to Mathematical Markup Language (MathML) and Microsoft's Office Math Markup Language (OMML) formats.

Use Cases

The use cases in figure 2.4 describe the features that were identified as necessary to be supported by the interpreter interface.

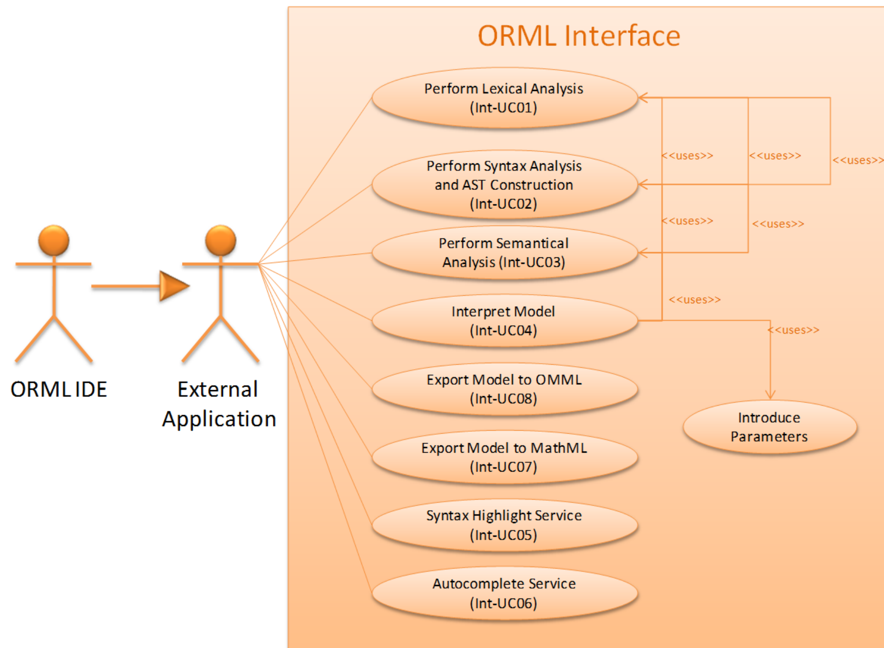

Figure 2.4: Interpreter Interface - Use Cases

Using this interface, it's defined as a goal to develop, also within the scope of this project, a simple console interface. This makes this project partially independent of the IDE project [38] while giving the user some additional usage options like being able to use it outside the Dynamics AX system as an external application.
The use cases for such console application are presented in figure 2.5.

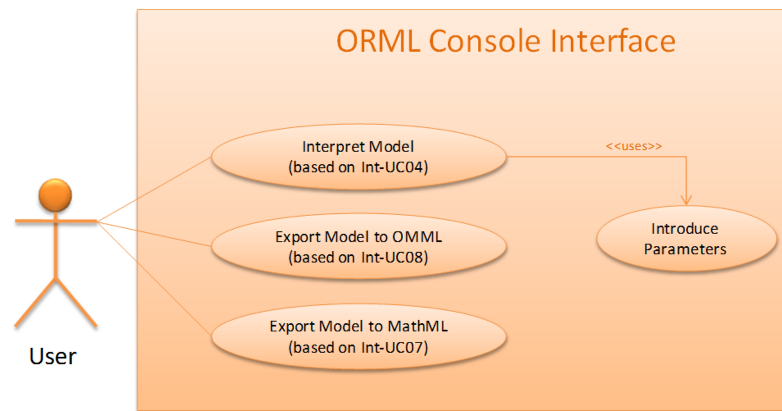

Figure 2.5: Console Interpreter - Use Cases

2.4.2 Common Libraries

Running along with this project, since the very first day, is, like previously presented the *ORML IDE* project. Given the amount of common necessities among the projects, some common libraries were defined. These libraries have to do, mainly, with the access to the Dynamics AX data.
Two common libraries have been defined: the Common Data Layer and the Model Management Layer.

Common Data Layer

The common data layer was defined as the library that both projects would use to operate on Dynamics AX data. By using it, it is possible, for instance, to commute between the AX Business Connector and a regular ADO.NET access to a database.
The usage scenarios for this library were divided in two types: data and meta data.
The data usage scenarios, focus on retrieving, inserting and updating the actual data in the system.
On the other hand, the meta data's mainly focus on retrieving information about the different data elements (i.e. tables, views and X++ Query-like classes). While tables and views can easily be understandable as data source elements, the X++ Query-like classes have been thought to support more advanced functionalities. These classes were thought to resemble what is commonly known as stored procedures since these, aren't in fact, available because of the underlying Dynamics AX design.

Model Management Layer

The Model Management Layer was thought as an abstraction layer over the models[3] making it easier to operate with them, by supporting, what is commonly referred as the CRUD operations. Besides models, it was defined that there should also exist model instances which represent a particular instance of a model with pre-defined parameter values. This allows the user to define a specific model that is ready to be used with some given parameters (section 3.4).

2.5 Schedule and Deliverables

The definition of the project's priorities, along with a schedule and a list of all the deliverables that should result from it, is an important step of every project. It gives the project's stakeholders a better vision and control over it, while establishing milestones and outlining what may or not be accomplishable. This section will start by establishing the project's priorities, followed by defining what are the deliverables to be expected given those priorities and ending up by presenting the schedule for such accomplishments.

Priorities

Prioritizing objectives is an important task in any project. This task can, however, prove to be harder to execute than it may seem to an inexperienced developer. Many different factors have to be taken into account and many of those may even prove to be very wrought estimates of what will be in practice thus providing a wrong decision criteria. Some of the factors taken in account for the project scheduling and prioritizing were:

- Importance to the prototype as a proof-of-concept project;
- Development difficulty;
- Dependencies to other projects;
- Dependency to the *ORML IDE* Project;
- Estimated task duration.

The priority levels are to be seen as relative to each other and serve only to guide the development schedule. The different use cases were categorized in the following way:

[3] A Model represents a problem definition in the ORML language. It represents the actual model text with its objective functions, constraints, parameters, bindings, etc

Use case	Priority
Perform Lexical Analysis (Int-UC01)	High
Perform Syntax Analysis and Abstract Syntax Tree (AST) construction (Int-UC02)	High
Perform Semantic Analysis (Int-UC03)	High
Interpret Model (Int-UC04)	High
Syntax Highlight Service (Int-UC05)	Medium
Auto Complete Service (Int-UC06)	Low
Export model to MathML (Int-UC07)	Medium
Export model to OMML (Int-UC08)	Medium

Table 2.1: Project Priorities

Schedule

This project time line has been divided in 4 different phases:

1. Spike[4]/Technology study
2. M1 Development
3. M2 Development
4. Documentation

These phases were decided upon the first phase taking into account the functionalities and priorities that were previously shown. A detailed Gantt chart for the project can be seen in Appendix C.

Spike/Technology Study

The Spike phase took about 1 month (15 February - 14 March) and consisted on the problem understanding, requirements elicitation and the major planning for the development phases, including the definition of priority levels and milestone dates. It was also during this phase that the main technologies were identified, briefly studied, and decided upon. This phase resulted in a Vision document, a Requirements documents containing a brief language description with examples and a preliminary Test Plan.

M1 Development

Starting where the previous phase ended, this phase also took about 1 month (17 March - 16 May) and it was during it that the language definition was completely developed and the first version of the interpreter developed using a Test-Driven Development (TDD) approach aiming at a vertical prototype to demonstrate the data-binding capabilities. The

[4] Spike is a focused exploratory approach to a problem.

prototype for this version wasn't yet fully compliant with the language definition nor did it supported all the services and features. It did, however, support the syntax highlight service to ensure a quick integration with the IDE. This phase produced the M1 code, an updated Requirements Document, an updated Test Plan and a simple design document. Some presentation materials were also produced during this phase for project promotion within the Company and to FEUP.

M2 Development

Milestone 2 development phase took roughly another month (19 May - 4 June) and it essentially took over the previously developed prototype and expanded it to, within reasonable and possible with respect to time constraints, comply with the language definition and the requirements on services and features.
On the services, the auto-complete service[5] was implemented and the export to MathML and export to OMML functionalities were also fully implemented.
This phase has mainly produced the M2 code. Again, some presentation materials were also produced and updated for the project promotion.

Documentation

The final month was totally reserved for the project documentation and the final report writing. Although much of its content has already been written along the project, there was still plenty to be done. This phase resulted in the final documentation and in this report.

2.6 Summary

This chapter began by presenting this project's goal of defining a declarative optimization language, called ORML, which would be used to address Optimization Research problems in the Microsoft Dynamics AX product. It was also defined, within its scope, the development of an interpreter that can process this language.
This interpreter should be accessible through a console application, also to be developed within the same project, and provide some services for the *ORML IDE* (an IDE based on the Windows Presentation Foundation (WPF) [39] technology that will be developed to support the language of this project). These two projects share two common libraries that have been jointly developed: the Common Data Layer and the Model Management Layer.
Finally, the four milestones for the project were presented: Spike / Technology Study,

[5] It was also result of this project the implementation of the graphical part of the auto-complete functionality in the IDE using the WPF Framework.

milestone 1 where the first prototype was developed, milestone 2 where the code for the project was developed and the documentation phase.

Chapter 3

State of the Art

Losers live in the past. Winners learn from the past and enjoy working in the present toward the future.

Denis Waitley[40]

This chapter starts off by building the required base of knowledge concerning the current Dynamics AX product, then presents the current state of the art on the field of relevant optimization problems, Optimization/Mathematical Languages and it ends with a technological review to prepare the reader for the following chapters.

3.1 Microsoft Dynamics AX

Dynamics AX is an ERP application that is part of the Microsoft Business Solutions Dynamics products family. It gives ERP application developers both a unique development and a run-time environment. This means that the system can be costumized or added extra functionality directly from the application itself.
The development is supported by the MorphX Integrated Development Environment that allows developers to graphically design data types, base enumerations, tables, queries, forms, menus and reports while supporting drag and drop features. It also allows access to any application classes that are available in the application, by launching the X++ code editor. Microsoft Dynamics AX also offers a fully integrated Version Control System (VCS) which supports collaborative development.
Like most ERP systems, it follows a client-server architecture where both a rich (native windows application) and a thin (Enterprise Portal[1]) clients are available which provide most of the same functionalities.

[1]The Enterprise Portal is an ASP.NET application which can be accessed, through the internet, using a common web browser.

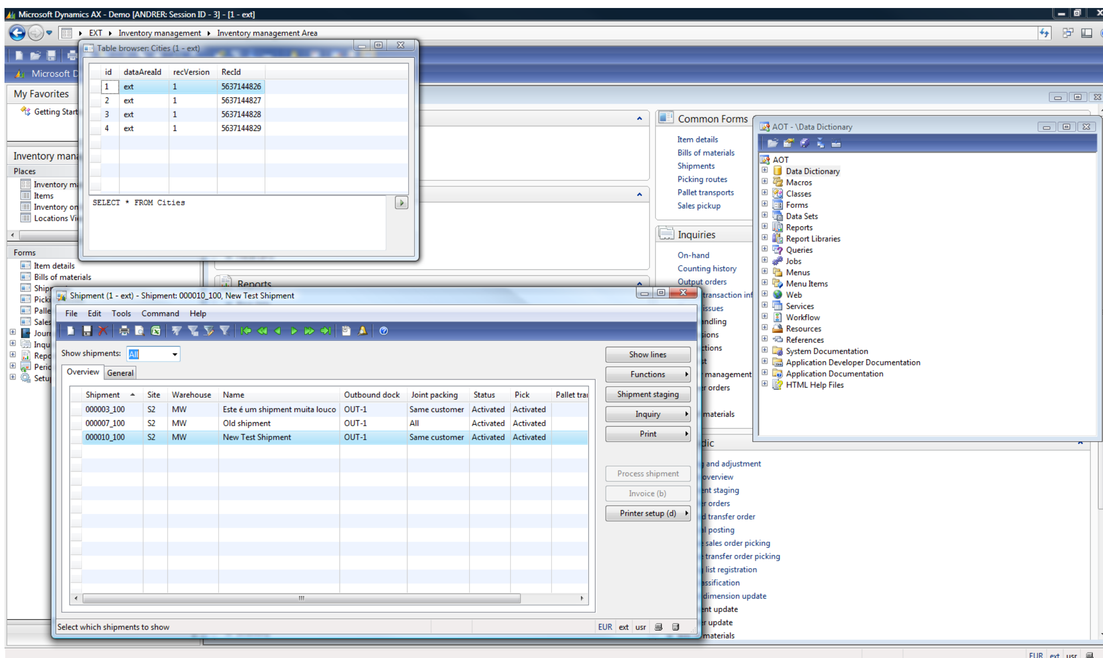

Figure 3.1: Dynamics AX Rich Client

3.1.1 Application Model Layering

Dynamics AX uses an application model layering [41] that allows very granular customizations and extensions to model element definitions. When a new Dynamics AX version, that is not country or region specific, is released, all elements reside in the lowest layer of this stack. Elements defined at higher levels can, however, override others defined at lower levels.

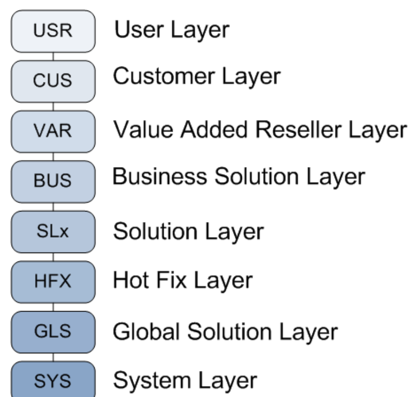

Figure 3.2: Dynamics AX Application Model Layers

Using this model, means that customizations may be preserved even if the application is re-installed/upgraded.

3.1.2 The X++ Programming Language

The Dynamics AX business logic is specified by code written in the X++ programming language which was specially designed for this product[2]. X++ is an object-oriented programming language with similarities to C++ and C#, supporting inheritance, encapsulation, polymorphism and it is case insensitive and strictly typed [42].
The language is application-aware because it includes keywords such as client, server, changecompany and display that are useful to specify if the code should be executed either on the client application or on the Application Object Server (AOS). It is also data-aware because it includes keywords such as firstFast, forceSelectOrder and forUpdate, as well as, a database query syntax that are useful for programming database applications.

```
1 public static void activateOrderTrans(str routeId)
2 {
3   WMSOrderTrans wmsOrderTrans;
4   ;
5
6   while select forupdate wmsOrderTrans where wmsOrderTrans.routeId == routeId {
7     wmsOrderTrans.expeditionStatus = "Activated";
8   }
9 }
```

Figure 3.3: Sample X++ Code

3.1.3 Data Sources

One of the Microsoft Dynamics AX key points regarding data sources is that it targets at abstracting the system from the specifics of the database system allowing it to be used with Microsoft SQL Server, Oracle or, eventually, other systems[3]. Because of this, its database doesn't have any information regarding relations, constraints and checks and the option was to implement a data access layer which provides most of the common entities available in a Database Management System (DBMS). These include:

- Tables
- Views

[2]The introduction of the X++ relates to the Damgaard company - the original Dynamics AX (Axapta) product developer.

[3]Dynamics AX 2009 currently supports Oracle Database 10g and Microsoft SQL Server 2000 and 2005.

- Queries

3.1.4 Integration

Integration of ERP systems with other systems within and beyond an organization is now a common requirement, thus, the Microsoft Dynamics AX product provides different ways of interacting with it.
Regarding customizations or new features additions, it can be done, like previously mentioned, from the application itself by using X++ and the MorphX environment. This approach is, however, limited in different aspects including, but not only, the limited knowledge on X++ and the necessity to use the limited built-in form designer. It is, however, possible[4] to import and use .NET libraries. It is also possible to load external ActiveX controls via the Component Object Model (COM) runtime.
Regarding data interaction, Dynamics AX currently offers three main ways of doing it:

- Natively in the Dynamics AX Rich Client, using X++ code and its extensive class library;

- In pure .NET, using the framework's libraries;

- Using the Microsoft Dynamics AX Business Connector.

Accessing data and business logic through the X++ facilities is the actual way the AX Rich client is implemented. Through it's usage it is possible to use classes, tables, views and much more. Interaction with external applications (e.g. .NET applications) it's, however, not always an easy task due to the type mapping limitations.
The second alternative, encompasses the usage of pure .NET code making use of the available libraries like ADO.NET which can be further extended using Language Integrated Query (LINQ). This method is, however, not recommended because it lacks all the information regarding the business logic (section 3.1.3).
Finally, by using the Business Connector, which mirrors most of the facilities actually provided by the native Dynamics AX data layer, like being database-agnostic, it is possible to access both the data and the business logic but also most of the facilities provided in the X++ libraries (e.g. it is possible to instantiate and use X++ classes). Currently, this platform is constituted by two components: the COM Business Connector and the .NET Business Connector.

[4] Although possible and relatively flexible, since X++ doesn't run in the Common Language Runtime (CLR), there are some type mapping limitations.

3.2 Optimization Problems

In mathematics and computer science, a *mathematical optimization problem*, or just *optimization problem*, has the form []:

$$\text{minimize/maximize} f_0(x) \tag{3.1}$$

$$\text{subject to:} f_i(x) \leq b_i, i = 1, \ldots, m. \tag{3.2}$$

Here, the vector $x = (x_1, \ldots, x_n)$ is the *optimization variable* of the problem, the function $f_0 R^n \to R$ is the *objective* function, the functions $f_i : R^n \to R, i = 1, \ldots, m$ are the (inequality) constraint functions, and the constants $b_1, \ldots, b_m$ are the limits, or bounds, for the constraints. A vector x^* is called *optimal*, or a *solution* of the problem 3.2, if it has the smallest objective value among all vectors that satisfy the constraints: for any z with $f_1(z) \leq b_1, \ldots, f_m$ we have $f_0(z) \geq f_0(x^*)$.

3.2.1 Linear Programming

Linear Programming (LP) dates back at least as far as Fourier [43], after whom the method of Fourier-Motzin elimination [44] is named. George B. Dantzig and John von Neumann were its main founders for publishing the Simplex method and the theory of the duality in that same year. Leonid Kantorovich, a Russian mathematician who used similar techniques in economics before Dantzig and won the Economics Nobel prize in 1975 should also be mentioned as one of them.
Linear programming studies the case in which the objective function g is linear and the set I is specified using only linear equalities and inequalities. If it is bounded, this set is called a polyhedron or a polytope.
This kind of problems can be expressed in the following canonical form:

$$Maximize/Minimize(C^T x) \tag{3.3}$$

Subject to:

$$Ax \leq b \tag{3.4}$$

Where x represents the vector of variables, c and b are vectors of coefficients and A is a coefficient matrix.
Arising as a mathematical model in the Second World War, Linear Programming was kept in secret until 1947, when, in the postwar, many industries found its use in their daily planning.
This clearly demonstrates that this kind of problems is worth studying in an ERP system implementation context. Many of the problem solutions in this kind of systems, as this report will briefly demonstrate, actually lie in this problem category or within one of its

subsets, like Integer Programming.

3.2.2 Integer Programming

Integer Programming (IP) can be formulated similarly to LP problems and occur when all the variables are required to be integers. Contrary to LP problems, which can be efficiently solved, IP problems are in many cases NP-Hard[5]. If the variables, rather than arbitrary integers, are required to be 0 or 1, the problem is defined as 0-1 integer programming or Binary Integer Programming (BIP). Finally, if only some of the variables are required to be integers, then the problem is called a Mixed Integer Programming (MIP) problem. These different variants are generally also considered to be NP-Hard.

3.2.3 Constraint Satisfaction Programming

Constraint satisfaction problems study the scenario where the objective function g is constant [45]. In this kind of problems, one must find states that satisfy a number of constraints. Constraint Satisfaction Programming (CSP) problems are subject of intense research in both artificial intelligence and operations research. Some of the most common algorithms used for solving constraint satisfaction problems include the AC-3 algorithm, Backtracking and the min-conflicts algorithm.

3.3 Optimization Engines

Optimization Engines are in the heart of the computer assisted Operation Research methodologies. Many products exist in the market to address one specific type of problem while others offer a variety of solvers. One of such products clearly has a dominant position in the market today - ILog CPLEX. For this reason, and since it lies within the objectives of this project to use a new Microsoft engine, this section will only address these two.

3.3.1 ILog CPLEX

CPLEX is an optimization software package developed by ILOG [46]. The key features in the product are its high performance, presence of fundamental algorithms, robustness, reliability and flexibility of interfaces. According to ILog, today, over 1000 corporations (including some of the world's leading software companies like SAP, Oracle and Sabre) and government agencies use CPLEX, along with researchers at over 1000 universities. This makes it one of the most, if not the most, used optimization engine in the market.

[5]NP-hard or nondeterministic polynomial-time hard problems are defined as.

The same company also distributes the ILog OPL which is a modeling language that provides a way to represent optimization models, requiring far less effort than using general-purpose programming languages.
Its usage is, however, not adequate for this project since the option, like previously stated, is to use, where possible, in-house technologies.

3.3.2 Microsoft Solver Foundation

Microsoft Solver Foundation is a NETfx/CLR-based platform for business planning, risk modeling, scheduling, configuration, and decision optimization. It delivers advanced technologies of mathematical programming and constraint processing to developers, modelers, and technical analysts to simplify and improve strategic and tactical decision making. MSF uses highly declarative models, consisting of simple components that are solved by the application of solvers, meta-heuristics, search techniques and combinatorial optimization mechanism to accelerate the solution finding process. The model components include:

- Decisions - The ”outputs” of the solver - correspond to the results of the model that is being solved;
- Constraint - Business constraints that have to be accounted for in solving a problem;
- Goals or Objectives - The business goal or goals to be achieved;
- Parameters - Data that plugs into the model so that a solution may be computed.

Based on its set of extensible solvers, that include, among others, support for linear and constraint programming, the platform defines several running modes (Figure 3.4) which can be used to both use it directly or to build applications based on it.

Figure 3.4: MSF High Level Architecture

In its current development stage, the platform provides a runtime Application Programming Interface (API), the Solver Foundation Services (SFS), which further augments the capabilities of the individual solvers with framework-level model analysis and meta-heuristics. Its usage, abstracts the users from the complexities of optimization allowing

them to focus on the modeling and solving. One of the key factors in this API is the Optimization Modeling Language (OML) which is made available, in a type safe version, to any .NET application, but also, to the Microsoft Excel Product in a syntax safe version.

3.4 Mathematical Modeling Languages

The word "modeling" comes from the Latin word *modellus* and describes a typical human way of representing the reality. Although abstract representation of real-world objects have been in use since the stone age, a fact backed up by caveman paintings, the first recognizable models were the numbers; as it is documented since about 30,000 BC.
A model can be simply put as:

A model is a simplified version of something that is real.

Mathematical modeling (or programming) consist in translating problems from an application area into tractable mathematical formulations whose theoretical and numerical analysis provides insight, answers, and guidance useful for the originating application.
Mathematical modeling languages were motivated by the desire to simplify the solving of mathematical programming problems. The fundamental underlying reason is the recognition that many mathematical programming problems can be expressed in a computer language whose syntax is close to the standard presentation of these problems in textbooks or scientific papers.
These languages abstract away the implementation details of the underlying solver technologies and allow users to focus on the modeling [36].
An important clear separation in these languages is the separation between the model and the instance data, which ensures that the same model can be applied to many instances without further additional work. For this, data binding mechanisms have became increasingly required and most solutions offer a variety of offers concerning this.
Although, traditionally, these modeling languages were particularly strong in linear and integer programming problems, new versions now include support to constraint and other types of problems.
Mathematical models have a close relationship with the mathematical modeling languages that are built upon them and which attempt to tackle the problem of building efficient optimization solutions in a faster way and requiring less effort than using general-purpose programming languages.
The translation process from a conceptual mathematical model to a modeling language is intended to be as easy and straightforward as possible. Since most of these languages have similar structures, this process has, however, become a somewhat standard process and can be defined in the following steps:

1. Describe the input and output data using sets and indexed identifiers;

2. Specify the optimization model;
3. Specify procedures for data pre- and post-processing;
4. Initialize the input data from files, spreadsheets and/or databases;
5. Solve the optimization model;
6. Display the results (or write them back to a database).

This section will present some of these languages. As a side note, it is worth noticing that all of them have support for the CPLEX optimization engine [47]. This clearly shows the importance of this software package in the market today.
One should also note that, during the analysis, a special focus was put in the data-binding mechanisms that each languages supports since this is one of the requirements for this project (Section 2.4).

3.4.1 OPL 6.0

The Optimization Programming Language (OPL) was originally developed by Pascal Van Hentenryck [6], and is a modeling language which combines the power of a Constraint Programming language and of a mathematical modeling language. It is the *defacto* language of the ILOG OPL Development Studio (currently in version 6.0).
Models developed in OPL can then be used from the ILOG OPL-CPLEX Analyst IDE or deployed into an external application written in Java, .NET or C++

Data-Binding

OPL models support data import from different internal and external data sources. This section focus on the external ones: Excel spreadsheets and database sources.

Excel Spreadsheets

Spreadsheet optimization allows users to create models that are easy to use, enabling the user to quickly update the data and solve the model. Microsoft Excel thus provides some excellent user-interface capabilities for optimization models. OPL builds on these capabilities by offering the ability to import and export data directly from Excel ranges. The model can then be solved with no limits on the size, speed or robustness of the solution.
Reading data form an Excel spreadsheet can be accomplished using the syntax of the example shown in figure 3.5. In the example of figure 3.5, the *Cutstock.xls* document is

[6] Pascal Van Henteryck was also the co-author of a previous, noticeable mathematical modeling language - Numerica

```
1  /* .mod file */
2  {string} CutsInPattern;
3
4  /* .dat file */
5  // connect to the spreadsheet
6  SheetConnection sheet("Cutstock.xls");
7
8  // read spreadsheet range
9  CutsInPattern from SheetRead(sheet,"patterns!A2:A4");
10
11 // write to spreadsheet range
12 CutsInPattern to SheetWrite(sheet,"patterns!B2:B4");
```

Figure 3.5: OPL - Spreadsheet Reading Example

opened and the specified cell range is first read and then written to another range using the `SheetRead` and `SheetWrite` commands.

Databases

Reading data from a database can be accomplished by using syntax of the example shown in figure 3.6.

```
1  /* .dat file */
2  // connect to database
3  DBConnection db("odbc","cutstock/user/passwd");
4  DBConnection db("access","cutstock.mdb");
5
6  // read values from database
7  CutsInPattern from DBRead(db,"SELECT * FROM patterns");
```

Figure 3.6: OPL - Database Reading Example

In the example of figure 3.6 the CutsInPattern values are read from the *cutstock.mdb* Microsoft Access database using the DBRead command.

3.4.2 MPL 4.2

Mathematical Programming Language (MPL) [48] is an ”advanced modeling system that allows the model developer to formulate complicated optimization models in a clear, concise and efficient way”. Models developed in MPL can be solved with any of the multiple commercial optimizers available on the market today.
Within the advantages that this product offers to the users, one should also highlight its feature-rich model development environment and its data import and export capabilities

from databases or spreadsheets. Some of the more notable and relevant features of the MPL language include [49]:

- Separation of the data from the model formulation;
- Import data from different data sources;
- Independence from specific solvers;
- Readable and helpful error messages.

An MPL model file is divided into two main parts: the definition and the model. The definition part is where the various items that are used throughout the model are defined - the model parameters. The model part then contains the actual model formulation. Each part is then further divided into sections, which are as follows: *The Definition Part:*

```
TITLE - The problem name.
INDEX - Dimensions of the problem.
DATA - Scalars, datavectors and files.
DECISION VARIABLES - Vector variables.
MACRO - Macros for repetitive parts.
```

The Model Part:

```
MODEL
MAX or MIN - The objective function.
SUBJECT TO - The constraints.
BOUNDS - Simple upper and lower bounds.
FREE - Free variables.
INTEGER - Integer variables.
BINARY - Binary (0/1) variables.
END
```

None of these sections is mandatory in a MPL model but, in order to have a valid optimization model, it should have, at least, an objective function and a constraint.
MPL allows sections to be placed in any order in the model, as long as, any item is declared before it is used in the model. Multiple entries of each section are also allowed.
Figure 3.7 shows a simple example model.

```
1 MODEL
2     MAX  x + y ;
3 SUBJECT TO
4     x + 2y  <  10 ;
5 END
```

Figure 3.7: MPL - Model Example

Data-Binding

MPL models support importing data from different data sources.

Text Files

MPL supports reading data from external text files which are mainly used when the data for the model is stored locally, generated by other programs or by running Structured Query Language (SQL) queries from a database. Reading data from a text file can be accomplished using the *DATAFILE* keyword followed by a filename inside parentheses. Figure 3.8 presents an example of such usage.

```
1 DATA
2     Demand[product,month] := 1000 DATAFILE("demand.dat");
```

Figure 3.8: MPL - Text File Reading Example

In this example, the demand vector has 36 values (12 months times 3 products) which are stored in the file demand.dat. Each number in the file is multiplied by 1000 as it is read. The file is a free-format text file where numbers are read in the order they appear, separated with commas or spaces. It is also required that there are enough numbers to satisfy the size of the vector.

Excel Spreadsheets

Reading data from an *Excel* spreadsheet can be accomplished using the *EXCELRANGE* keyword followed by parentheses containing the *Excel* wordbook filename and the Excel range name that the user wants to import from. An example of this, can be seen in Figure 3.9.

```
1 DATA
2     CutsInPattern[patterns ,cuts] := EXCELRANGE ("Cutstock.xls", "Patterns");
```

Figure 3.9: MPL - Spreadsheet Reading Example

In this example, MPL will open the Excel spreadsheet *Cutstock.xls*, locate the range patterns, and then retrieve the entries for the data vector CutsInPattern.

Databases

Elements for a data vector can also be directly imported from a database. Reading data from a database can thus be accomplished by using the *DATABASE* keyword followed by parentheses containing the table name and the column/filed name the user wants to import from. An example of this, can be seen in Figure 3.10.

```
1 DATA
2     FactDepCost[factory , depot] := DATABASE("FactDep", "TrCost");
```

Figure 3.10: MPL - Database Reading Example

In this example, MPL will open the database table FactDep, locate the columns *TR-Cost*, *FactID* and *DepotID* and then read in the entries for the datavector *FactDepCost*. It also allows filtering the instances one may want to read from a table. In that case, the *WHERE* keyword might be used followed by a condition on one of the columns. Figure 3.11 outlines an example of such situation.

```
1 DATA
2     FactDepCost[factory , depot] := DATABASE("FactDep", "TrCost" WHERE Region =
         "NorthWest");
```

Figure 3.11: MPL - Spreadsheet Reading Example with Filtering

In this example, only the transportation costs from the *FactDep* table where the *Region* column contains the entry *NorthWest* will be read.

3.4.3 AIMMS 3.8

Advanced Interactive Mathematical Modeling Software (AIMMS) is an advanced modeling tool for building decision support applications and advanced planning systems[50]. Along with its mathematical modeling language, AIMMS offers a graphical interface for

both developers and end-users, including a wide variety of solvers, and can be further extended to incorporate any of the advanced commercial solvers available on the market today.
With this, AIMMS provides a platform to create advanced prototypes that can be easily transformed into operational end-user systems.
An AIMMS model can be divided into three main parts:

- A declarative part which specifies all sets and multidimensional identifier defined over these sets, together with the fixed functional relationships defined over these identifiers;
- An algorithmic part consisting of one or more procedures which describes the sequence of statements that transform the input data of a model into the output data;
- A utility part consisting of additional identifier declarations and procedures to support a graphical end-user interface for the application.

Data-Binding

The initialization of sets, parameters and variables in an AIMMS application can be accomplished in several ways [51].

Text Files

AIMMS supports reading data from both ascii files in its own data format and from binary case files with a previous session.

XML Files

Extensible Markup Language (XML) Files read and write operations are also supported in AIMMS in two modes:

- Generate and read XML data based on identifier definitions in the model, or
- Generate and read XML data according to a given XML schema specification.

To access them, AIMMS provides the following functions:

- WriteXML(XMLFile,MappingFile[,merge]);
- ReadXML(XMLFile,MappingFile[,merge][,SchemaFile]).

Excel Spreadsheets

AIMMS supports reading data from an external Open Database Connectivity (ODBC)- or OLE DB-compliant database. Figure 3.12 presents an example of this.

```
1 ExcelRetrieveSet("Cutstock.xls", CutsInPattern, "Patterns");
```

Figure 3.12: AIMMS - Spreadsheet Reading Example

Databases

AIMMS supports reading data from an external ODBC- or OLE DB-compliant database. An example of the syntax required to accomplish this is given by figure 3.13.

```
1  SET:
2    identifier : Routes
3    subset of : (Cities, Cities) ;
4  DATABASE TABLE:
5    identifier : RouteData
6    data source : "Topological Data"
7    table name : "Route Definition"
8    mapping :
9      "from" --> i, ! name substitution
10     "to" --> j,
11     "dist" --> Distance(i,j),
12     "fcost" --> TransportCost(i,j,'fixed'), ! slicing
13     "vcost" --> TransportCost(i,j,'variable'),
14     ("from","to") --> Routes ; ! mapping to relation
```

Figure 3.13: AIMMS - Database Reading Example

3.4.4 AMPL

Developed at Bell Laboratories, A Mathematical Programming Language (AMPL) (AMPL) is a comprehensive and powerful algebraic modeling language for linear and nonlinear optimization problems with discrete or continuous variables[52]. As opposed to some others, AMPL doesn't have any native Graphical User Interface (GUI) environment, but there is an *AMPL Studio* [53] developed by OptiRisk Systems Company.

Data-Binding

The initialization of sets, parameters and variables in an AMPL application can be accomplished in several ways.

Text Files

In AMPL, the model and the data are stored in text files (.mod and .dat accordingly). Figure 3.14 presents an example of the syntax.

```
1 param max_prd := 123.7;
```

Figure 3.14: AMPL - Text File Reading Example

Excel Spreadsheets

The same mechanism that allows AMPL to interact with relational databases, the ODBC, also allows it to access common excel spreadsheets. An example of the syntax to accomplish this type of data binding may be seen in figure 3.15.

```
1 table Foods IN "ODBC" "diet.xls": FOOD <- [FOOD], cost, f_min, f_max;
```

Figure 3.15: AMPL - Spreadsheet Reading Example

Databases

The AMPL table declaration allows to import/export data and solution values from AMPL back to a relational database. This is done through standard handler support packages that can communicate via the ODBC standard under Windows. Figure 3.16 presents an example of such situation.

```
1 table Foods IN "ODBC" "diet.mdb": FOOD <- [FOOD], cost, f_min, f_max;
```

Figure 3.16: AMPL - Database Reading Example

3.4.5 Language Comparison

A comparison of some modeling languages is an interesting step. Table 3.1 presents a comparison between the different data-binding mechanisms offered by some of them:

Language	Text Files	XML Files	Excel Spreadsheets	Databases
OPL	No	No	Yes	Yes
MPL	Yes	No	Yes	Yes
AIMMS	Yes	Yes	Yes	Yes
AMPL	Yes	No	Yes	Yes

Table 3.1: Language Data Binding Comparison

It's interesting to notice that, like it was previously referred, most of them offer a wide variety of data-binding mechanisms being the most common ones the databases and the excel spreadsheets. Based on this, it's fair to conclude that these should be supported in any future modeling language, the moment it hits the market, if it wants to be fairly competitive.

3.5 Optimization Problems Formulation

The first step on solving optimization problems is usually to identify a mathematical model which can describe the model. To do this, the problem's parameters, variables, objectives and constraints have to be identified. The problems, for which this report presents some of these models, are, however, subject of much study from mathematicians, scientists and engineers. Therefore, the solutions which this report presents are just a part of the vast literature that exists and serves merely as an approach to the study of the framework which this project intended to fulfill.

3.5.1 Traveling Salesman

The traveling salesman problem, as it was presented, it is one of the most well studied problems in optimizations research. Different formulations have been presented and further extended to include a variety of transportation scheduling problems, such as the Multi-Traveling Salesman problem, the Delivery problem, the School Bus problem or the Dial-a-Bus problem.
The most compact mathematical formulation to the problem was, however, the formulation given by [31]:

Theorem 1. *Let $d_{ij}(i \neq j = 0, 1, \ldots, n)$ be the distance covered in traveling from city i to city j.*

Objective: Minimize the traveled distance

$$Minimize\Big(\sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} c_{ij} * \delta_{ij}\Big) \tag{3.5}$$

Where $c_i j$ represents the cost from city i to city j and $\sigma_i j$ is 0 if the path is not taken and 1 if it is.
Subject to:
Exactly one city must be visited immediately after city i.

$$\Big(\sum_{\substack{j=1\\ i\neq j}}^{n} \delta_{ij}\Big) = 1, i = 1, 2, \ldots, n \tag{3.6}$$

Exactly one city must be visited immediately before city j.

$$\Big(\sum_{\substack{j=1\\ i\neq j}}^{n} \delta_{ij}\Big) = 1, j = 1, 2, \ldots, n \tag{3.7}$$

Avoid sub tours.

$$u_i - u_j + nx_{ij} \leq n-1, i, j = 2, \ldots, n \wedge i \neq j \tag{3.8}$$

Where u_i is the sequence in which city i is visited $(i! = 1)$.
This is a mixed integer programming formulation with n^2 zero-one variables and n-1 continuous variables. One of the major drawbacks of this formulation is, however, the fact that it is usually limited to the traveling salesman problem only and cannot be easily extended to other transportation scheduling problems which are related to it [54].

3.5.2 Warehouse Picking Routes

The warehouse Picking Routes problem can be approached in a way which is very similar to the TSP problem. Given the different pickup points that the worker should visit to fulfill an order, the distance between them and the start and end points the problem almost completely resembles the TSP with the difference that the worker doesn't have to start and end at the same place.

Theorem 2. *Let $d_{ij}(i! = j = 0, 1, \ldots, n)$ be the distance covered in traveling from pickup point i to pickup point j.*

Objective: Minimize the traveled distance

$$Minimize\left(\sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} c_{ij} * \delta_{ij}\right) \tag{3.9}$$

Where $c_i j$ represents the cost from pickup point i to pickup point j and $\sigma_i j$ is 0 if the path is not taken and 1 if it is.
Subject to:
Exactly one pickup point must be visited immediately after pickup point i.

$$\left(\sum_{\substack{j=1\\ i\neq j}}^{n} \delta_{ij}\right) = 1, i = 1,2,\ldots,n \wedge i \neq StartLocation \tag{3.10}$$

Exactly one pickup point must be visited immediately before pickup point j.

$$\left(\sum_{\substack{j=1\\ i\neq j}}^{n} \delta_{ij}\right) = 1, j = 1,2,\ldots,n \wedge j \neq StartLocation \tag{3.11}$$

Where u_i is the sequence in which pickup point i is visited ($i! = 1$).
This formulation does, however, pose a couple of restrictions which were considered to be acceptable for the purpose of the study:

- A product has to be in only one specific location;
- The complete order has to be processed by only one worker.

It does, however, take into account that multiple items can be taken from one position since the model works with locations and not items.

3.5.3 Production Scheduling

The production scheduling problem can be approached as a common job-shop problem[55]. In the job-shop scheduling problem (JSSP), a finite set of jobs is processed on a finite set of machines. In the common approach, each job is characterized by a fixed order of operations, each of which is to be processed on a specific machine for a specific duration. Each machine can process at most one job at a time and once a job initiates processing on a given machine it must complete processing uninterrupted. The objective of the JSSP is to find a ”schedule” that minimizes the overall duration of the jobs. An mathematical model formulation for this type of problems can be stated as:

Objective: Minimize the traveled distance

$$Minimize\Big(\sum_{i=1}^{I}\sum_{j=1}^{M} c_{ij} * \delta_{ij}\Big) \tag{3.12}$$

Subject to:

$$ts_i \geq r_i, i = 1,2,\ldots,n \tag{3.13}$$

$$ts_i \leq d_i - \sum_{m=1}^{M} p_{im}\delta_{ij}, i = 1,2,\ldots,I \tag{3.14}$$

$$\sum_{j=1}^{n} \delta_{ij} = 1, i = 1,2,\ldots,n \tag{3.15}$$

$$\sum_{i=1}^{I} \delta_{ij}p_{ij} \leq max\{d_i\} - min\{r_i\}, m = 1,2,\ldots,M \tag{3.16}$$

$$y_{ii'} + y_{i'i} \geq \delta_{im} + \delta_{i'm} - 1, i, i' = 1,2,\ldots,I, i' > i, m = 1,2,\ldots,M \tag{3.17}$$

$$tS_{i'} \geq tS_i + \sum_{m=1}^{M} p_{im}\delta_{ij} - U(1 - y_{ii'}), i, i' = 1,2,\ldots,I, i' \neq i \tag{3.18}$$

$$y_{ii'} + y_{i'i} \leq 1, i, i' = 1,2,\ldots,I, i' > i \tag{3.19}$$

$$y_{ii'} + y_{i'i} + \delta_{ij} + \delta_{i'j'} \leq 2, i, i' = 1,2,\ldots,I, i' > i, m, m' = 1,2,\ldots,M, m \neq m' \tag{3.20}$$

$$ts_i \geq 0 \tag{3.21}$$

$$\delta_{ij} \in \{0,1\}, i = 1,2,\ldots,I, m = 1,2,\ldots,M \tag{3.22}$$

$$\begin{Bmatrix} y_{ii'} \in \{0,1 \\ , i, i' = 1,2,\ldots,I, i' \neq i \end{Bmatrix} \tag{3.23}$$

Since this problem, due to the reasons stated in 2.3.3, was never solved, its studied ended here.

3.6 Mathematical Document Formats

Mathematical document formats allow describing mathematical notations and formulas making it easy to interchange and represent them. MathML is one of the most popular formats but others do exist, like the OpenMath, the OMDoc and the OMML.
Some of these formats have been considered in order to support the exporting functionalities (Appendix B).

3.6.1 Mathematical Markup Language

The Mathematical Markup Language or, in short, MathML [56] is a XML format document for describing mathematical notations and capturing both its structure and content. MathML's original application is about encoding the structure of mathematical expressions so that they can be distributed, displayed and manipulated across the World Wide Web (WWW). With the increase support of the Mathematical Software vendors, MathML is rapidly becoming the defacto language for scientific publications on the Web.

3.6.1.1 Microsoft Math

Microsoft Math [57] is an educational program that allows users to solve math and science problems. Some of its top features include a graphics calculator with extensive graphics and equation-solving capabilities, a triangle solver that help to explore the relationship between the parts of triangles, a unit conversion tool, a step-by-step equation solver that provides step-by-step solutions to many math problems from basic math to calculus, a formulas and equation library with more than 100 common equations and formulas in a single location and, finally, Ink Handwriting support to work with Tablet and Ultra-mobile PCs.

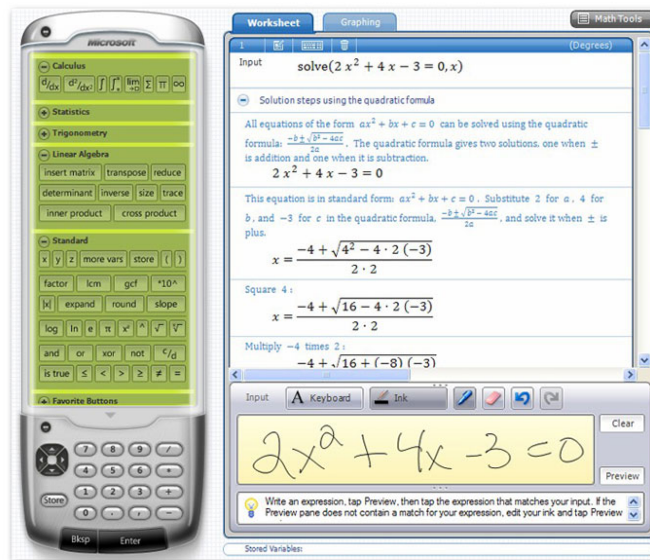

Figure 3.17: Microsoft Math

Its Software Development Kit (SDK) [58] offers a wide range of functionalities and futures to extend it or use it in external applications. An example might be its Math Engine which offers a wide variety of mathematical representation formats as well as, mechanism to convert between them (Appendix D) including the MathML format. Its usage, through the extension of this library was then leveraged as a basis for the export services that this project intended to achieve (Appendix B).

3.6.2 Office Math Markup Language

OMML is a mathematical markup language which can be embedded in an Open Office XML document [59], with intrinsic support for including word processing markup like revision markings, footnotes, comments, images and elaborate formatting and styles. The Office Open XML (OOXML) format uses a file package conforming to the Open Packaging convention that uses the ZIP file format [60] and contains the individual files that form the basis of the document. In addition to office markup, the package can also include embedded files such as images, videos or other documents.
Its primary markup languages are:

- WordprocessingML for word-processing (used in Office Word©2007);
- SpreadsheetML for spreadsheets (used in Office Excel©2007);
- PresentationML for presentations (used in Office Powerpoint©2007);
- DrawingML used for vector drawing, charts, and for example, text art (additionally, through deprecated, VML is supported for drawing).

The OMML format is partially compatible with the MathML format through relatively simple Extensible Stylesheet Language Transformations (XSLT) [61] based transformation files which are actually shipped with the Microsoft Office 2007 [62] product.

3.6.2.1 Open XML Formats SDK

Following the introduction of the Open XML formats, Microsoft developed an SDK to access and process files in this format as part of the WinFX technologies in the System.IO.packaging namespace[63].
The OOXML was built on top of this technology providing an easier-to-use Application Programming Interface (API) that provides strongly-typed part classes to manipulate Office 2007 documents [64]. By encapsulating many of the common tasks that are typically performed on OpenXML packages, complex operations can be performed with less lines of code. Some of these common tasks include [65]:

- Search;
- Document assembly;
- Validation;
- Data update;
- Privacy.

It is possible to use this technology in any language supported by the Microsoft .NET Framework which makes it especially suitable for this project.

3.7 .NET Framework

The Microsoft .NET Framework [66] was first publicly released in the year of 2002 as a way of competing with the unchallenged popularity of Java [67]. This language, created by James Gosling and other coworkers at Sun Microsystems in June 1991 [68], has several reasons for its success. Its popularity has been attributed to a series of characteristics that were introduced in it mainly from the fact that the code was run in a virtual machine.
By doing so, java outperformed other languages in a large variety of tasks, especially in the enterprise market, due to a series of features like platform-independence, object-oriented language, has garbage collection and huge variety of libraries.
Following this launch, Microsoft tried to present its own alternative, a modified version of Java, called J++. This product was, however, unsuccessful due to copyright issues. After this attempt, bearing in mind the critical success factors in Java and in some other programming languages, Microsoft developed its own solution: The .NET framework.

Its main design goals were: interoperability, having a common runtime engine, language independence, security, portability, simplified deployment and having a vast Base Class Library (BCL).

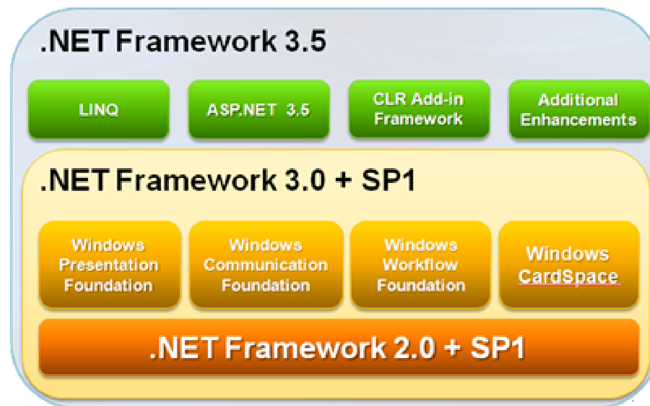

Figure 3.18: .NET Framework 3.5 Architecture

3.7.1 Programming Languages

Microsoft's .NET Framework [69] offers its users a variety of general-purpose programming languages which have in common the fact that they run over the Common Language Infrastructure (CLI). The purpose of the CLI is to provide a language-agnostic platform for application development and execution. Microsoft's implementation of the CLI is called the CLR. The CLR is composed of four primary parts:

- Common Type System (CTS);
- CTS;
- Metadata;
- Virtual Execution System (VES).

Visual C#

Microsoft Visual C# [70] is Microsoft's implementation of the C# programming language specification which Microsoft also created. It is an object-oriented programming language developed as part of the .NET framework. It has a procedural, object-oriented syntax based on C++ and includes influences from several other programming languages (most notably Delphi and Java). The term Visual denotes a brand-name relationship with other

Microsoft Programming Languages such as Visual Basic, Visual FoxPro, Visual J# and Visual C++.

Visual F#

Functional Programming (FP) is the oldest of the three major programming paradigms. Invented in 1995, about a year before FORTRAN, IPL was the first FP language. Visual F#[71][72], like visual C#, is Microsoft's implementation of the new F# programming language, which is a new programming language that is being developed in the Microsoft Research Labs at Cambridge. One of its great strengths is that it can use multiple programming paradigms and mix them to solve problems.
Supporting functional programming, F# is specially suited to language oriented programming because this paradigm generally has features that are well suited to creating parsers and compilers.
F# therefore allows the implementation of DSLs easily through two different approaches:

Metaprogramming with quotations: by using quote operators, the user can instruct the compiler to generate data structures representing the code rather than IL. This means that instead executing the code directly, the programmer will have a data structure that represents the code that was coded. This can be used to interpret or compile it into another language.

FSLex and FSYacc: This is the regular approach to the creation of stand-alone DSLs. F# currently supplies its own Lex and Yacc implementations supporting the programmer with tools that can ease the development of a language processor.

Implementing the ORML language/interpreter in F# would be a suitable decision. Using quotations, it is possible to develop an easy to use and straightforward grammar while keeping it inside a common programming language with all the advantages this can bring. For instance, if a user wants to add heuristic processing after an initial solution, that can be done. However, since most of the .NET code in Dynamics AX product is C# and since there was no upfront knowledge on F# (meaning that it would require some learning time) the option was not use it. C#'s usage was also a request from the project coordinator.

3.8 Language Processors

Programming languages are notations for describing computations to people and machinery. They differ from most other forms of human expression in that they require a greater degree of precision and completeness. Since computer do exactly what they are told to do, and cannot understand the code the programmer ”intended” to write, they require a greater degree of precision and completeness than natural languages.

Higher-level programming languages are generally divided for convenience into compiled languages and interpreted languages although there are also modern trends toward just-in-time compilation and byte code interpretation which introduce some gray areas into traditional categorizations.
Briefly stated, a compiler is a computer program (or set of programs) that translates text written in a computer language (the source language) into another computer language (the target language)[73]. Commonly, the output has a form suitable for processing by other programs (e.g. a linker), but it may be a human-readable text file. The first self-hosting[7] compiler that was capable of compiling its own source code in a high-level language was created by Hart and Levin at MIT in 1962 for the LISP programming language.
If a compiler had to process only correct source files, its design and implementation would be greatly simplified. However, programmers often write incorrect programs and a good compiler should assist them in identifying and locating errors. It is, therefore, an important compiler role, to report any errors in the source program detected during the translation process.
Errors may be of different levels and types. For example, errors can be:

- Lexical (e.g. misspelling an identifier, keyword or operator);
- Syntactic (e.g. arithmetic expression with unbalanced parentheses);
- Semantic (e.g. operator applied to an incompatible operand);
- Logical (e.g. infinitely recursive call).

An interpreter is another common kind of language processor. Instead of producing a target program as a translation, it appears to directly execute the operations specified in the source file, eventually, with user input.
The machine-language target program produced by a compiler is usually much faster than an interpreter at mapping inputs to outputs. An interpreter, however, can usually give better error diagnostics than a compiler, because it executes the source program statement by statement.

[7] Self-hosting refers to the use of a computer program as part of the tool chain or operating system that produces new versions of the same program.

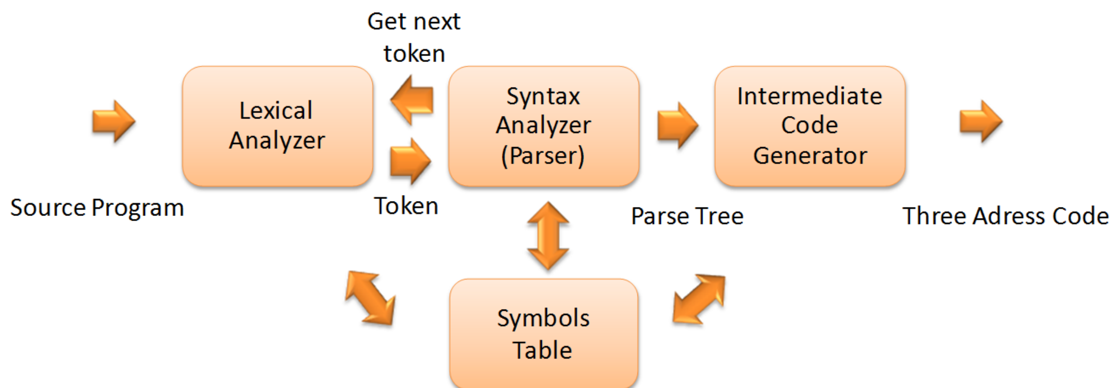

Figure 3.19: Compiler Front End Model

3.8.1 Compiler Compiler Tools

Compiler Compiler Tools have become increasingly popular in the 60's [74]. The first compiler-compiler tool was written by Tony Brooker in 1960 and was used to create compilers for the Atlas Computer at the University of Manchester.
Many others of such tools were to follow, most noticeable Yet Another Compiler Compiler (YACC), ANTLR [75], JavaCC or the GNU's Not Unix (GNU) bison.
These tools aren't able to read directly from a text input stream and require a series of tokens and, as such, are normally used with lexical analyzer generators like Lex or Flex which support the creating of lexical analyzers whose specification is usually based on regular expressions.

Lex/Yacc

Lex and Yacc are generators that can help the developer in creating lexical analyzers and parsers for his compiler.
Lex was originally written by Eric Schmidt and Mike Lesk and is the standard lexical analyzer generator on many Unix systems [76]. Its behavior is specified as part of the Portable Operating System Interface (POSIX) standard. Lex can read an input stream that specifies the lexical analyzer using regular expressions to describe patterns for tokens and outputs source code that implements the Lexical Analyzer in the C programming language. Internally, Lex transforms the input patterns into a transition diagram and generates code in a file called "lex.yy.c" that simulates this transition diagram. The outputted C program is a working lexical analyzer that can take a stream of input character and produce a stream of tokens. Its normal use is based on parser invocations of its functions that return an integer

that represents the code for one of the possible token names. The attribute value, whether it will be another numeric code, a pointer to the symbol table, or nothing, is placed in a global variable yylval, which is shared between the lexical analyzer and parser, thereby making it simple to return both the name and an attribute value of a token.

```
1  Definition section
2  %%
3  Rules section
4  %%
5  C code section
```

Figure 3.20: Lex File Structure

Yet Another Compiler Compiler (YACC) is a Lookahead left-right (LALR) parser generator developed by Stephen C. Johnson for the Unix Operating System [76]. Like its lexical brother (Lex), YACC also generates the code for the parser in the C programming language. It shares a similar file structure to Lex although it has a different, parser oriented, syntax.

```
1  Definition section
2  %%
3  Translation rules section
4  %%
5  C code section
```

Figure 3.21: Yacc File Structure

Microsoft Phoenix

Microsoft Phoenix is the code name for the software optimization and analysis framework that is the basis for all future Microsoft compiler technologies [77]. Because of this, before developing a language processor, its usage should be leveraged.
Phoenix is being developed at Microsoft Research and it's also available as an SDK. Using ASTs, flow graphs and an exception handling model. It also defines an Intermediate Representation (IR) and for any program to be handled, it needs to be converted to it.
Phoenix comes bundled with readers for Portable Executable binary files, CIL and the output of the Visual C++ front-end. Readers for other languages, have, however, to be written through third party tools such as Lex and Yacc. As such, it is not particularly suited for this project since an interpreter for a DSL doesn't rely on most of the usual mechanisms that are present in the back-end of a compiler for a general purpose programming language.

MPLEX/MPPG

Managed Package LEX (MPLEX) is a scanner generator which accepts a ”LEX-like” specification and produces a C# output file. MPLEX development is closely related to the Gardens Point Scanner Generator (GPLEX) application developed by John Gough and the Queensland University of Technology.
It is currently shipped with Microsoft Visual Studio 2008 SDK [78] and, offers some additional VS-specific interfaces when compared to GPLEX.
It is meant to be used with Managed Package Parser Generator (MPPG), which is also closely related to the Gardens Point Parser Generator (GPPG) generator for LALR(1) parsers. It accepts a ”YACC/BISON-like” input specification and produces a C# output file.
The parsers they both produce are thread-safe, with all parser state held within the parser instance.
Both of these tools use the generic types defined in C# 2.0 and, therefore, the .NET framework 2.0 or later is a requirement.
Among the advantages of using MPLEX and MPPG we can refer that since it is Microsoft proprietary software everything built with it can easily be put in production without any license issues. Another advantage is that it is very oriented to the visual studio IDE therefore allowing an easier future integration if it is seen as an advantage for the product. This also implies that, most probably, other Microsoft IDE’s that might exist now or in the future will have better compatibility with the language since its usual to maintain the ways of interaction.

3.9 Summary

This chapter started by presenting the Microsoft Dynamics AX product, introducing its high-level architecture, how the model layering system works and the X++ programming language with which most of the system is actually programmed. It also introduced the different ways of accessing its data giving a special focus to the .NET Business Connector.
Some of the most common optimization problem types were presented: linear, integer, and constraint programming. The solvers that support these kinds of problems were also presented: specifically CPLEX and the Microsoft Solver Foundation.
An analysis of existing modeling languages in the market followed. OPL, MPL, AMPL and AIMMS represent some of the most used mathematical/optimization modeling languages in the market today and serve as a reference point to the introduction of the ORML language.

A section about potential mathematical document formats followed, introducing some of the formats that may support the intended export functionality: the MathML and the OMML formats.
The chapter ended by presenting some additional technological stack that will be used during the project implementation like the .NET framework and a brief section about language processors, introducing some tools that may be used for the actual implementation.

Chapter 4

Solution Specification

I have been impressed with the urgency of doing. Knowing is not enough; we must apply. Being willing is not enough; we must do.

Leonardo da Vinci[79]

This chapter presents the solution defined to address the previously presented problem, making use of the knowledge that was gathered along the state of the art. It starts by presenting the language that was designed, defining its structure, elements and rules while giving demonstrative examples of its usage. It then presents how this language can address the example problems (Section 2.3) that this project was set to study.

4.1 Operations Research Modeling Language Specification

After studying and carefully analyzing the previously presented general requirements (section 2.4) and the different languages already in the market (section 3.4), a language definition was prepared. It wasn't a goal of this solution to address all of the identified requirements but only the ones which would allow this project to serve its purpose - a proof of concept of such system in Dynamics AX.

4.1.1 Language Overview

ORML is an optimization modeling language that aims to be solver and problem agnostic. Although it is mainly targeted at linear programming (IP included) and constraint satisfaction problems for the scope of this project, it has been designed in such a way that future extensions to other types of problems are possible. Being solver agnostic means:

- A model doesn't have to include an explicit mention to the type of model being solved - the interpreter will figure it out;

- The syntax is similar for the different types of models (e.g. a variables section, except for problem specific options, has most of the same syntax in both a constraint and a linear programming models);

The language is an interpreted and it has mechanisms to bind data to and from the Dynamics AX system.

Types

ORML supports two kinds of types: value types and reference types. Variables of value types directly contain their data whereas variables of reference type store references to their data. ORML's value types can further be classified as simple types and the reference types can be further classified as Vector types. Table 4.1 provides an overview of ORML's type system.

Category		Description
Value types	Simple Types	Signed integral: Integer IEEE floating point: Real
Reference types	Set types	Single and multi-dimensional, for example, [Integer] and [Integer,Integer]

Table 4.1: ORML Data Types

Expressions

Expressions are constructed from ***operands*** and ***operators***. The operators of an expression indicate which operations to apply to the operands. Example of operators include +, -, * and /. Examples of operands include literals, locations and expressions.
When expressions contain multiple operators, the precedence of the operators controls the order in which the individual operators are evaluated. For example, the expression x + y * z is evaluated as x + (y * z) because the * operator has higher ***precedence*** than the + operator.
The following table summarizes ORML's operators, listing the operator categories in order of precedence from highest to lowest. Operators in the same category have equal precedence.

Category	Expression	Description
Unary	-x	Negation
Multiplicative	x * y	Multiplication
	x / y	Division
	x % y	Remainder
Additive	x + y	Addition
	x - y	Subtraction
Relational	x < y	Less than
	x > y	Greater than
	x <= y	Less than or equal
	x >= y	Greater than or equal
Equality	x == y	Equal
	x != y	Not equal
Conditional AND	x && y	Evaluates y only if x is true
Conditional OR	x || y	Evaluates y only if x is false
Assignment	X = y	Assignment

Table 4.2: ORML Operators

Statements

The actions of a model are expressed using statements.
ORML supports several different kinds of statements:

- Labeled statements - used to distinguish between the different parts of the models;
- Declaration statements - used to declare variables and constants;
- Expression statements - used to evaluate expressions. Assignment expressions can be used as statements;
- Iteration statements - used to repeatedly execute an embedded statement. In this group we mainly highlight the where statement.

4.1.2 Grammars

Language grammars can be described by context-free grammars or Backus-Naur Form (BNF) notation [80]. Grammars offer significant advantages to both the language designers and compiler writers [81]:

- They give a precise definition of a language;
- From certain classes of grammars we can automatically construct an efficient parser that determines if a source program is syntactically well formed. As an additional benefit, the parser construction process can reveal syntactic ambiguities and other difficult-to-parse constructs that might otherwise go undetected in the initial design phase of a language and its compiler;

- Languages evolve over a period of time, acquiring new constructs and performing additional tasks. These new construct can be added to a language more easily when there is an existing implementation based on a grammatical description of the language.

The lexical grammar defines how characters are combined to form line terminators, white spaces, comments and tokens. The syntactic grammar defines how the tokens resulting from the lexical grammar are combined to form ORML models. Further definition, including the regular expressions and the grammar rules can be seen in Appendix E.

4.1.2.1 Lexical Grammar

An ORML model uses a lexical grammar based on four types of elements. This section will give a short description on them.

Line Terminators Line terminators divide the characters of an ORML source file into lines. ORML's line terminators comply with the Unicode standard.

White Spaces White spaces can appear anywhere in the language and are ignored. A white space is defined as a space character, a horizontal tab character, a vertical tab character or a form feed character.

Comments Two types of comments are supported: ***single-line comments*** and ***delimited comments***. Single-line comments start with the characters // and extend to the end of the source line. Delimited comments start with the characters /* and end with the characters */. Delimited comments may span multiple lines.

Tokens There are several kinds of tokens: identifiers, keywords, literals, operators and punctuators. Comments, white space and line terminator characters are not tokens, though they act as separators for tokens.

Identifiers Identifiers (also called symbols) represent entity names. The identifiers can be constituted by letters (a to z), digits and underscores (_). An identifier cannot contain only underscores and should contain at least one letter. Letters can be upper or lower case and the identifiers, like the keywords, are also case sensitive.

Keywords A keyword is an identifier-like sequence of characters that is reserved, and cannot be used as an identifier. Keywords are case-sensitive.

Literals A literal is a source code representation of a value.
Integer numbers are sequences of (only) digits that are expressed in decimal system. They cannot start with 0. Negative numbers are expressed with a starting dash (-).
Float numbers are also always expressed in the decimal system and should contain a dot (.). They can start with an integer number (0 included) and can have an optional fractional part after the dot. There should also exist at least an integer or a fractional part.

Operators and Punctuators There are several kinds of operators and punctuators. Operators are used in expressions to describe operations involving one or more operands. For example, the expression a + b uses the + operator to add the two operands a and b. Punctuators are for grouping and separating. Based on these types of elements, it is then possible to defined proper grammar rules that can build complete statements and expressions based on them.

4.1.2.2 Grammar Rules

An ORML model is conceptually divided into 6 different sections:

1. Indexes - Represent sets and are used to iterate through data sets. Can come from databases or from direct user input;
2. Inputs - Represent the model parameters. Can also come from a database or be directly specified by the user. In this section, the input data binding is done;
3. Variables - Represent the model variables. Among other things, can specify the domain, the type and their names and dimensions (arrays/maps);
4. Functions - The model objective. In linear models can be either maximize or minimize;
5. Constraints - Specify the rules that bound the model;
6. Outputs - Specify where the model's results should be stored.

Figure 4.1 outlines an example of a model written in ORML.

```
1  Model salesman {
2    Indexes:
3      // the indexes section
4    Inputs:
5      // the inputs section
6    Variables:
7      // the variables section
8    Minimize:
9      // the function section
10   Constraints:
11     // the constraints section
12   Outputs:
13     // the outputs section
```

Figure 4.1: ORML Model - Structure

Although order does matter in the interpretation (e.g. an identifier may not be used before declared), the sections in an ORML model may appear in any order within the model.
As a corollary to what this means, the output section will, however, be interpreted after all the other rules in the model are processed.

Indexes

Indexes are used as sets that specify iteration domains and is a relatively common section in optimization modeling languages of this nature (section 3.4.2). They can be used to specify domains which can be used to iterate over data. [82].
The user can specify four types of indexes:

1. Numeric;
   (a) Fully defined sequence;
   (b) Interval for integer indexes.
2. Named (Strings);
   (a) Fully defined sequence.
3. Database;
   (a) Which has to be a database primary key or unique column from a table or a view.
4. Query-like classes.

(a) Specially designed classes that are implemented to be query like and which can be integrated with ORML models. These classes will be further addressed in chapter 5.

Used in a model, an index section could be something like the example presented in figure 4.2.

```
1  Model {
2    Indexes:
3      Cities = Tables.Cities.name; // values from the name field in the cities
          table
4      Locations = Classes.WMSGetRouteItemLocations(RouteId, StartLocation,
          EndLocation).location; // values from the location field of the query
          like WMSGetRouteItemLocations class when invocated with the RouteId,
          StartLocation, EndLocation parameters.
5      Costs = [0 .. 4]; // a set with the numbers in the interval from 0 to 4
          inclusive
6      Costs2 = [0 , 4]; // a set with the numbers in the interval from 0 to 4
          inclusive
7      Sequences = {1, 2, 3, 4, 5}; // a set with the specified numbers
8      StringSequence = { "Leiria", "Porto", "Coimbra" };
9  }
```

Figure 4.2: ORML Model - Index Section Example

Inputs

The input section represents the data to be used by the model. It represents the data to be used, both explicitly or implicitly through the usage of the binding mechanisms.
The following types of inputs can be used:

1. Constant inputs or scalars - Mainly used to aid readability and make the model easier to maintain;

2. Sets - Are used when the coefficients come in lists or tables of numerical data. They can be specified as lists of numbers in the model. Their domain can be specified by using an index. To specify the indexes used, list the index names in brackets immediately after the set name and separate them with commas;

3. Database Sets - After the definition of the name of the set, the user can specify a database source, like in the indexes section, followed by the name of the columns the user wants to import;

4. Query-like classes - Query-like classes can be implemented in the AX Client which can be integrated with ORML models. These classes will be further addressed in the chapter 5.

An example of a input section is given in figure 4.3.

```
1  Model {
2    Inputs:
3      Num = 2; // constant declaration
4      Test = {12, 13, 14, 15, 16 }; // set declaration
5      Test[Cities] = {12, 13, 14, 15, 16}; // indexed set declaration
6      Cities = Tables.Cities.name; // database set declaration
7      Costs = Tables.Costs.cost[from, to]; // indexed database set declaration
8      Costs2 = Tables.Costs.cost[from=FromIndex, to]; // filtered indexed
           database set declaration
9  }
```

Figure 4.3: ORML Model - Input Section Example

Having a better look into the example, we can describe each of the declarations as follows:

Constant Declaration A constant declaration is defined as a direct attribution of a value (Line 3 in figure 4.3).

Set Declaration A set declaration is defined by a list of values. Line 4 in figure 4.3 outlines an example of such declaration.
The ORML language also supports indexed sets (i.e. Test[0] = 12, Test[4] = 16). An example can be seen in line 5 of the same figure.
If the specified index dimensions are different from the number of given elements in the Set, a semantic error should be given (e.g. Test[”Leiria”] = 12).

Database Set Declaration Database set declaration are similar to regular set declarations with the difference that, this time, the values come from a database. The syntax is similar to most of other languages in the market (section 3.4) with the specificities required to make it easier to model in the Dynamics AX product. An example of such situation can be seen in figure 4.3, line 6.
The indexing process is, however, a little different when compared to regular set declarations. Database set indexing is done by specifying the columns whose values should be used to index the set. An example can be seen in line 7 of the same figure (eg. Costs[”Leiria”,”Porto”] = 2).
It is also possible to apply filters to the elements in the set by writing matching conditions against the specified indexes. An example may be seen in line 8 of, again, the same figure.

Variables

The variables section is where the user may declare the decision variables he wants to use in the model definition.
They are the elements under control of the model developer and their values determine the solution of the model.
Like in the inputs, there are two types of decision variables: plain variables and vector variables (sometimes called subscripted variables). Each variable can also have one of two types: Integer or Real and a specific domain.
Figure 4.4 presents an example of a variables section within an ORML model.

```
1  Model {
2    Variables:
3      Integer in [0 .. 1] : PathFromCityToCity[Cities, Cities];
4      Integer in [1 .. NumCities]: SequenceCityVisited [Cities];
5      Integer in [0 .. 1]: AnotherVariable, OneMoreVariable;
6      Real in [0 .. 2]: YetAnotherVariable;
7  }
```

Figure 4.4: ORML Model - Variables Section Example

Still in this section, it is worth noticing how the grammar supports multiple variable declarations in one line (Lines 3 and 4).

Functions

The functions sections specify the objective in a model. It can either be a maximize or minimize objective and it may be used in LP or MIP models.
Figure 4.5 presents an example of a functions section in an ORML model.

```
1  Model {
2    Minimize:
3      Sum(i in Cities ; Sum(j in Cities, i != j ; Costs[i, j] *
           PathFromCityToCity[i, j]))
4  }
```

Figure 4.5: ORML Model - Functions Section Example

Constraints

The constraints section specifies to which limits will be the variables bound and how they relate to which other in other to solve to model.
An example of a constraints section can be seen in figure 4.6.

```
1 Constraints:
2   Sum(j in Cities | i != j ; PathFromCityToCity[i, j]) == 1 where(i in Cities
      );
3   Sum(i in Cities | i != j ; PathFromCityToCity[i, j]) == 1 where(j in Cities
      );
4   (SequenceCityVisited[i] - SequenceCityVisited[j] + NumCities *
      EdgeFromCityToCity[i, j]) <= (NumCities - 1) where(j in Cities | j != i
      );
```

Figure 4.6: ORML Model - Constraints Section Example

Outputs

The output section allows the user to specify where he wants to put the results of an execution in a database. Figure 4.7 presents an example of such section.

```
1 Outputs:
2   Tables.sequences.order[city] = SequencecityVisited;
```

Figure 4.7: ORML Model - Outputs Section Example

Expressions

Concerning the valid expressions in ORML, the following rules apply:

Calls A call expression represents the invocation of a pre-defined function. Although the ORML language, in its current version, doesn't support custom function definition, some system functions exist. Furthermore, this type of expressions could be easily extended in future language versions to support custom functions. An example of a call is given in figure 4.8.

```
1 Model {
2   Inputs:
3     Test = {12, 13, 13, 14, 15, 16, 17};
4   Variables:
5     Integer in [0 .. 20] : b;
6   Constraints:
7     b < max(Test); // call to the max function
8 }
```

Figure 4.8: ORML Model - Call Expression Example

Boolean Expressions A boolean expressions consists in a boolean operation between two child expressions. An example is shown in figure 4.9.

```
1  Constraints:
2    b > 2 || b < 3;
```

Figure 4.9: ORML Model - Boolean Expression Example

Comparison Expressions A comparison expressions consists in a comparison operation between two child expressions. An example is given in figure 4.10.

```
1  Constraints:
2    b >= 2;
```

Figure 4.10: ORML Model - Comparison Expression Example

Arithmetic and other expressions An arithmetic expressions consists in an arithmetic operation between two child expressions. An example of such expression is given in figure 4.11.

```
1  Sum(j in Cities | i != j ; PathFromCityToCity[i, j]) == 1 where(i in Cities);
2  Sum(i in Cities | i != j ; PathFromCityToCity[i, j]) == 1 where(j in Cities);
3  (SequenceCityVisited[i] - SequenceCityVisited[j] + NumCities *
       EdgeFromCityToCity[i, j]) <= (NumCities - 1)
```

Figure 4.11: ORML Model - Arithmetic Expression Example

4.2 Modeling Optimization Problems in Microsoft Dynamics AX

After the mathematical formulation of the problem and defining the ORML language syntax, it is now possible to formulate, in ORML, the models that were presented as the ones this report would address. It is however to mention that, given the reasons previously presented, it was not interesting enough to continue the study regarding the production scheduling since the pre-processor wasn't available within time for the purpose of this project and, therefore, such model will not be addressed here

4.2.1 Traveling Salesman

The ORML code that corresponds to the previously presented TSP mathematical model (see Appendix F for the complete model) starts by defining a model that has an Integer parameter:

```
1 Model(Integer NewPath) {
```

This parameter will be used to assign a code to the newly created path that will be outputed at the end of the run.
Next, a set cities is declared. This set is bound to the id field in the cities table and will be retrieved with no selection. This assumes that a path that travels through all the cities in the table is wanted.

```
2   Indexes:
3     Cities = Tables.Cities.id;
```

The next step will be to get the costs between the previously selected cities. To do this, an input will be declared and bounded to the cost field in the Paths table. This input will be then indexed by the from and the to column.
An input storing the number of cities is also declared.

```
4   Inputs:
5     Costs   = Tables.Paths.cost[from, to];
6     NumCities = Count(Cities);
```

At this point, every needed value to solve the problem is already in the model. Next, the decision variables need to be declared. Considering the model presented in 3.5.1, two variable sets are required: one that defines if a path is, or isn't, taken and, the other one, that is the actual result stating the order by which each city is visited.

```
7   Variables:
8     Integer in [0 .. 1] PathFromCityToCity [Cities, Cities];
9     Integer in [0 .. NumCities-1] SequenceCityVisited [Cities];
```

The objective function is then defined. It defines a minimize goal just like the mathematical model.

```
10  Minimize:
11    Sum(i in Cities ; Sum(j in Cities | i != j ; Costs[i, j] *
        PathFromCityToCity[i, j]));
```

Then, the three constraints that specify the visiting rules are specified.

```
12  Constraints:
13    Sum(j in Cities | i != j ; PathFromCityToCity[i, j]) == 1 where(i in Cities
        );
14
15    Sum(i in Cities | i != j ; PathFromCityToCity[i, j]) == 1 where(j in Cities
        );
```

```
16
17      (SequenceCityVisited[i] - SequenceCityVisited[j] + NumCities *
            PathFromCityToCity[i, j]) <= (NumCities - 1) where(j in Cities, i in
            Cities | j != i && i != 1 && j != 1);
```

Finally, it is specified that the output of this model is the set SequenceCityVisited and that this set will be stored in the ordernum field of the CityPathSequence and that the results will be indexed by the city (remember that the SequenceCityVisited is uni-dimensionally indexed) and by the path columns. The path column will have the fixed value of the NewPath parameter.

```
18    Outputs:
19      Tables.CityPathSequence.ordernum[city, path=NewPath] = SequenceCityVisited;
20  }
```

4.2.2 Warehouse Picking Routes

The ORML model that corresponds to the previously presented mathematical model starts by the definition of the model parameters. For this model, three parameters are required: the routeId that specified the route which the model should optimize, the StartLocation that specifies the picking start location and the EndLocation that specifies the droping location.

```
1  Model(String RouteId, String StartLocation, String EndLocation) {
```

Then, the costs input is read from the field distance of the query-like class WMSLocationDistance indexing the results by the values of the WMSLocationOrigin and WMSLocationDestination fields of the same class.

```
2      Inputs:
3          Costs = Tables.WMSLocationDistance.Distance[WMSLocationOrigin,
               WMSLocationDestination];
```

An index is then read from the WMSGetRouteItemLocation query-like class that receives the model parameters and returns the complete list of locations that should be visited.

```
4      Indexes:
5          Locations = Classes.WMSGetRouteItemLocations(RouteId, StartLocation,
               EndLocation).location;
```

Like in the previous model, the next step is the declaration of an input that will hold the number of locations.

```
6      Inputs:
7          LocationNum = Count(Locations);
```

Next, the variables are declared. Again the same type of variables as in the TSP model are used.

```
8  Variables:
9      Integer in [0 .. 1] PathFromLocationToLocation [Locations, Locations];
10     Integer in [1 .. LocationNum] SequenceLocationVisited [Locations];
```

The objective function is set.

```
11 Minimize:
12     Sum(i in Locations ;
13         Sum(j in Locations | i != j ; Costs[i, j] *
               PathFromLocationToLocation[i, j]));
```

And the constraints that state how the picking points should be visited, are then defined.

```
14 Constraints:
15     Sum(j in Locations | i != j && Exists(Costs[i,j]) ;
           PathFromLocationToLocation[i, j]) == 1
16         where(i in Locations | i != EndLocation);
17     Sum(i in Locations | i != j && i != EndLocation && Exists(Costs[i,j]) ;
            PathFromLocationToLocation[i, j]) == 1
18         where(j in Locations | j != StartLocation);
19     SequenceLocationVisited [i] - SequenceLocationVisited [j] + LocationNum
           * PathFromLocationToLocation[i, j] <= (LocationNum - 1)
20         where(i in Locations, j in Locations | j != i && i != StartLocation
               && j != StartLocation);
```

The final step, specifies that the outputs - SequenceLocationVisited - should be "placed" in the sortcode field of the WMSUpdateOrderTrans Query-like class, indexed by the location and routeId fields.

```
21 Outputs:
22     Classes.WMSUpdateOrderTrans.sortcode[location, routeId=RouteId] =
           SequenceLocationVisited;
23 }
```

4.3 Summary

This chapter began by introducing the goal for this project: a declarative modeling language that can address optimization problems within the AX system. The language specification was presented together with some usage examples. Simply stated, an ORML model can be divided in six different sections: Indexes - which are used as sets that specify iterations domains; Inputs, which represent the data to be used by the model; Variables, that represent the decisions in the model; Functions, that specify the objectives; Constraints, that specify the limits to which the variables will be bound and how they

relate to each other and Outputs that specify where to store the results of an execution.
After the introduction of the language, the problems previously introduced were addressed using it. Examples of complete models for both the Traveling Salesman and the Warehouse Picking Routes problems were given and explained.
These models will make part of the evaluations made in chapter 6 to the developed solution and serve as a basis to draw some conclusions in chapter 7.

Chapter 5

Design and Implementation

Do, or do not. There is no try.

Jedi Master Yoda[83]

This chapter presents an inside look into the system implementation process. Special emphasis will be given on the complex tasks, such as the compilation and interpretation mechanisms, as well as, data access, services and export functionalities.
It starts by presenting the *ORML Interpreter* design details. This defines how the system was built, based on which components and how they interact. This will set the stage for the following section regarding the implementation details where the specifics regarding, especially, the interpretation process are presented.
This description is complemented by the details on the services implementation. Finally, some details concerning the export features are also be given.
The chapter then highlights some points on the implementation of the Common Libraries that are used by both this and the IDE project. It ends by presenting some development methodologies and tools as a way of presenting the mechanisms and practices that have been used throughout the project and that can provide a valuable help to further evolve or document it.

5.1 Design

Software design/architecture deals with the design and implementation of the high-level structure of the software. It deals with abstraction, with decomposition and composition, with style and aesthetics and is commonly organized in views, which are analogous to the different types of blueprints made in building architecture.
The goal, according to the Rational Unified Process [84], is to show how the system will be realized in the implementation phase.
The Software architecture can be expressed in a model composed of multiple views or perspectives - the 4+1 model [85].

This software model enables to describe the architecture of software-intensive systems, based on the use of multiple, concurrent views [86]. This enables to handle separately the functional and non functional requirements and allows to address separately the concerns of the various people involved in the architecture: systems engineers, developers, project managers and end-users [85]. While the first view was already presented in section 2.4 through the use cases, this section presents the other four.

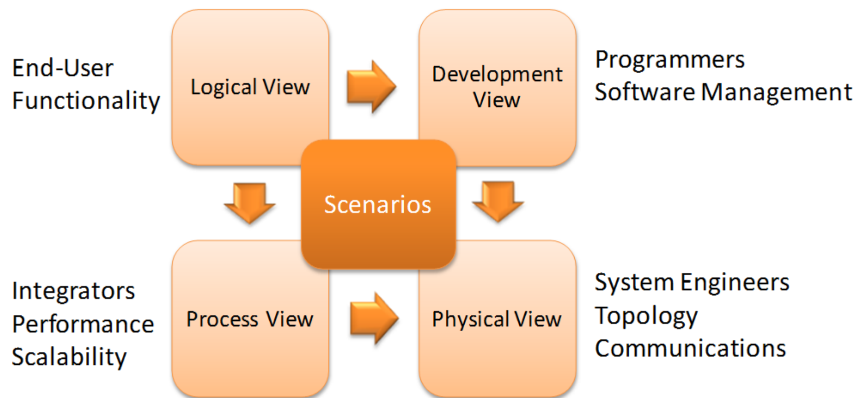

Figure 5.1: 4+1 model (adapted from [85])

5.1.1 ORML Interpreter

After the essential task of establishing a correct language definition, a language processor can be designed to accommodate it. It's important to mention that the language definition should be treated completed separately from the interpreter since others could also be implemented based on the ORML language definition. The *ORML Interpreter* represents the set of components that are related to the model's interpretation and that serve as the basis to the services implementation.

An Interpreter's front-end architecture is quite similar to the one of a compiler having its major differences in the back-end - whereas a compiler aims at translating representations like a general purpose programming language (e.g. C#) to machine code, an interpreter expects to process the instructions, eventually, outputting some result. Generally, an interpreter may be a program that either [87]:

- Executes the source code directly;
- Translates source code into some efficient intermediate representation (code) and immediately executes this. This is also called Just-in-Time (JIT) compilation;
- Explicitly executes stored precompiled code made by a compiler which is part of the interpreter system.

While the second and third options may give some performance advantages, they also required some added effort in development and may bring some security issues for having an intermediate data representation. Thus, it was decided that the source code for the models would be executed directly. As such, the interpreter should perform the following operations:

- Lexical analysis - This is the process of converting a sequence of characters into a sequence of tokens being a token a categorized block of text, also known as lexeme.
- Syntax Analysis - This is the process of analyzing the sequence of tokens retrieved from the lexical analysis and determining the grammatical structure with respect to a given format grammar.
- AST Construction - The AST is a tree representation of the syntax of some source code where each node denotes a construct occurring in the source code while being abstract because it may not represent some constructs that appear in the original source (parentheses is a common example of this).
- Semantic Analysis - In this phase, the semantic information is added and the symbols table is built. Semantic checks such as type checking are performed during this phase.
- Interpretation - In this last phase, the AST is finally interpreted and the operations executed.

Details on how these phases were implemented will be given in section 5.2.

5.1.1.1 Logical View

The logical view focuses on presenting the application's functionality in terms of structural elements, key abstractions and mechanisms, separation of concerns and distribution of responsibilities.
This section will presents each of the different components functionality by exposing their main classes and structures.
In order to give the interpreter the desired level of flexibility and, an overall, good design, a proper component distribution is required. Figure 5.2 represents the *namespaces* that were identified for the complete interpreter, including its interfaces and the console application.

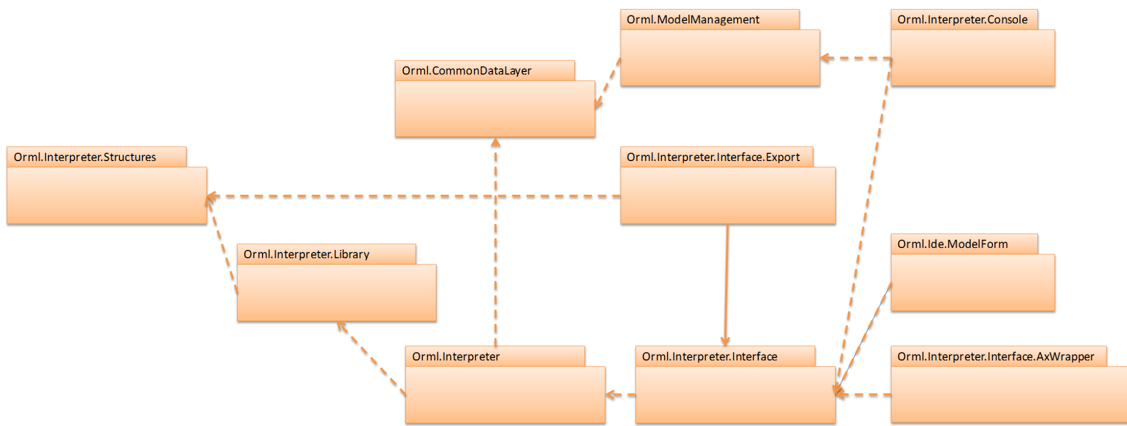

Figure 5.2: ORML Interpreter - Namespaces Diagram

ORML Interpreter Structures

The Orml.Interpreter.Structures namespace includes all the classes that represent the common structures that the interpreter uses. This includes the AST nodes, the symbols table classes, but also some common interfaces.
The AST nodes allow to represent an ORML model and are mainly defined by the expression and statement types that exist in the language definition. The tree's fundamental element is the AstNode class since all other nodes inherit from it. Figure 5.3 represents this structure. One should note, however, that the diagrams that follow are not exhaustive and don't include all the classes that may constitute an AST.

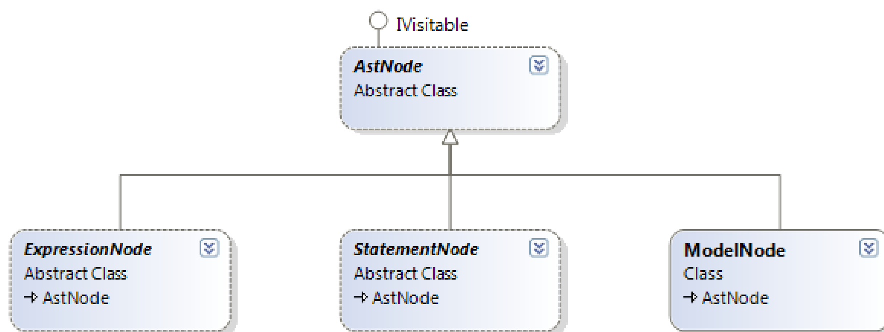

Figure 5.3: ASTNode Inheritance - Class Diagram

Further into the structure, the statement nodes are mainly divided in the section or named statements that define the different parts of the model and the attribution nodes.

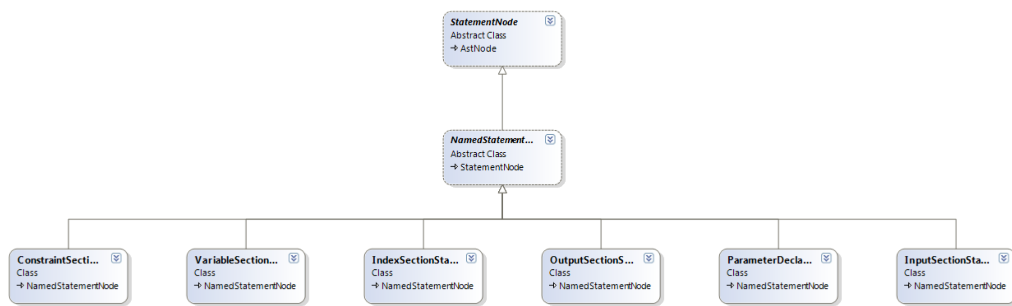

Figure 5.4: Example of ORML Statement Nodes - Class Diagram

Expression nodes, contrary to statement nodes, represent operations, locations, and all the other kind of compositions that may return something. This includes locations, functions, and the common arithmetic/boolean operations.

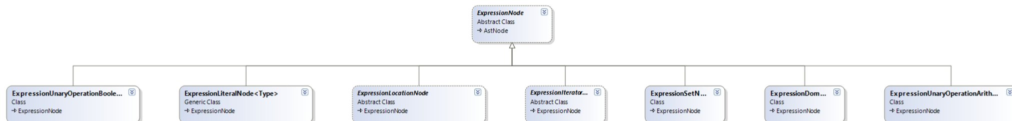

Figure 5.5: Example of ORML Expression Nodes - Class Diagram

Finally, still concerning the AST classes, it is important to mention that they were designed having in mind a visitor pattern [88] [89] [90] which abstracts this general structure from an entity that may need to access it. This is ideally suitable to compilers/interpreters because it allows to have different algorithms working on the exact same structure without adding any specific code to the structure itself[81].
Another important part of this namespace is the symbols table. This is the data structure that is used by the semantic analyzer and the language interpreter where each identifier in the model is associated with information related to its declaration in the source code such as its type and scope level. Its implementation is based on a hash table to optimize its most usual operations - insertions and lookups.
The symbols table has to deal with scopes and the option was to use a stack approach where, for each scope, a new table is created and added to a stack. When an entry is requested, the search begin in the top-most to the lower table on the stack.

ORML Interpreter Library

The Orml.Interpreter.Library namespace contains the interpreter's front end classes. This includes the lexical analyzer and syntax analyzer classes, with the required extensions to further support services in higher layers.
The lexical analyzer is the entry point for the interpretation engine. It is responsible for

reading the input source file and identifying its tokens and producing a sequence that the syntax analyzer may use.
Since it is the part of the interpreter/compiler that reads the source text, it also performs some secondary tasks like stripping out from the source program comments and white spaces. Additionally, it is also responsible for signaling eventual errors that may come across during this process. The errors are associated with positions to enable the users to easily correct them.
The Lexical Analyzer was created using the, previously presented (see section 3.8.1), MPLEX tool. This means that some classes were auto-generated from a rules definition file. Details concerning its implementation and extensions will be discussed in the next chapter.
A Lexical Analyzer, built with such tool, derives from the ScanBase class, which contains the methods that are common to all the lexical analyzers that may be built with it, and implements the language specific functionality in the Scanner class, using the Scanner::Table structure, which supports input from three different buffer types - Text, Stream and String. The decision to use MPLEX was based on two things:

- Use Microsoft Technology
- This project aims to create a prototype that can easily evolve (keep in mind that this project works like a proof of concept).

Since MPLEX ensures both these points while supporting the programming language of choice (C#), it was the perfect match.
Controlling the lexical analysis process, it is the syntax analyzer. This component, requests the tokens from the lexical analyzer and verifies if they can be matched against a rule in the languages's grammar.
The syntax analyzer implementation is supported by the, MPLEX counterpart, MPPG tool. Its definition file contains the grammar rules, as well as, the procedures to be executed against each of the rules.
These rules, allow the construction of the AST which will be then used by the semantic analyzer and interpretation engine to actually process the model. The tree is built using the nodes defined in the Orml.Interpreter.Structures namespace which are generic enough to then allow the programmer to implement its own mechanisms to process it.
It is also function of both the syntax and the lexical analyzers to signal eventual errors that they may find during this phase. This is done using an error signaling method which abstracts the programmer from how the errors are stored, processed and what information to keep about them. The syntax analyzer's error handler has simple-to-state goals:

- Report the presence of errors clearly and accurately;
- Recover from each error so that it is possible to detect subsequent errors;

- Avoid slowing down the processing of correct programs

ORML Interpreter

The Orml.Interpreter namespace builds on top of the Orml.Interpreter.Library namespace adding the semantic analysis and the interpretation. It provides an API that actually allows to run the interpreter over a source file and obtain results programmatically. It does, however, lack the ideal abstraction level to be used in the development of external applications.

The semantic analysis consists in checking types, identifier declarations, domains and overflows, as well as, incorrect model parameters and many other semantic rules. It is executed immediately after the AST construction and has to be executed before the interpretation itself since it is here that the symbol table is built. Although this phase could actually be done together with the interpretation itself, the chosen 2-phases approach gives a higher level of abstraction and makes the system more modular.

The decision to have semantic analysis was taken after having a careful look at what exists in the market concerning optimization languages. Most of them don't include this step and that makes it harder for the user to code, detect errors and debug models.

Concerning the interpretation engine, it is responsible for processing the models and returning the results after solving them. To do so, it makes use of the solver technologies provided by the Microsoft Solver Foundation.

The integration with this technology will be made using the services provided by the SFS API and will mainly consist on the translation of the models from the ORML language format to the syntax accepted by this framework (OML) with the necessary interpretation required to do the data-binding against the specific Dynamics AX sources through the Business Connector .NET.

By using these services, it is possible to abstract from the specifics of the solvers like, for instance, the definition of the coefficient matrixes for the constraints and/or goals. To better understand what this is consider the example of some simple constraints in a possible LP model:

$$x <= 4 \tag{5.1}$$

$$3x + 2y <= 18 \tag{5.2}$$

$$2y <= 12 \tag{5.3}$$

This would be translated to the following coefficient matrix:

$$\begin{pmatrix} 1 & 0 & 4 \\ 3 & 2 & 18 \\ 2 & 0 & 12 \end{pmatrix} \tag{5.4}$$

Figure 5.6: Example of a coefficient matrix

Regarding the specific AX data sources, they were divided in two types:

- Tables/Views
- Query-like classes

The first, was accomplished through the usage of the Common Data Layer which will be further describes in section 5.1.3.1. As a note, in this phase, it worth noticing that, through the regular Create/Read/Update/Delete (CRUD) operations are simplified.
Concerning the second, it's necessary to understand what a Query-like class is. Query-like classes are X++ classes, implemented in the Dynamics AX which implement a specific interface. Their goal is to resemble what is commonly known as stores procedures in DBMS.
This approach was defined when it was perceived that direct interaction with the data-base through the regular .NET stack (i.e. using ADO.NET or even LINQ) would be impossible due to reasons already presented in this chapter. Also, it's worth noticing that the "Queries" already provided by the AX system are not especially suited, nor sufficiently extendable, for the wanted usage.
Regarding the interaction with this type of data-sources, it is possible through the direct usage of the .NET Business Connector which provides an interop platform between .NET and X++ code.

ORML Interpreter Interface

The Orml.Interpreter.Interface namespace provides an easy-to-use interface to the interpreter itself. Built on top of the Orml.Interpreter namespace, it has flow control over the its different phases allowing the user to focus on supplying a source model and obtaining the results in an easy-to-use API. This API is actually used by both the IDE and the console application.

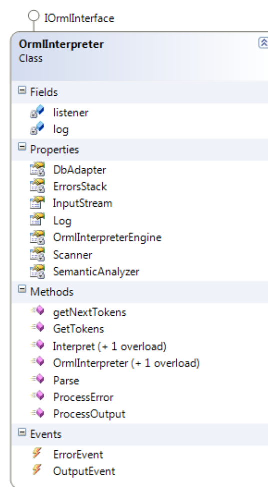

Figure 5.7: ORML Interpreter - Interface - Class Diagram

ORML Interpreter Interface AxWrapper

The Orml.Interpreter.Interface.AxWrapper was designed to make the integration with the Dynamics AX product easier to acomplish. Through it's usage, and with only a couple of lines of code, it is possible to create and execute a model or a model instance directly from X++ code. This makes the models usable throughout the application in transparent way.

The ability to run models, since the data-type conversions from X++ to .NET aren't complete (table 5.1 that represents the ones that are) was one of the design problems to deal with in the implementation of this namespace. To pass the parameter Dictionary used in the .NET implementation, the option was to design a method that would receive the name value of the parameter and apply it to every parameter.

Dynamics AX data type	.NET Framework data type
String, RString, VarString	System.String
Integer	System.Int32
Real	System.Double
Enums	System.Enum
Time	System.Int
Date	System.Date
Container	AxaptaContainer
Boolean (enumeration)	System.Boolean
GUID	System.GUID
Int64	System.Int64

Table 5.1: Data Type Mappings

After the parameter attribution, it is then possible to run the model instance. If the user wants to run a model instance directly, there is no need to specify parameters and the instance can be run directly.

ORML Interpreter Console

The console application is a simple application that makes use of the interface library and gives a standalone way of using the interpreter while providing access to many of the interpreter's features. This includes the export functionalities to both MathML and OMML. This proved to be a good approach for the first development stages, when the *ORML IDE* wasn't still available.

5.1.1.2 Development View

The development view, contrary to the Logical View which is mainly at a conceptual level, represents the physical-level artifacts. For ease of development and modularity, the project was divided in different *namespaces* and Dynamic Link Librarys (DLLs).

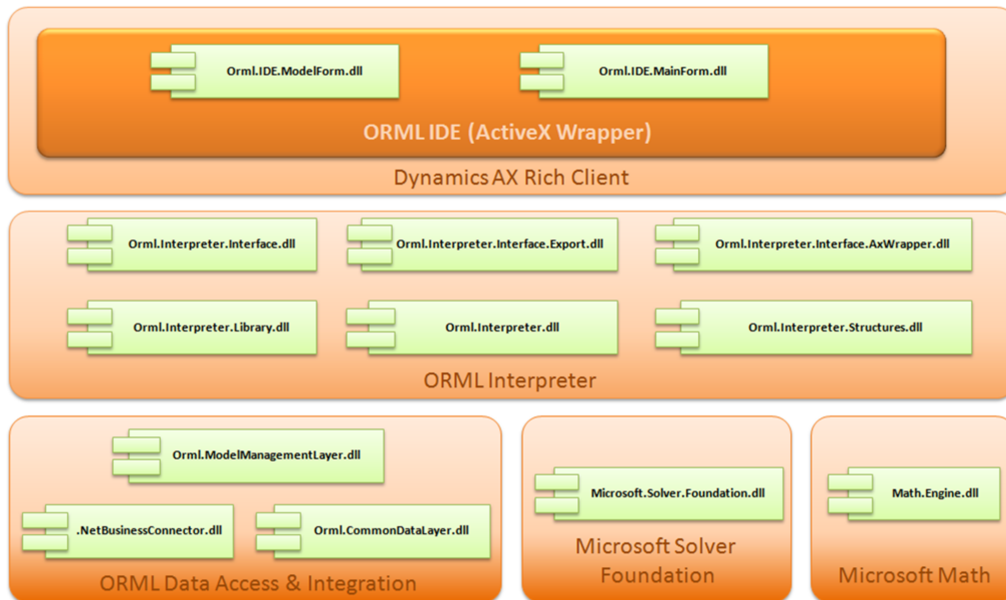

Figure 5.8: ORML Interpreter - Component Distribution

By taking this is approach it is easier to use different parts of the program and to replace them. This view, complements the logical view, which already presented the overall system entities. Both of them serve as an input to the development process.

5.1.1.3 Process View

The process view considers non-functional aspects such as performance, scalability and throughput. In order to evaluate such aspects it's required to identify the main processes in the systems. For the *ORML Interpreter* the main process is the interpretation of an ORML model. Like previously presented (section 5.1.1), the proposed interpreter executes 5 main steps which can be divided in 4 main activities plus outputting the results.
There is, however, the possibility of not being able to execute all of them when the user submits a model to the interpreter. The most common reason for this is the presence of errors which may or not be critical to the interpretation process. The following diagram presents the different flows between the activities:

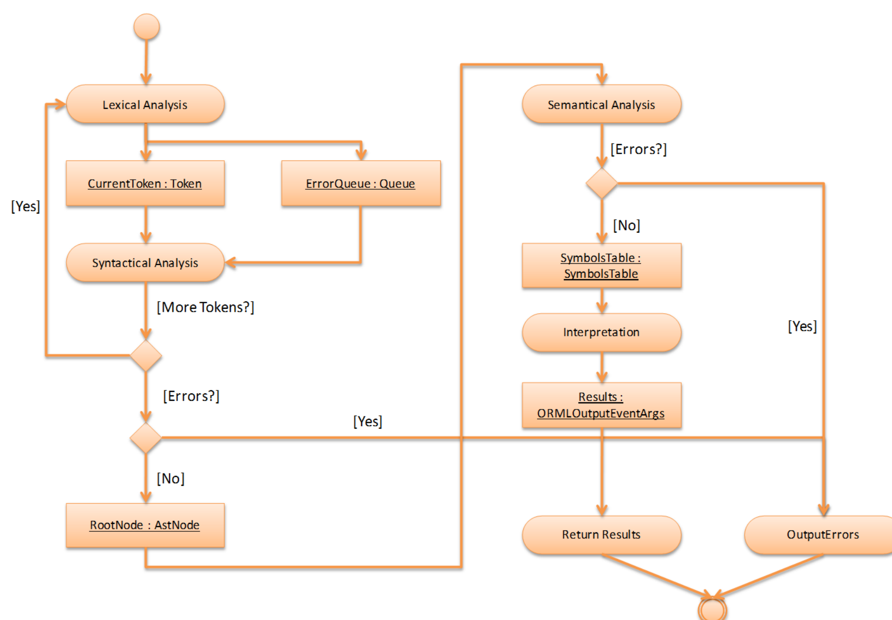

Figure 5.9: ORML Interpreter - Activities

An important part of the interpreter is its front end conversion from the source code into the AST. After the definition of the types of nodes (section 5.1.1.1), an example situation can be presented:

```
1 Model {
2   Variables:
3     Integer in [0 .. 1] a;
4     Integer in [0 .. 1] b;
5   Maximize:
6     a + b;
7 }
```

Figure 5.10: ORML Model - Example

This model has two sections: variables and function. The variable section is constituted by two declarations and the objective function by one objective. This can easily be translated to the AST of figure 5.11.

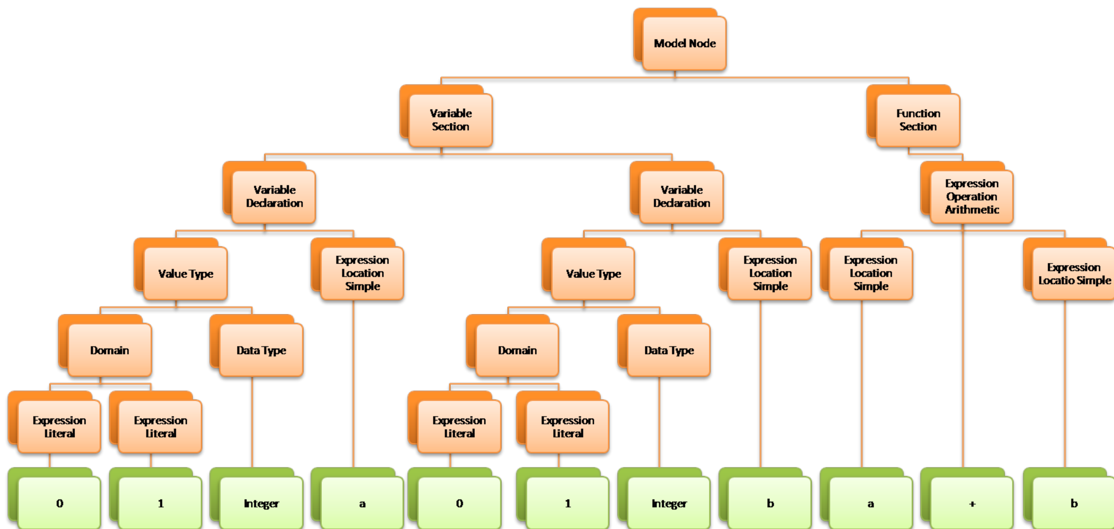

Figure 5.11: Figure 5.10 corresponding Arithmetic Abstract Syntax Tree

Knowing that this is the main process of the developed product, all the four steps should be optimized with respect to the non-functional requirements. It's worth, however, mentioning that since this is a prototype, these aren't actually critical objectives for the project and it was defined as more important to implement a flexible design, which can be easily changed to improve and test the language, than to have a performer system.
Finally, one should also note that, since the ultimate solving steps will be done by an external application (Microsoft Solver Foundation) it is out of the reach of this project to optimize on that aspect being the only possibility to give the input in the best possible way.

5.1.1.4 Physical View

This view encompasses the nodes that form the system's hardware topology on which the system executes; it focuses on distribution, communication and provisioning.
The deployment of the *ORML Interpreter* can be easily bounded to the deployment of the AX Rich Client since this is the target platform.

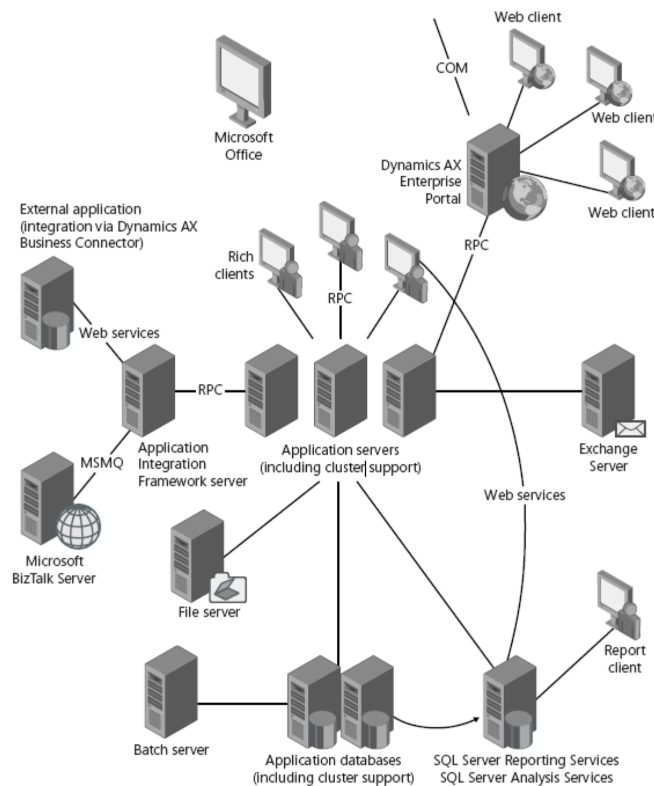

Figure 5.12: Dynamics AX Deployment Scenario

The *ORML Interpreter*, through the common data layer and the .NET Business Connector can communicate with the application servers.

5.1.2 ORML Interpreter Services

5.1.2.1 Syntax Highlight Service

The syntax highlight service intends to help the user identifying what kind of tokens he's using and overly making the model easier to read and understand.
The basic algorithm behind it consists in running the lexical analyzer, keeping the positions of each token, and then providing this information to external applications (e.g. *ORML IDE*).

5.1.2.2 Autocomplete Service

The autocomplete service objective is to help the user in writing models faster and safely. The overal quality of the results and modeling experience tends to improve with the amount of information this service can get about the context of edition.
The decision was to base this system in the syntax analyzer engine. This way, it is easy to retrieve context information based on the grammar rules. Consider the situation of figure 5.13.
In this situation, and considering that the cursor is currently after the "" character, the

```
1 Model {
```

Figure 5.13: Autocomplete Situation Example

system should sugest the various types of namedstatement as possible completions (e.g. the keyword "Variables").

5.1.2.3 Export to MathML and OMML Service

The Orml.Interpreter.Interface.Export namespace provides the export features supported by the Interpreter. The export functionalities are partially supported by the Microsoft Math Engine Library. This Library, like it was presented in the state of the art, contains conversion mechanisms between different mathematical notations. By adding support to the ORML Language it was possible to make it easier to convert to most (if not all) of the different supported formats. The way it works is similar to a compiler - The library receives an input representation in some format, converts it to an intermediate representation similar to an AST and then back to the output representation format. Unfortunately, and since this is proprietary code, complete details on this architecture cannot be subject of this report.
Conversions between MathML and OMML will be carried out using Office's own XSLT documents that do this transformation in both ways. After having the OMML outputing the results to an OOXML file using the Open XML Formats SDK (section 3.6.2.1) becomes a straighforward operation.
Figure 5.14 summarizes the activities as well as the technologies used in the "Export to OMML" (Int-UC08) feature.

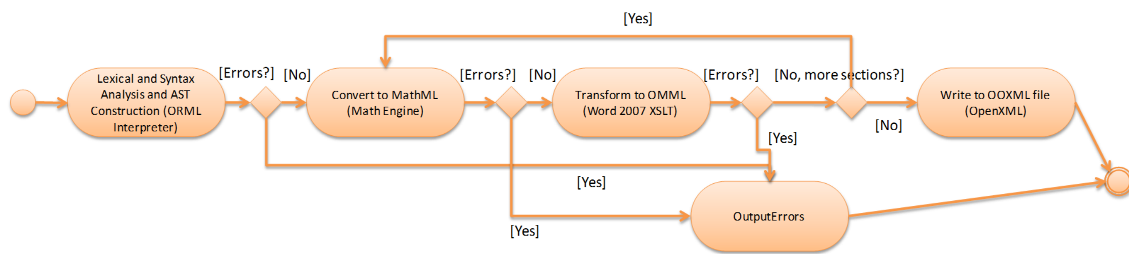

Figure 5.14: ORML Interpreter - Export Activities

Regarding the ”Export to MathML” feature, since the format doesn’t support all the necessities in correctly representing a model in a single file, the option was to write to different files the different formulas within the model.
It’s worth noticing that, for this specific feature, there was some contribution from the *ORML IDE* project in the initial design phase. After the initial analysis of the Microsoft Math Technology the implementation carried on, independently by this project.

5.1.3 Common Libraries

The Common Libraries are each bundled in one DLL that supports in some way the overall system. They’re common because they support both the interpreter and the IDE directly. Its architecture and development was, thus, a joint effort of both these projects.

5.1.3.1 Common Data Layer

The Common Data Layer implements a level of abstraction over the access to the database, allowing the software to easily commute between using AX Business Connector or another kind of provider like pure ADO.NET.
By default, the system will connect to the AX database through the AX Business connector. This decision was based on the fact that the AX Database doesn’t have any notion on business logic or even relations (foreign keys, for instance, are inexistent in the database). Its design was mainly based on the adapter pattern to allow the necessary abstraction from the database provider, although some other common design patterns were also put in practice like the factory method to request rows from a table or view or the template pattern to abstract the usage of the different elements (tables, views or classes).

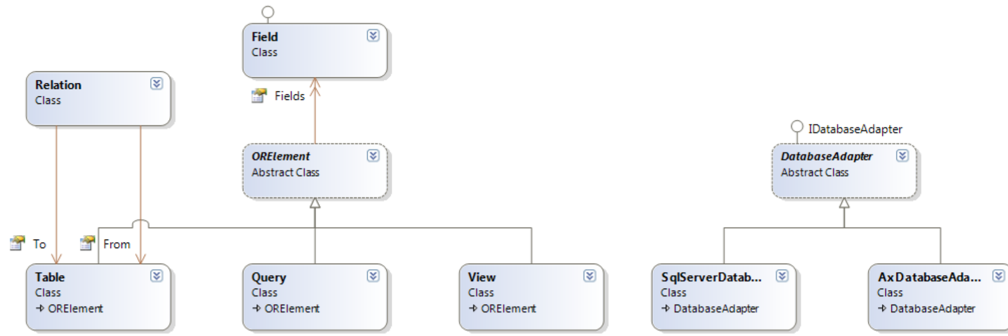

Figure 5.15: Common Data Layer - Class Diagram

5.1.3.2 Model Management Layer

The model management layer goal is to allow the necessary level of abstraction over the way the models are managed and stored. It's design was based on using the previously defined common data layer which makes it effectively database technology independent. The model management layer supplies the classes and methods necessary to manage both models and model instances. Models represent the actual definition of an ORML model and can be defined to have parameters. These can parameters can be instantiated in model instances which can also be stores, in a ready to run way, in the same data storage system as the models.
A class diagram for this library is presented in figure 5.16.

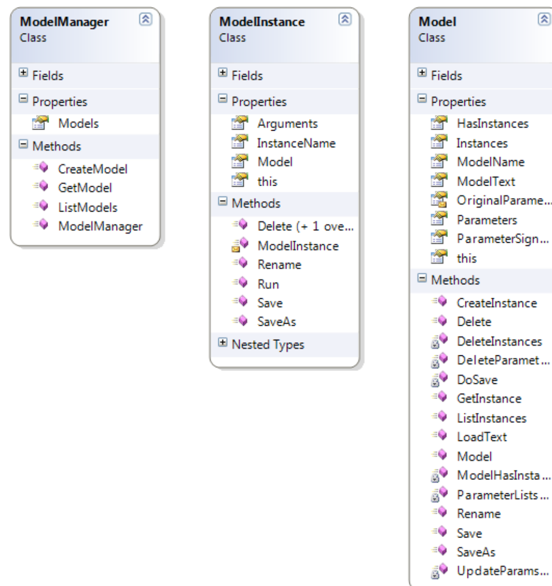

Figure 5.16: Model Management Layer - Class Diagram

In order to store the models in the Dynamics AX server, it was necessary to also define some new tables. Figure 5.17 represents the tables that were created, as well as, the relations between them.

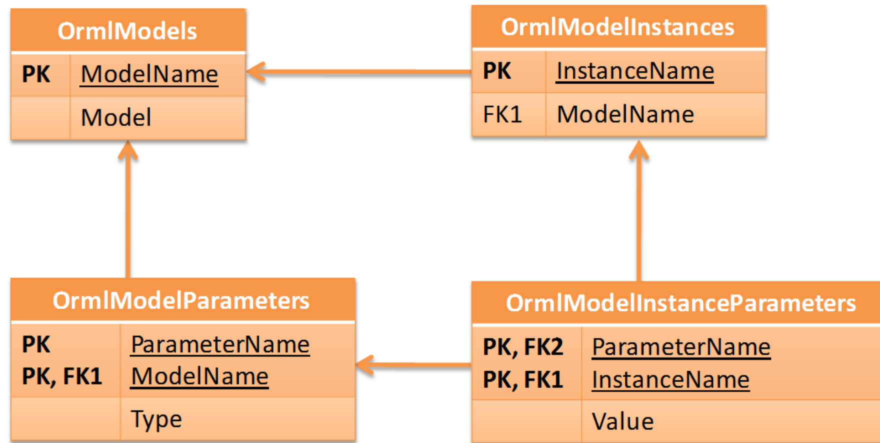

Figure 5.17: Model and Model Instances Entity-Relationship Diagram

Besides the general model and model instance managing features, another relevant feature in this library is the fact that, relying on the interpreter, the model instances have a method to solve them. This creates the following possibilities:

- Solving a saved model by fetching it from the database, creating an instance, setting the arguments values and invoking Run();
- Solving a saved model instance by fetching the model, from it, get the required instance and invoking the method Run().

These are, in fact, the most interesting operations for the *ORML Interpreter*. They are especially used when running the models from the console application.

5.2 Implementation

This section discusses in deeper detail how the implementation actually took place, describing the main application classes and their respective features, referring back to the issues discussed in the previous design section when relevant.

5.2.1 ORML Interpreter

The *ORML Interpreter* represents the main component of this project. Its implementation was supported by the usage of the MPLEX and MPPG tools, as well as, other technologies

as the Microsoft Solver Foundation and specific libraries of the Microsoft Math application.

5.2.1.1 Structures

The implementation of the AST classes is a direct result of the identified token types and grammar rules of the language.
The separation and abstraction of the structure from the concrete algorithms that will execute on it is an important design option that was addressed using the Visitor. The IVisitor and the IVisitable interfaces specify the required infrastructure to implement the concrete AST node classes, as well as the different visitor classes, with the desired amount of abstraction.

IVisitor

The IVisitor interface was designed to be implemented by the different AST visitors. The visitors will execute specific operations over the AST nodes and may have different purposes. They contain all the logic that is required to do the operations while keeping the structure free of it. A visitor has then to implement a specific method for each type of concrete (non abstract) node that may exist in a given AST. This insures the right amount of flexibility required for the implemented interpreter.

- void Visit(ModelNode node) - Visits a node of the ModelNode type and executes some operation with it. This includes, usually, do recursive calls to also process inner nodes (recursive descend).
- ...
- void Visit(IndexDeclarationNode node)

This interface is implemented by both the semantic analyzer and the interpretation engine classes.

IVisitable

The IVisitable interface was designed to be implemented by the AST node classes. By implementing it, the classes are supplying a common way of accessing them which can then be used by the specific visitors to operate over them, thus keeping all the algorithmic details out of the structure itself.

- void Accept(IVisitor visitor) - Accepts a visitor class and the method definition will call the correct visitor method back based on what class is accepting it.

With this structure defined the required AST node classes and the dependencies between them were identified. This resulted in the implementation of an abstract class that would contain the methods and attributes common to all the nodes in an AST - The AstNode class.

- public Position Position { get; set; } - Defines the position of a given node in the source file. This information is mainly useful for error signaling.
- public AstNode NextNode { get; set; } - Nodes usually have references to others nodes in the same scope level (example: sections / named statements in a model). This property allows building a linked list between the different nodes.

The specifics of each node concerning possible Childs are implemented in the specific node. This is possible since the visitors have the information about what kind of node they are visiting and thus is possible to properly navigate through them.

Symbols Table

The symbols table allows a compile/interpreter to keep track of scope and binding information about names. It is searched every time a name/identifier is encountered in the source text and changes occur if a new name/identifier or some new information about one has been founded.
There are two common approaches to implement a symbols table: a linear list or a hash table. Although the linear list is easier to implement, it performs poorly in regard to additions and consults when compared to an hash table, especially for bigger programs. Hash table, however, provide a better performance at the cost of greater programming effort and space overhead. One should also keep in mind that a symbols table should have a dynamic size so that it can handle any program that might be presented.
The option, for the *ORML Interpreter*, like presented in the previous chapter, was to implement the symbols table using a Hash Table.
Each entry in the hash table represents a declaration of a name and it contains specific information regarding the type of declaration: variable, input, index or inline variable.
To support the ORML language inline variables, used, for instance, in iterations, it is necessary to have the notion of scope of declaration. The scope is defined as the enclosing context where values and expressions are associated. A scope declaration cannot, therefore, be used in the parent scope. To support scopes in a symbols table, different strategies may be used. The chosen one was to maintain a separate symbol table for each scope. In effect additions are done to the current symbols table and lookups are recursively done starting from the current to the oldest (with the largest scope) symbols table returning the first match. Tables are then maintained in a Stack and have information regarding the node where they were defined. This way, when a new scope is started, the method can

verify if there is already any scope defined for such node and, if not, create a new symbols table and put it on top of the stack.

- public SymbolsTableEntry<Type> Enter(string name, Type value) - Enter a new symbol in the current symbols table (in the current scope);
- public SymbolsTableEntry<Type> Lookup(string name) - Looks up a symbol in the current symbols table stack;
- public void BeginScope(AstNode node) - Begins a new scope by creating a new symbols table for the given node or by retrieving the previously declared one, depending if there is already one or not;
- public void EndScope() - Terminates the current scope by removing the current symbols table from the active symbols table stack.

It is worth mentioning that the proposed *ORML language* interpreter doesn't however support in-scope declaration of a previously defined name. This was purely a design decision based on the fact that this could make the model harder to understand and, if wanted, is a restriction that could be easily withdrawn.

5.2.1.2 Lexical and Syntax Analyzers

The Lexical Analysis is the first phase of the *ORML interpreter*. It is Lexical Analyzer's task to identify and pass the tokens in the source file by Syntax Analyzer's requests. Its implementation was based on the MPLEX tool that is currently shipped with the *Visual Studio 2008 SDK*. This tool, which is "lex-like" accepts a specification and produces a C# output file.
Regarding the sections in an MPLEX specification file (section 3.8.1), one important part is, clearly, the patterns definition. Patterns are regular expressions [91] which are special text strings for describing search patterns. They work like extended *wildcards* so if the reader is familiar with *wildcard* notations such as "*.txt" to find all text files in a file manager, the *regex* counterpart would be `".*\\.txt"`.
It lies outside of the scope for this report to present all the regular expressions and tokens that were defined for the ORML language. To illustrate the subject, a few examples can, however, be presented:

- 0—[1-9]{digit}* - Regular expression used to recognize an integer literal. The regular expression is defined to match the number 0 or a sequence of numbers that start with a number from 1 to 9 and that has any number of digits. Possible overflows are dealt later in the rule specific user code.

- Maximize - Maximize keyword. Identifies the string "Maximize" like it is defined. It's important to note here that this means that the keywords are case sensitive. This was also a design option and resembles common programming languages like C# [92].

- >= - Equals or greater than operator. Identifies the ">=" token which represents the equals or greater than operation.

The Syntax Analyzer uses the information about the tokens returned by the Lexical Analyzer and, matching them against the language grammar rules, builds the correspondent AST. The Lexical Analyzer was built using the MPPG tool that is also shipped, together with the MPLEX tool, in the *Visual Studio 2008 SDK*. Its grammar is based on the traditional YACC language.
Also, outside of the scope for this report, lie the syntax grammar rules. To illustrate the subject, once again, a few examples can be presented:

- ModelStatement: KWMODEL LCURLYBRACE NamedStatementList RCURLYBRACE - Rule for a model statement. A model is defined by the Model keyword following by a list of named statements enclosed in curly braces. Upon identifying this rule, the NamedStatementList is processed and a model node is built in the AST with the enclosed named statements.

- Literal: INTEGER - Rule for an integer literal. Upon identifying this rule, the integer value is extracted and a literal node is built in the AST.

5.2.1.3 Semantical Analyzer and Interpretation Engine

The ORML.Interpreter namespace is mainly defined by the SemanticAnalyzer and OrmlInterpreterEngine classes.

Their external structure is quite similar since both of them represent visitors (considering the visitor pattern in order to operate over a given ORML AST. They both begin their analysis process starting from the root (model) node and recursively descend the tree operating on the consecutive nodes.
Since one of the main tasks for this interpreter is the data-binding, this is consecutively an important part for both the Semantic Analyzer and the interpretation engine.
Like previously presented in section 5.1, the binding to data sources is accomplished through the Common Data Layer.
The binding for the Query-like classes, on the other hand, it is accomplished through direct method calling of the classes. This is possible through the usage of the following methods that the classes define when implementing the required interface:

- anytype getField(str field) - This method is used to get the current row's field value based on its name;

- Array getFields() - returns an array of all the fields that are available in the current row;
- str getFieldType(str name) - returns the type of a field in the current row (example: ”String”);
- boolean moveNext() - Changes the current row to the next row if available (returns false if there aren’t any more rows)

Regarding the semantic analyzer’s basic operations, it consists in traversing the AST and, for each node visited, returns its resulting type and, when the parent is visited, this type is verified. If valid, it is used to form the new return type of the current node, if not an error is signaled. Figure 5.18 presents an example of such procedure for a valid expression.

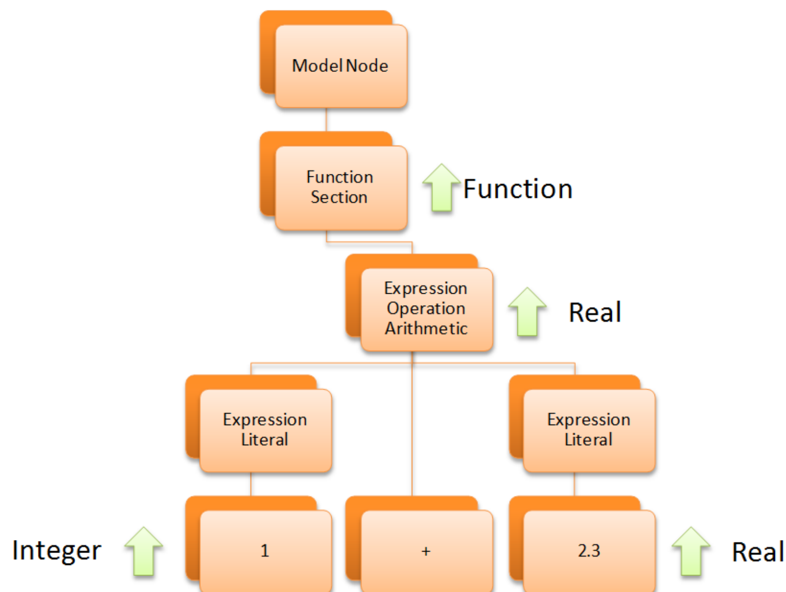

Figure 5.18: ORML Interpreter - Semantic Analyzer - Check Example

Besides the type checking operations, some other semantic checks are also performed, being, most of them, based on checking identifier declarations:

- Duplicate identifier
- Undeclared identifier
- Wrong type identifier

These are performed using the symbols table that stores, at each declaration, the name and type of the identifier, eventually, together with some other information.
Upon finding one identifier declaration in the source code the steps in algorithm 1 are performed.

```
Input: AST node
Output: AST node type
begin
    if identifier found then
        if declaration then
            if identifier was declared then
                return duplicate declaration error
            end
        else
            if identifier was declared then
                return identifier type and, if it is the case, value
            else
                return undeclared declaration error
            end
        end
    end
end
```

Algorithm 1: Semantic Analyzer - Identifier Checks

Regarding the interpretation engine, it was the main component to be implemented in this project. It works in a similar way to the semantic analyzer but, instead of doing checking, it instantiates and stores declarations values and processes constraints, variables and functions submitting them, in the desired format to the Microsoft Solver Foundation Rewrite system.
This system allows the user to submit well defined problems which are then solved by the framework.
Unfortunately, due to the previously presented reasons (section 3.3.2), further details, on this specific component aren't possible on this report since it would disclose Microsoft Solver Foundation details.

5.2.1.4 Interpreter Interface

The *ORML Interface*, defined in the ORML.Interpreter.Interface namespace, provides the necessary abstraction over the Interpreter system. It allows third party application to easily communicate and make use of the system by providing access its functionality through a common interface.
The most noticeable and relevant methods are exposed by the OrmlInterpreter class:

- public OrmlInterpreter(StreamReader inputStream, DatabaseAdapter dbAdapter) - Builds a new instance of the *ORML Interpreter* Interface Class. The user should specify the input stream that he wants to process and the database adapter that should be used during the interpretation for the data retrieving and binding.

- public AstNode Parse() - Performs the Lexical and Syntactical Analysis on the specified source stream.
- OrmlStatistics Interpret(AstNode astRootNode, Dictionary<string, object> parameters) - Performs the Semantical Analysis and Interprets the specified AST and parameters.
- public OrmlStatistics Interpret(AstNode astRootNode, bool process, Dictionary<string, object> parameters) - Performs the Semantical Analysis and/or Interprets (depending on the process flag) the specified AST and parameters.

5.2.1.5 Interpreter AxWrapper

The Orml.Interpeter.Interface.AxWrapper namespace makes the integration with the Dynamics AX product code easier.
The wrapper defines the following methods:

- public void InitWrapper(string modelId, string company) - Inits the wrapper for a specific model in a specific company database;
- public void InitWrapper(string modelId, string modelInstanceId, string company) - Inits the wrapper for a specific model instance in a specific company database;
- object GetParameter(string name) - Gets the current value of a given parameter;
- void SetParameter(string name, object value) - Set a new value for the given parameter;
- InterpreterRunResult RunModel() - Runs the specific model with the, already defined, parameter values.

By using this wrapper it is possible to, with just a few lines of code, run and obtain results from the *ORML Interpreter*.

5.2.2 ORML Interpreter Services

This section presents a brief overlook on how the *ORML Interpreter* services were implemented and how can they be used in-code.

5.2.2.1 Syntax Highlight Service

The syntax highlight service, like presented in the design section, consists in one method that runs the Lexical Analyzer in one model source text and provides the list of tokens

that are identified, together with their position within the text. This method was defined as follows:

- public IEnumerable<ColorToken> GetTokens()

5.2.2.2 Autocomplete Service

The *auto complete* service is based on the Lexical and Syntax Analyzers. Based on the current state of the Syntax Analyzer, the list of possible suggestions is updated with possible tokens and, if there is the possibility of identifiers following, the previously declared identifiers are added to the list.
The service can be used from the previously discussed Interface class and consists in the following method:
public List<ColorToken> getNextTokens(out int offset) - Auto complete service. Supplies the next possible tokens based on the specified input stream text and on the given cursor offset.
Following the implementation of the autocomplete engine on the *ORML Interpreter*, the service had to be implemented on the IDE as well. To do this, the custom AutocompletePopup user control was created. This component could be used from the CodeEditor user control, that represents the code edition area and that was created, as a result of the *ORML IDE project*, as part of its core infrastructure.

5.2.2.3 Export to MathML and OMML Service

The export features are supported by the Orml.Interpreter.Interface.Export namespace. This namespace provides the necessary abstraction over the operations to export to both MathML and OMML. To convert a model to this formats the user must first instantiate the class OrmlExport with the default constructor:

- public OrmlExport(string modelText) - Export to a string.

Then, the class provides the methods necessary to export to the required formats:

- public void ConvertToWord(string outputFile) - Converts the model to the OOXML format and writes it to a string.
- public void ConvertToWord(string outputFile) - Converts the model to the OOXML format and writes it to a stream (e.g. file).
- public MathMLModel ConvertToMathML() - Returns an MathMLModel which contains the various sections of the model.

5.2.3 Common Libraries

This section presents some implementation details regarding the common data and the model management layers.

5.2.3.1 Common Data Layer

The main operations that the common data layer provide, are based on the DatabaseAdapter abstract class. Using it, the user can obtain table, views or query-like classes from the system and, based on these, it is possible, for instance, to select, insert or update rows.
A normal Common Data Layer (CDL) usage session begins by constructing the specific implementing adapter class. If the user wants to target a Dynamics AX installation, through the Business Connector (which is the only option that was actually implemented), it can be done through the specific constructor:

- public AxDatabaseAdapter(string company, string language, string objectServer, string configuration)

With this object, it is then possible to retrieve regular elements using the following methods:

- public Table GetTable(string name)
- public View GetView(string name)
- public View GetQuery(string name)

It is then possible to operate over these elements using, for instance, the following methods:

- public IEnumerable¡TableRow¿ GetRows(Table table, TableRow match)
- public void Insert(Table row)
- public void UpdateRows(Table table, TableRow newValues, TableRow match)

5.2.3.2 Model Management Layer

The model management layer library is mainly used, in this project, by the Console application to execute models directly from the database. To do this, two main methods have been created to execute models and model instances respectively.
Independently of the method, the first step is to instantiate the ModelManager by providing a CDL adapter.

- public ModelManager(DatabaseAdapter adapter)

The next step is to get the model to be run or that will provide the correct model instance. This can be accomplished with the method:

- public Model GetModel(string modelName)

From the model, it is possible to then get (if the user wants to run an already existent) or create (if the aim is to run the model with some specific parameters) a model instance.

- public Model GetModel(string modelName)
- public Model CreateModel(string modelName)

Finally, the model instance can be run.

- public InterpreterRunResult Run()

5.3 Development Methodologies

Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software [93]. Engineering Excellence deals with studying the best approaches in performing that activity. By systemizing methodologies and best practices it is possible to improve the overall product quality while reducing costs and efforts.
This section presents some of such methodologies, best practices and tools that were put in use to enforce them. This isn't however, an extensive description on the subject, only highlighting the ones that were identified as the ones of the most importance.

Test Driven Development

TDD [94] [95] [96] is related to the test-first programming concepts of extreme Programming which begun in the late 20th century. It is a software development technique consisting of short iterations where test cases are written before implementing the desired improvement or functionality. Then, the production code necessary to pass the tests is implemented and finally the software is refactored to accommodate changes.
Among its advantages one can highlight the rapid feedback after any change and its use not just as a mere test method but also as a method of designing software.
The TDD Cycle can be divided [97] in the following steps (slight variations exist among TDD practitioners):

1. Understand the requirements of the features one wants to work on;
2. Create a test and make it fail - This test must inevitably fail because it is written before the feature has been implemented. This ensures that the developer understands

the specification and the requirements of the feature clearly. This differentiation against writing test after the code is written, makes one focus on the requirements before actually writing any code which is a subtle but very important difference;

3. Make the Test pass by any means necessary - this involves writing the production code to make the test pass. In this step, some advocate the hard-coding of the expected return value first to verify that the test correctly detects success. It is important that the code written is only designed to pass the test; no further should be predicted because that may make to user to disregard eventual test scenarios;

4. Refactor - Change the code to remove duplication in the project and to improve the design while ensuring that all tests still pass;

5. Repeat the cycle (2-4) - Each cycle should be very short.

This project has used a TDD Approach from the start. Since the project's early stages, when the language was being defined, it was clear that the example scenarios created for the language could help the design and test process in the product development. By creating small test scenarios and evolving them it was possible to ensure that the design was properly done before actually implementing the code. This also assured that if the language changes, everything that was already coded can be verified against the introduced changes, thus detecting regressions [98].
This is a small list of the benefits immediately drawn from performing Test-Driven Development:

- The tests provide constant feedback about whether each component is still working;
- The tests act as documentation that cannot go out-of-date, unlike separate documentation, which can and frequently does;
- When the test passes and the production code is refactored to remove duplication, it is clear that the code is finished and the developer can move on to a new feature;
- Forces critical analysis and design because the developer cannot create the production code without truly understanding what the desired result should be and how to test it;
- Software design tends to improve, that is, it tends to be loosely coupled and easily maintainable, because the developer is free to make design decisions and refactor at any time while making sure that the software is still working by running the tests;
- The test suite acts as a regression safety net on bugs;
- Reduced debugging time - Since the tests are created, run and the code incrementally verified, debugging time tends to diminish because of the reduced waterfall effect.

Version Control

In software engineering, revision control [99] is any practice that tracks and provides control over changes to source code. Software tools for revision control are increasingly recognized as being necessary for the organization of multi-developer projects.
Although Microsoft has had a rather known tool called Visual Source Safe (VSS) for version control, within Microsoft very few projects have relied in VSS. Source Depot, a custom version of Perforce[100], has been the most used system and it did in fact support many popular products.
Nowadays, Microsoft Developer Division is, however, starting to use the new Visual Studio Team System [101] for most of the internal project. This project, at the time of its execution, has, however, still relied in the Source Depot tool where the different versions were stored.

Nightly Builds

In software development, a nightly build is a kind of neutral build that takes place automatically, this is, is a build that reflects the current state of the source code checked into the version control system without any developer-specific changes [102].
This project has had from its very early stages nightly builds. This allowed to keep notion of eventual problems in the repository, keep track of work and ensure a better coordination with the *ORML IDE* project. By checking the generated logs it is possible to understand, each day, if something unexpected was done wrong in the previous day. The Nightly Build Process can be summed to the following operations:

1. Retrieve latest source code version from source depot repository
2. Store Source File
3. Build the Source Code
4. Store Binaries
5. Run Tests
6. Store Tests Results

If something during this process fails, the following phases will not be done and a log is kept to report the error. By doing so, nightly builds, also made sure that, if needed, the project team would always be able to easily find a stable version just by looking back to the most recent dates (some other advantages may be seen in [103] or [104]).
One should also note that, in the event of a build failing, the team corrected the problem

and execute a rebuild to try to keep, as least, one working build for each day.

Bug Tracking Tool

A bug tracking tool is a software application that is designed to help quality assurance and programmers to keep track of reported software bugs in their work [105].
Since this project was implemented together with the *ORML IDE* project and, during its development, has become clear that while testing one project; it is common to discover problems or enhancement suggestions in the other and given that it is not always possible to correct such things immediately, it was important to track all of these issues and keep them accessible to everyone.
The usage of a bug tracking tool, available in the intranet, through a web-page, has made this possible and easy to use. This has greatly improved both projects awareness on problems/requests while giving a general perspective on both projects progress.

5.4 Summary

This chapter presented the design and implementation details on the developed solution. Regarding the design details, it was defined that the *ORML Interpreter* executes the glsacro:orml language source and outputs its results directly. To do this, the interpreter performs the following operations: Lexical analysis, Syntax Analysis with the associated AST construction, Semantic Analysis and Interpretation.
The fact that the result of this project is a prototype and because it was given preference for Microsoft Tools, it was defined that the MPLEX and MPPG tools would be used to implement the Lexical Analyzer and the Syntax Analyzer respectively. Regarding the Semantic Analyzer and the Interpretation Engine, they are implemented as visitors of the AST structure.
To suppot the data-binding, the CDL library was implemented to the tables and views sources and a common structure to support X++ Query-like classes was defined.
It was then presented how the Interpretation Engine makes use of the Microsoft Solver Foundation to actually solve the modeled problems.
Finally, details regarding how the export to MathML and the export to OMML features were implemented. The export to MathML functionality is supported by the Microsoft Math Engine library and is further backed up by the Office's 2007 MathML to OMML XSL transformation documents and by the OpenXML framework to support the word document creation.

Chapter 6

Evaluation of the Solution

Put everything that you are, in the minimum that you do.

Fernando Pessoa[106]

This chapter presents an evaluation of the developed solution against its compliance to the goals of the project, its requirements, testing and benchmarking.

6.1 Testing

Software testing is used to assess the quality of software. It is conducted to provide information about the quality of the product, usually, by comparison against a specification. During this project some tests were developed to ensure that the result met the specification within scope and quality, while helping and supporting the actual development process.

6.1.1 Test Scope

Before beginning the implementation of the actual tests, it is important to precisely define what should or shouldn't be tested. When looking at the overall project, taking into consideration the design breakdown, three test targets can be identified:

- The *ORML Interpreter*;
- The services it provides;
- The Common Libraries

This section defines the scope for each of these targets.

6.1.1.1 ORML Interpreter

These are the tests where most of the effort was put. Following the TDD approach, these tests, have been designed progressively and before the actual feature was implemented. Concerning the lexical analyzers the tests should verify the ability to:

- Correctly identify the different tokens in a given input (Valid Scenario);
- Signal the eventual errors that may be detected (Invalid Scenario).

Regarding the syntax analyzer, the tests should be designed to test its abilities to:

- Correctly identify the grammar rules based on a correct list of token (Valid Scenario);
  - Build a correct AST;
- Signal the eventual errors that may be detected (Invalid Scenario).
  - Recover from errors and continue with the parsing;

The next tests aimed at testing the semantic analyzer. Their purpose was to test the following capabilities:

- Detect semantic errors:
  - Duplicate symbols;
  - Undeclared Symbols;
  - Inappropriate use of expressions.
- Signal eventual errors that may be detected during this phase.

Finally, the tests for the interpretation engine aimed at testing its ability to:

- Process a valid AST and give appropriate results:
  - Read inbound data;
  - Declare the declared parameters, inputs, indexes and variables;
  - Recognize and process the objectives;
  - Recognize and process the constraints;
  - Write outbound data.
- Output results.

Some tests were also designed to test the solution's performance against both the models used in the project and, some other, common, optimization models.

6.1.1.2 ORML Interpreter Services

Some autoamted tests were also developed to test the export functionalities in the *ORML Interpreter* Services. These tests target specifically the ”export to OMML” features which, indirectly, also tests most of the ”export to MathML” feature.
Regarding the *autocomplete* and *syntax highlight* services, the option was to do manual testing on them by actually using them in the *ORML IDE*.

6.1.1.3 Common Libraries

Some unit tests were also developed for the Common Libraries. These tests where developed together with the *ORML IDE* project and intended to verify that the basic CRUD functionalities before they were put into use.

6.1.2 Test Strategy

The test strategy defines the testing procedures, tools and conditions. For this project, there were to main types of testing:

- Automated tests - Unit tests were developed to cover most of the product functionalities like the interpretation process, common libraries and export features
- Manual tests - Although automated tests are enough for most of the features, manual testing was also used, mainly, to test the *autocomplete* and *syntax highlight* services.

6.1.2.1 Testing Procedures

This section presents the testing procedures used to test the different components of the developed solution.

ORML Interpreter

The option was to classify and document each test by category and description so that one could easily track which test is failing and what is it failing on. This followed a divide-and-conquer[107] test approach which attempted to make sure each component of the interpreter is properly tested before moving to the next one. Figure 6.1 represents the hierarchy of such division into categories.

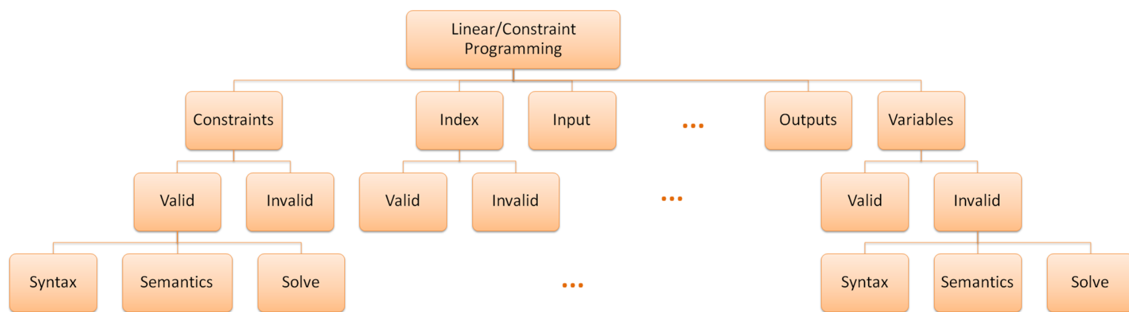

Figure 6.1: Testing Categories

ORML Interpreter Services

The approach taken to test the *ORML Interpreter* services, and in particular, the ”export to OMML” capabilities, was to develop specific models which could support the development of the different parts of the features and, in a later stage, extend the testing to all the test scenarios that were already used for the *ORML Interpreter*.

Common Libraries

The test strategy for the *Common Libraries* was to develop unit test that could test, mainly, its CRUD operations.

6.1.2.2 Test Tools

Currently, Visual Studio Team System 2008 offers all the tools required to do Test-Driven Development [94]. The creation of unit tests is possible through generation from the source code or by authoring them by hand. It is then possible to group tests in different lists and run them separately.
Additionally, code coverage features are also available to further ensure the correct test coverage.

6.1.3 Test Resources

The tests were mainly performed in two machines (Appendix G). The automated tests were mainly performed on the Quad-core machine and the manual tests were performed in both machines.

6.1.4 Test Results

This section presents the main results that were obtained through the, previously defined, testing strategies.

6.1.5 ORML Interpreter

Following the defined strategy, the following tests were implemented and tested successfully for the *ORML Interpreter*:

- Lexical and syntax analyzers - 45 *Valid* scenario tests and 12 *invalid* scenario tests (approximately 82% code coverage);
- Semantic analyzer - 34 *Valid* scenario tests and 20 *invalid* scenario tests (approximately 80% code coverage);
- Interpreter engine - 58 *Valid* scenario tests (approximately 80% code coverage).

Concerning the tests that were presented, it's important to notice that what they intend to test is incremental and, thus, the semantic analyzer tests will also test the lexical/syntactical analyzer capabilities and the interpretation tests will also test the semantical/lexical/syntactical analyzer capabilities.
During the development phase of the project it's important to note that the TDD methodologies were put in practice and tests were run before each commit to the repositories in order to enforce that the code that it is shared is actually executable and to help avoiding late detection of eventual regressions.
By the time this project was concluded, all the tests that were designed were passing, giving an extra confidence to the project's outcome. Like any other testing approach, one should, however, be aware that this doesn't mean that the product doesn't include bugs. It only means that the features one intended to have implemented, in the specific situations that they were tested, work.

Execution Times

Execution times tests allow to identify bottlenecks in the system as well as to evaluate its usability. Some generic problems were identified as to be benchmarked by the system. The chosen problems are among the most common in this kind of systems. Some AX Problems specific benchmarks were also performed to account with the data binding experience and, in fact, its overall goal to be integrated within the system.
Among the factors to take into account when creating or choosing the tests one can highlight the following:

- Type of problem: LP, MIP, CSP;
- Number of variables;
- Number of objective functions;
- Number of constraints.

Although a comparison between the model solving time can't be made due to the internal strategies that might be used even by the same solver, the absolute values may still contribute to the general understanding of the system. The complete definitions on these models may be seen in Appendix H.

Model	Variables	Objectives	Constraints	Lexical and Syntactical Analysis	Semantical Analysis	Interpretation
Constraint Programming						
Zebra	25	-	19	158 ms	36 ms	277 ms
Linear/Integer Programming						
Without Data Binding						
Boeing	384	1	440	276 ms	52 ms	5399 ms
PetroChem	2	1	7	153 ms	31 ms	427 ms
WycoDoors	2	1	3	158 ms	32 ms	422 ms
With Data Binding						
Traveling Salesman	42	1	32	164 ms	179 ms	621 ms
Warehouse Picking Routes	30	1	20	170 ms	581 ms	553 ms

Table 6.1: Execution Times Tests

Looking at the results, for the syntax and lexical analysis it's possible to confirm the guess that the bigger the model, the longer it would take to scan and parse. It's also possible to see that, usually, the semantic and interpretation phases take longer than these two first phases since they are where the model is actually solved and where the data-binding exists.

Without any test to testify this, it's also fair to admit that, for the same model, if data-binding is performed against a database, the model should take longer to process and

solve than the same model with explicit binding. Proof's on these ideas where actually never enforced during the project since they were considered to be out of scope, since performance wasn't a key issue. Their study would, however, be interesting for a final (non-prototype) product.
A comparison between these execution times and the execution times of the same problems in similar languages would be interesting. However, due to the time constraints, these comparisons where actually never performed.

6.1.6 ORML Interpreter Services

The defined strategy to test the *ORML Interpreter* services resulted in a total of 130 Tests. These tests helped ensuring that the MathML and OMML export functionalities worked within expected while making the test creation process fast by reusing many of same tests (that already included most of the intended language constructs that one should test) that were used in the other test scope.

6.1.7 Common Libraries

The test strategy defined for the Common Libraries conducted to 30 unit tests that tested mostly of the functionalities that were intended to be tested.

6.2 Market Requirements Analysis

The ORML language has been designed taking into account the requirements that were identified as the ones of most importance given the project scope: a proof of concept in the Dynamics AX system to evaluate its viability.
Considering the general requirements for new generation mathematical modeling languages that were presented in section 2.4, and although they weren't a goal for the project, some of them were tackled.

- Syntax - The ORML language has an easy to use language syntax. Although it is not complete, given that this is a prototype, it could easily accommodate further options.
- Solver Suite - The ORML interpreter engine has been designed using a Visitor pattern, this way, if a new solver is to be used, the user only has to implement a new engine.
- Indexing - Indexes are a distinctive part of the ORML language and, within memory constraints, any number of them is allowed.

- Scalability - By using variable sets, it's possible, and easy, to define a large number of variables in just a few commands.
- Robustness - Although the limited scope of this project, industry standard development methods have been applied to improve the robustness level of the solution.

Although these characteristics already give an good perception of such language, it is the belief of the author that, a generic final product, would, in fact, have to address most, if not all, of the initial design requirements.
The designed language, however, was targeting a specific platform and, given the time span of the project, was designed to address specific problems and serve its purpose as a proof-of-concept.

6.3 Modeling Experience

The modeling experience is an important part in a project of this nature. It is, in fact, pointed out as one of the defying factors in new generation mathematical modeling languages (section 2.4).
The overall language syntax has been designed to be easy to use and to be similar to solutions that already exist in the market (section 3.4).
To further enhance this experience, some services were also developed taking into account what already exists in the market and some feedback from potential users (Questions 5c and 5d - Appendix A).
This approach has conducted to good results which can be further verified by the acceptance of the product by the same potential users (Question 6 - Appendix A).
Further usability tests could and should be done to evaluate how the users really interact with the systems. Contrary to popular belief, some authors assure [108], in fact, that a small number os users can account for a very accurate result.

6.4 Summary

This chapter reviewed the implemented solution giving some notes on methodologies that were put in practice to ensure its successful outcome taking into account important criteria like quality and scope.
A test-driven methodology was used during the project development. This approach allowed ensuring a correct understanding of the requirements while supporting a good level of quality in the code produced. This methodology has been in practice while designing the tests.
The solution was then reviewed with regard to the requirements and the developed tests.

This chapter ended by presenting some early results on how the users perceive the system, their experience with it, in regard to user experience and outlining the fact that, in a small survey, everyone understood it's possible interest in future AX versions.

Chapter 7

Conclusions and Future Work

You're not thinking fourth dimensionally!

Emmett Brown[109]

This chapter reviews the project development process and its outputs. Based on this, conclusions are drawn and some suggestions are presented as an input for the project continuum.

7.1 Success Evaluation

To evaluate the success of the project, one should assess its compliance against the original objectives (section 1.4). As such these are the assessments which can be drawn:

1. Research the state of the art on methodologies and technologies regarding modeling and solving this type of problems - This report presented a rather complete overview on the different existent technologies including modeling languages and solver technologies but also supporting frameworks and platforms

2. Define a modeling language (. . .) - A language was defined which can bind against Dynamics AX data its applicability as been demonstrated using some common problems in the industry like the PetroChem or the Boeing models;

3. Identify 1-2 optimization problems in the Dynamics AX platform - Three problems have been identified in the Dynamics AX product: warehouse, traveling salesman and production scheduling;

4. Design models, using the defined language, to solve the problems identified under 3 - Models have been designed to solve two of three identified problems;

5. Design and implement the *ORML Interpreter* - The ORML Interpreter has been successfully implemented and totally supports the ORML language definition being its developed supported by industry recognized methodologies and tools;

6. Provide Integration with Microsoft Dynamics AX database, both in terms of data and meta data - The ORML language provides mechanisms to bind against Dynamics AX data.

Looking at the secondary objectives, they could be also considered as achieved.
As such, and given the results that have been presented (section 6), one can conclude that the project met the requirements it was proposed to, with the implementation of every component suggested in chapter 2.

7.2 Conclusions

The proposed ORML language and its interpreter present a solution to the problem that this project aimed to tackle 1.4). Looking at the results of a small survey (Appendix A) taken among some of the persons who assisted some of the project's in-site presentations, when asked about the interest of such a product coupled with the Dynamics AX, its potential becomes clear.
The *ORML Interpreter*, was successfully developed, implementing every feature of the language specification and supplying a set of services that can be used by third party applications such as the *ORML IDE* or the, also developed, console application.
It's worth mentioning that the developed interpreter includes some features that are not common in this type of languages like the semantic analysis. This greatly improves the modeling experience by giving early and better feedback to the user in regard to eventual errors.
The Interpreter is capable of reading data from and writing results back to Dynamics AX specific sources (tables, views or query-like classes). These data sources, especially the query-like classes have proven to be flexible enough to allow a great deal of usages by allowing custom user code. Through their usage, it is possible to implement very specific code and, although it was never tested, a similar syntax could be used to introduce heuristics in the search process (e.g. use a query-like class as an evaluation function).
It was also demonstrated[1], how the Microsoft Solver Foundation framework could be used, in an easy way, from the Dynamics AX system, not only by Microsoft developers and partners, but also by costumers which could now empowered to do specific modeling.
Finally, the export functionalities to both MathML and OMML proposed an easy way of representing the models to anyone who is not familiar with mathematical programming or with the ORML language specificities.
Some software engineering best practices were also developed and put in practice and,

[1] Although this subject isn't taken very deeply in this report due to the reasons stated in section 3.3.2

regarding this, the reader was presented with some notions on how the product was developed and what could be applied in similar projects to improve their outcome.

7.3 Originalities

The language this project presents has been developed to the image of many other proven languages that exist in the market for quite some time now. Innovation in this specific area, and for a project of this time span, would be hard to accomplish. Its underlying concept and some of its specificity is, however, something new. By being natively integrated in an ERP system, the ORML language, its interpreter and IDE, present an easy way to access, edit or create the models that are being used in the system. This comes in opposition to the hard to locate models that exist in other ERP systems and makes it much more usable by both partners and final costumers.
The developed solution also presents some services, like syntax highlighting or auto completion, that aid the user in the model edition and that aren't always present in other languages in the market.
Given the Dynamics AX specificities on accessing its database data and the impossibility of using the normal mechanisms which are presented by other frameworks in the market, the presented data-binding mechanisms also introduce something new in the system, which demonstrates how the concept could be put into production while providing a starting point for gathering usage experience and feedback.
Finally, the export functionalities, give an extra value to the project by providing a universal view of the models which makes them accessible to anyone who might not be familiar with the specificities of the ORML language.

7.4 Limitations

Given the time span of this project, there are, still some limitations in the developed product. This has mainly to do with language extensions to support new types of operations and even features like functions. This aspect could greatly restrain the number of models that could be modeled in its current version. If one considers that this project aimed at developing a prototype to demonstrate a concept, this is, however, not a big limitation and could easily be overcome since the language and its interpreter have been defined, in such a way, that it would be easy to extend them in feature versions. This was ensured, for instance, through the usage of well defined model structures in the language definition and by the usage of appropriate design patterns and tools in the interpreter's structure and components.
Some other key aspects for a new language are not completely present in the developed solution. Some of the most noticeable ones are performance and infeasibility tracing.

These weren't, however, identified as being primordial in a project of this nature and, their disregard has been a decision that had to be taken given the time span of the project.

7.5 Project Continuum

The development of a mathematical modeling language is a complex process and its evolution is coupled, among other things, with the evolution of solver and modeling technologies. For these reasons, and considering the time span of this project, the developed solution is, therefore, subject of many ideas for future improvements. This section will present some of them.

Extended language constructs

The constructs that were defined in the ORML language, at this stage, only cover basic functionalities and operations. This definition was prepared taking into account the prototyping purposes of this project and therefore aimed at a relatively narrow family of optimization problems.
Extended language constructs including, but not only, the ability of defining functions, and other operation types should be added in a production quality product.

Support for other types of optimization problems

The ORML language, and its interpreter, are not oriented towards a specific type of optimization problems. Enough freedom has remained in the language definition to support other types of optimization problems keeping a common construct grammar - this can be easily observed in the linear and constraint programming models that were presented and that are both partially addressed by the developed prototype.
However, only two types of such problems are addressed in the current solution. In a market-worthy product, other types of problems should be supported by both the language and the interpreter.

Debug System

A debug system could provide a simple way of understanding the logic behind the model, adding the possibility of knowing which constraints are making the model infeasible or even some additional information like shadow prices. The debug system should also allow the runtime stoppage of the solver to verify the algorithms it is using, as well as, the current variable values.

Bibliography

[1] MSN Encarta. Isaac newton, 2008. [Online; accessed 12-June-2008] http://encarta.msn.com/encyclopedia_761573959/Isaac_Newton.html.

[2] Microsoft. Microsoft corporation, 2008. [Online; accessed 9-June-2008] http://www.microsoft.com/en/us/default.aspx.

[3] Microsoft Corporation. *Microsoft CP/M BASIC: Reference Book*. Microsoft Press, 1977.

[4] David A. Lien. Ms-dos, 1986. ISBN: 0-932-76041-4.

[5] Intel. Intel, 2008. [Online; accessed 9-June-2008] http://www.intel.com/.

[6] Britannica. Micro instrumentation and telemetry systems, April 2008. [Online; accessed June-9-2008] http://www.britannica.com/EBchecked/topic/725876/Micro-Instrumentation-Telemetry-Systems.

[7] Linus Torvalds and David Diamond. *Just for Fun: The Story of an Accidental Revolutionary*. HarperBusiness, 2001. ISBN: 0-0666-2072-4.

[8] Glyn Moody. *Rebel Code: Linux and the Open Source Revolution*. Allen Lane, 2001. ISBN: 0-7139-9520-3.

[9] Softpanorama. Xenix – microsoft short-lived love affair with unix, February 2008. [Online; accessed 9-June-2008] http://www.softpanorama.org/People/Torvalds/Finland_period/xenix_microsoft_shortlived_love_affair_with_unix.shtml.

[10] Van Wolverton. *Running MS-DOS: Covers Through Version 6.0*. Microsoft Press, 2008. ISBN: 1-556-15542-5.

[11] Microsoft. Windows, 2008. [Online; accessed 9-June-2008] http://www.microsoft.com/WINDOWS/.

[12] Microsoft Corporation. Annual report 2007. Technical report, Microsoft, 2008.

[13] Forbes. Special report - the global 2000, March 2007. [Online; accessed 6-July-2008] http://www.forbes.com/lists/2007/18/biz_07forbes2000_The-Global-2000_Rank.html.

[14] Microsoft. Microsoft dynamics is familiar to your people, June 2006. [Online; accessed 16-June-2008] http://www.microsoft.com/dynamics/product/familiartoyourpeople.mspx.

[15] wcigroup. Does your it strategy maximise the potential of your existing investments? [Online; accessed 9-June-2008] http://www.wcigroup.com/Utilities/Microsoft+Stack/.

[16] Microsoft. Microsoft development center copenhagen - who we are, 2008. [Online; accessed 9-June-2008] http://www.microsoft.com/danmark/om/mdcc/introduction.mspx.

[17] Microsoft. Microsoft development center copenhagen - products, 2008. [Online; accessed 9-June-2008] http://www.microsoft.com/danmark/om/mdcc/products.mspx.

[18] Luis X. B. Mourão and David Weiner. *Dynamics AX - A guide to Microsoft Axapta*. Microsoft Press, 2005. ISBN: 1-590-059489-4.

[19] Internet News. Microsoft to buy navision for $1.3 billion, May 2002. [Online; accessed 9-June-2008] http://www.internetnews.com/bus-news/article.php/1038841.

[20] Hamdy A. Taha. *Operations Research: An Introduction*. Pearson/Prentice Hall, 2006. ISBN: 0-131-88923-0.

[21] Hamdy A. Taha. *Operations Research: An Introduction*. Prentice Hall, 1997. ISBN: 0-132-72915-6.

[22] What about he ”o” in o.r.?, 2007. [Online; accessed 5-July-2008] http://www.lionhrtpub.com/orms/orms-12-07/frqed.html.

[23] Anthony Hyman. *Charles Babbage - Pioneer of the computer*. Princeton University Press, 1982.

[24] Charles Babbage. *On the Economy of Machinery and Manufactures*. Charles Knight, Pall Mall East, 1832.

[25] Victorianweb. The penny post, October 2006. [Online; accessed 6-March-2008] http://www.victorianweb.org/history/pennypos.html.

[26] Educause. The erp dilemma: ”plain vanilla” versus customer satisfaction, 2003. [Online; accessed 9-June-2008] http://net.educause.edu/ir/library/pdf/EQM0327.pdf.

[27] MSN Encarta. Albert einstein, 2008. [Online; accessed 12-June-2008] http://encarta.msn.com/encyclopedia_761562147/Albert_Einstein.html.

[28] Gazeta Mercantil. Microsoft lança programa financeiro, June 2008. [Online; accessed 16-June-2008] http://www.investnews.net/integraNoticia.aspx?Param=606%2C0%2C+%2C1880772%2CUIOU.

[29] Rui Barbosa Martins. Base system architecture and constraints for a concurrent managed constraint solver at microsoft. Technical report, Copenhagen, 2006.

[30] Nuno José Pinto Bessa de Melo Cerqueira. Search and propagation engine for a concurrent managed constraint solver at microsoft. Technical report, Copenhagen, 2006.

[31] C.E. Miller, A.W. Tucker, and R.A. Zemlin. Integer programming formulation of travelling salesman problems. *ACM 7*, pages 326–329, 1960.

[32] Alexandre Medeiros Rodrigues. *Estratégias de picking na armazenagem*. Instituto COPPEAD de Administração, Centro de Estudos em Logística, Rio de Janeiro, 1999.

[33] Paul Schönsleben. *Integral Logistics Management: Planning & Control of Comprehensive Supply Chains*. CRC Press, 2003. ISBN: 1-574-44355-0.

[34] Steve McConnell. *Software Project Survival Guide*. Microsoft Press, 1997. ISBN: 1-57231-621-7.

[35] Ian Sommerville. *Software Engineering*. Addison-Wesley, 8^{th} edition, 2006. ISBN: 0-321-31379-8.

[36] Josef Kallrath. *Modeling Languages in Mathematical Optimization*. Kluwer Academic Publishers, Norwell, Masschusetts, 2005. ISBN: 1-4020-7547-2.

[37] Stefan Voß and David L. Woodruff. *Introduction to Computational Optimization Models for Production Planning in a Supply Chain*. Springer, Berlin, Heidelberg, 2005. ISBN: 3-540-00023-2.

[38] Joaquim Rendeiro. Operations research modeling environment for an erp system. Technical report, FEUP, Porto, 2008.

[39] Chris Sells and Ian Griffiths. *Programming WPF*. O'Reilly Media, Gravenstein Highway North, Sebastopol, CA, 2005. ISBN: 0-596-10113-9.

[40] Waitley. Meet dr. denis waitley. [Online; accessed 12-June-2008] http://www.waitley.com/Meet%20Denis%20Waitley.html.

[41] Hans J. Skovgaard and Arthur Greef. *Inside Microsoft Dynamics AX 4.0*. Microsoft Press, Redmond, Washington, 2006. ISBN: 978-0-7356-2257-4.

[42] MSDN. X++ programming guide, 2008. [Online; accessed 7-July-2008] http://msdn.microsoft.com/en-us/library/aa867122.aspx.

[43] MSN Encarta. Linear programming, 2008. [Online; accessed 11-June-2008] http://encarta.msn.com/encyclopedia_761560547/Linear_Programming.html.

[44] German Vitaliy. Solving linear constraints over real, December 2006. [Online; accessed 11-June-2008] http://www.risc.uni-linz.ac.at/projects/intas/Timisoara/Presentations/German/Grman.pdf.

[45] Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of Constraint Programming*. Elsevier, 2006. ISBN: 0-444-52726-5.

[46] ILog Changing the rules of business. Ilog cplex, 2008. [Online; accessed 12-June-2008] http://www.ilog.com/products/cplex/.

[47] ILOG. Model development, 2008. [Online; accessed 3-July-2008] http://www.ilog.com/products/model_development/.

[48] Maximal USA. Mpl modeling system - introducing mpl for windows 4.2, 2002. [Online; accessed 9-June-2008] http://www.maximal-usa.com/mpl/.

[49] Maximal-USA. Mpl manual - table of contents, 2003. [Online; accessed 14-June-2008] http://www.maximal-usa.com/mplman/mplwtoc.html.

[50] AIMMS. Aimms, world leader in optimization modeling, 2008. [Online; accessed 14-June-2008] http://www.aimms.com/aimms/index.cgi.

[51] AIMMS. Aimms - the language reference, may 2008. [Online; accessed 14-June-2008] http://www.aimms.com/aimms/download/manuals/AIMMS3_LR.pdf.

[52] AMPL. What's ampl?, April 2008. [Online; accessed 14-June-2008] http://www.ampl.com/.

[53] OptiRisk Systems. Ampl studio. [Online; accessed 17-June-2008] http://www.optirisk-systems.com/products_amplstudio.asp.

[54] Bezalel Gavish and Stephen C. Graves. The travelling salesman problem and related problems. Technical report, Massachusetts Institute of Technology, Boston, 1978.

[55] Peter Brucker. *Scheduling Algorithms*. Spring, Berlin, Heidelberg, 2007. ISBN: 978-3-540-69515-8.

[56] W3C Math Home. What is mathml?, May 2008. [Online; accessed 9-June-2008] http://www.w3.org/Math/.

[57] Microsoft. Microsoft math, 2008. [Online; accessed 16-June-2008] http://www.microsoft.com/math/default.mspx.

[58] Microsoft Corporation. Microsoft math sdk manual. 2006.

[59] Msdn. Introducing the office (2007) open xml file formats, 2008. [Online; accessed 9-June-2008] http://msdn.microsoft.com/en-us/library/aa338205.aspx.

[60] BNET. Pkware releases new zip file format specification, June 2008. [Online; accessed 9-June-2008] http://findarticles.com/p/articles/mi_m0EIN/is_2001_Dec_5/ai_80555082.

[61] The World Wide Web Consortium. Extensible markup language (xml). [Online; accessed 6-July-2008] http://www.w3.org/XML.

[62] Microsoft. Microsoft office online, 2008. [Online; accessed 16-June-2008] office.microsoft.com.

[63] MSDN. Welcome to the open xml format sdk 1.0, 2008. [Online; accessed 6-July-2008] http://msdn.microsoft.com/en-us/library/bb448854.aspx.

[64] Ecma International. Office open xml file format. *Ecma International*, pages 11–15.

[65] MSDN. About the open xml format sdk 1.0, 2008. [Online; accessed 6-July-2008] http://msdn.microsoft.com/en-us/library/bb456487.aspx.

[66] MSDN. .net framework, 2008. [Online; accessed 9-June-2008] http://msdn.microsoft.com/en-us/netframework/default.aspx.

[67] Java. Java, 2008. [Online; accessed 9-June-2008] http://www.java.com.

[68] Ken Arnold and James Gosling. *The Java Programming Language*. Addison-Wesley, 1998. ISBN: 0-201-31006-6.

[69] Andrew Troelsen. *Pro C# 2008 and the .NET 3.5 Platform*. Apress, 2007. ISBN: 978-1-59059-884-9.

[70] Jesse Liberty. *Programming C#: Building .NET Applications with C#*. O'Reilly, 2005. ISBN: 978-0-596-00699-0.

[71] Robert Pickering. *Foundation of F#*. Apress, 2007. ISBN: 978-1-59059-757-6.

[72] Don Syme, Adam Granicz, and Antonio Cisternino. *Expert F#*. Apress, 2007. ISBN: 978-1-59059-850-4.

[73] Kenneth C. Louden. *Compiler Construction: Principles and Practice*. PWS Publishing Company, Boston, 1997. ISBN: 978-3-540-69515-8.

[74] Wikipedia. Compiler compiler, June 2008. [Online; accessed 28-June-2008] http://en.wikipedia.org/wiki/Compiler_compiler.

[75] What is antlr?, 2008. [Online; accessed 5-July-2008] http://www.antlr.org/.

[76] Doug Brown, John Levine, and Tony Mason. *lex & yacc*. O'Reilly, 1992. ISBN: 978-1-56592-000-7.

[77] Microsoft Research. Phoenix academic program, 2007. [Online; accessed 11-June-2008] http://research.microsoft.com/Phoenix/.

[78] Lars Powers and Mike Snell. *Microsoft Visual Studio 2008 Unleashed*. Pearson Educationy, Boston, 2008. ISBN: 978-0-672-32972-2.

[79] MSN Encarta. Leonardo da vinci, 2008. [Online; accessed 12-June-2008] http://encarta.msn.com/encyclopedia_761561520/Leonardo_da_Vinci.html.

[80] Peter Naur. Revised report on the algorithmic language algol 60. Technical report, ACM, 1962.

[81] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers - Principles, Techniques and Tools*. Addison-Wesley Publishing Company, Stanford, California, 1986. ISBN: 0-201-10194-7.

[82] MPL. Mpl for windows manual, 2003. [Online; accessed 11-April-2008] http://www.maximal-usa.com/mplman/mplw6/mpw06000.html.

[83] Starwars. Yoda, 2008. [Online; accessed 14-June-2008] http://www.starwars.com/databank/character/yoda/.

[84] Rational Unified Process. Best practices for software development teams, November 2001. [Online; accessed 17-June-2008] http://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf.

[85] Philippe Kruchten. Architectural blueprints - the ”4+1” view model of software architecture. Technical report, Rational Software Corp., November 1995.

[86] Veer Muchandi. Applying 4+1 view architecture with uml 2 - white paper. Technical report, First FCG - Software Services, 2007.

[87] Wikipedia. Interpreter (computing), June 2008. [Online; accessed 16-June-2008] http://en.wikipedia.org/wiki/Interpreter_%28computing%29.

[88] James W. Cooper. *C# Design Patterns - A tutorial*. Addison-Wesley, 2003. ISBN: 0-201-84453-2.

[89] Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 2001. ISBN: 0-201-63361-2.

[90] Judith Bishop. *C# 3.0 Design Patterns*. O'Reilly, 2007. ISBN: 978-0-596-52773-0.

[91] Regular-Expressions. Welcome to regular-expressions.info, August 2007. [Online; accessed 25-June-2008] http://www.regular-expressions.info/.

[92] Msdn Visual C# Developer Center. The c# language, 2008. [Online; accessed 25-June-2008] http://msdn.microsoft.com/en-us/vcsharp/aa336809.aspx.

[93] IEEE Standards Board. Ieee standard glossary of software engineering terminology. Technical report, IEEE, New York, 1990.

[94] James W. Newkirk and Alexei A. Vorontsov. *Test-Driven Development in Microsoft .NET*. Microsoft Press, 2004. ISBN: 978-0-735-61948-7.

[95] Agile Data. Introduction to test driven design (tdd), March 2007. [Online; accessed 14-June-2008] http://www.agiledata.org/essays/tdd.html.

[96] Kent Beck. *Test-Driven Development - By Example*. Addison-Wesley, 2002. ISBN: 0-321-14653-0.

[97] Msdn. Guidelines for test-driven development, May 2006. [Online; accessed 14-June-2008] http://msdn.microsoft.com/en-us/library/aa730844(VS.80).aspx.

[98] AutomatedQA. Testcomplete - regression testing, 2008. [Online; accessed 14-June-2008] http://automatedqa.com/products/testcomplete/tc_regression_testing.asp.

[99] Better Exaplined. A visual guide to version control, 2007. [Online; accessed 14-June-2008] http://betterexplained.com/articles/a-visual-guide-to-version-control/.

[100] Perforce. Perforce software, 2008. [Online; accessed 6-July-2008] http://www.perforce.com/.

[101] Microsoft. *Team Development with Visual Studio Team Foundation Server*. Microsoft Press.

[102] Joel Spolsky. *More Joel on Software*. Apress, 2008. ISBN: 978-1-4302-0987-4.

[103] Joel Spolsky. Daily builds are your friend, January 2001. [Online; accessed 6-July-2008] http://www.joelonsoftware.com/articles/fog0000000023.html.

[104] Steve McConnell. Best practices: Daily build and smoke test. *IEEE Software*, 4, July 1996.

[105] Leena Singh and Leonard Drucker. *Advanced Verification Techniques*. Springer, 2004. ISBN: 978-1-4020-7672-5.

[106] MSN Encarta. Fernando antonio nogueira pessoa, June 2008. [Online; accessed 14-June-2008] http://encarta.msn.com/encyclopedia_762506694/Pessoa_Fernando_Antonio_Nogueira.html.

[107] Elfriede Dustin. *Effective Software Testing: 50 Specific Ways to Improve Your Testing*. Addison-Wesley, Boston, 2003. ISBN: 0-201-79429-2.

[108] Why you only need to test with 5 users, march 2000. [Online; accessed 5-July-2008] http://www.useit.com/alertbox/20000319.html.

[109] Wikipedia. Emmett brown, June 2008. [Online; accessed 15-June-2008] http://en.wikipedia.org/wiki/Doc_Brown.

BIBLIOGRAPHY

Appendix A

Survey Questions

Questions

***Question 1* - Career Path? (Choose one)**

- Developer
- Tester
- Program Manager

***Question 2* - What was your field of study? (Choose one)**

- Computer Science
- Mathematics
- Engineering
- Other

***Question 3* - What program did you take? (Choose one)**

- Undergratuate
- Master
- PhD
- Other

***Question 4* - Did you, during your education, have had any contact with the Operations Research subject? (Choose one)**

- Yes (go to question 5)
- No (go to question 6)

Question 5 **- Have you ever experimented an optimization modeling language? (e.g. OML, MPL, AIMMS) (Choose one)**

- Yes (go to question 5b)
- No (go to question 6)

Question 5b **- Which ones? (e.g. OML, MPL, AIMMS) (Choose the ones that apply)**

- NL (OML)
- OPL
- AIMMS
- AMPL
- MPL
- GAMS
- Other

Question 5c **- What do you think that are the most important things in such languages? (Choose the ones that apply)**

- Functions Available
- Ease of writting
- Data-binding Mechanisms
- Good Documentation
- Solver Independence
- Other

Question 5d **- What do you think that are the most important in an IDE for such languages? (Choose the ones that apply)**

- Syntax Highlight
- Good Error Feedback
- Autocompletion Mechanisms
- Other

Question 6 **- Do you think an Optimization language within the Dynamics AX product for problem modeling would be an advantage in regard to the methods used now? (Choose one)**

- Yes
- No

Question 7 - **Which Persona, in the MBS customer model, do you think that could benefit the most from such facility? (Free text)**

Results

This survey served the purpose of getting some feedback on the different points of the project and provide some new information and opinios. It was therefore, narrowed to a small sample of 10 persons.

Question 1

These are question's 1 results. Looking at them, it is clear the the majority of the answers were taken from developers. The fact that there aren't any PM answers has to do with the fact that it wasn't possible to send the survey to one.

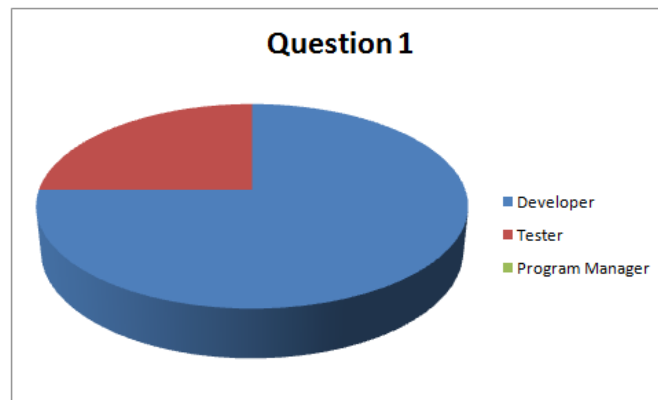

Figure A.1: Question 1 answers

Question 2

Question 2 showed that the majority of the interviewed had a computer science background.

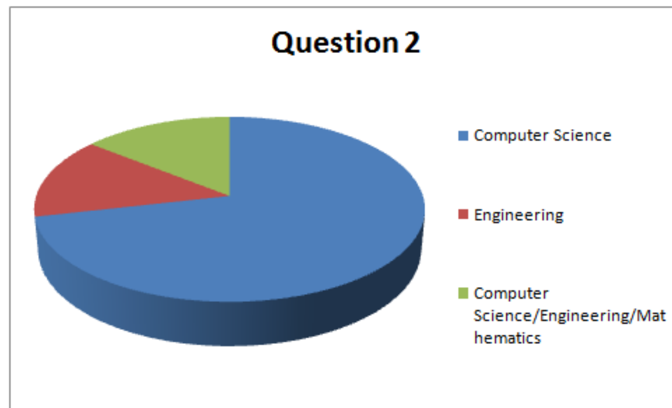

Figure A.2: Question 2 answers

Question 3

Question 3 showed that we're interviewing, mostly, persons with a masters degree which is important information considering the next following questions.

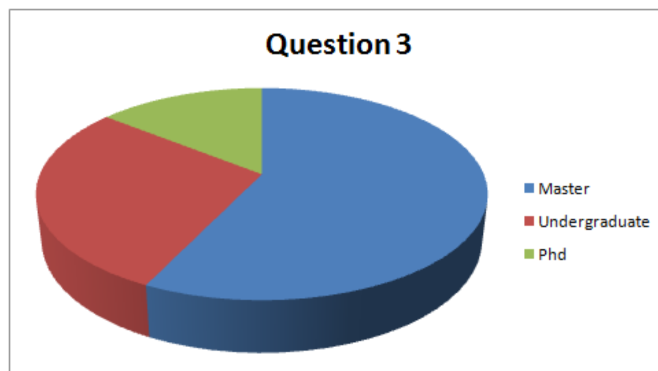

Figure A.3: Question 3 answers

Question 4

Question 4 showed that most of the persons have had some contact with the operations research topic during their education. These kind of persons should be able to use the ORML language with little effort.

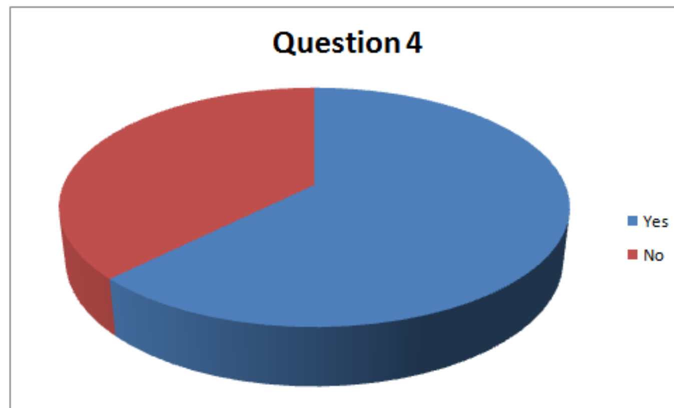

Figure A.4: Question 4 answers

Question 5

Question showed that, although many people have had contact with operations research, this was probably a more theorical contact with the algorithms and processes and not with the environments. Thus, the adaptation the the ORML language can be harder for the ones who haven't tried any.

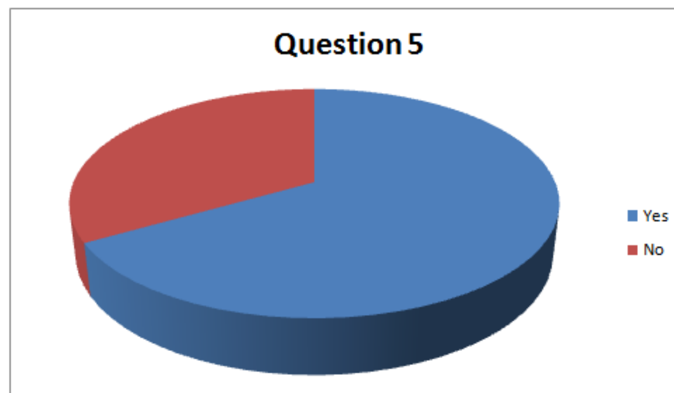

Figure A.5: Question 5 answers

Question 5b

From the ones that have tried one modeling language, NL is clearly the most used one, mainly because this is the language that the Microsoft Solver Foundation team promoted. Other than this, the OPL language was also apointed.

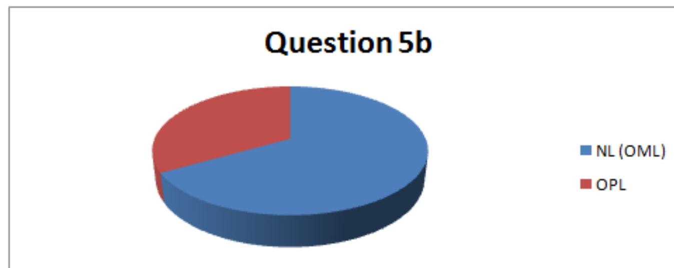

Figure A.6: Question 5b answers

Question 5c

This and the following questions prooved that users want all sort of features that they are already used to and take for granted. The introduction of a new technology that doesn't address all this issues will probably be frustrated.

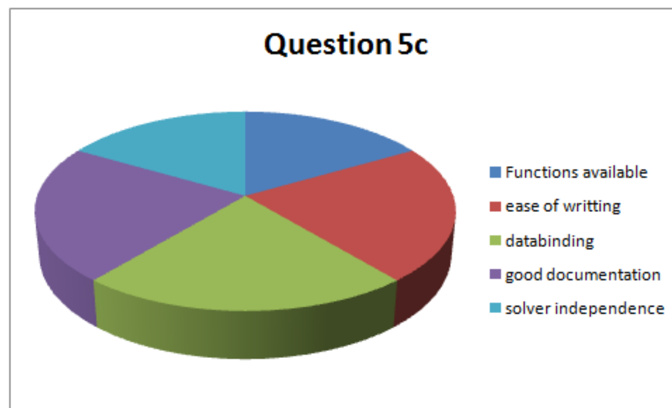

Figure A.7: Question 5c answers

Question 5d

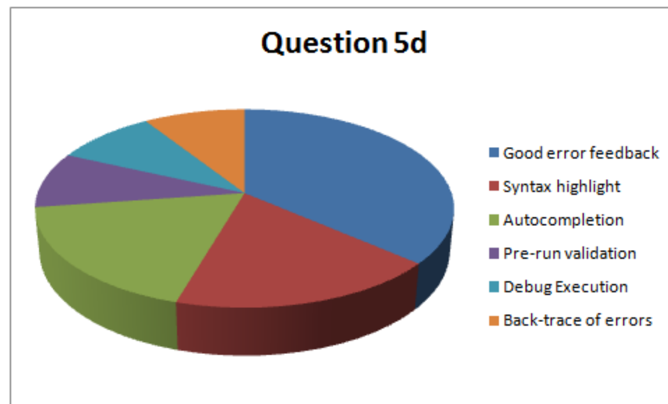

Figure A.8: Question 5d answers

Question 6

Question 6 was, in a certain way, a surprise. It showed that everyone, from the enquired, agreed that a language like ORML with the specific details, including the data-binding mechanisms, to the Dynamics AX product, would be an advantage for the it.

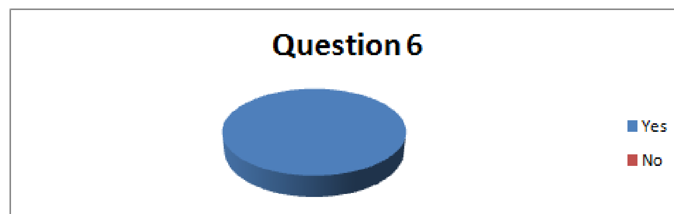

Figure A.9: Question 6 answers

Question 7

Although the interest was clear, there wasn't a clear definition on who could use this feature. On the other hand, this may show that the usage is transversal and may reach all sorts of users.

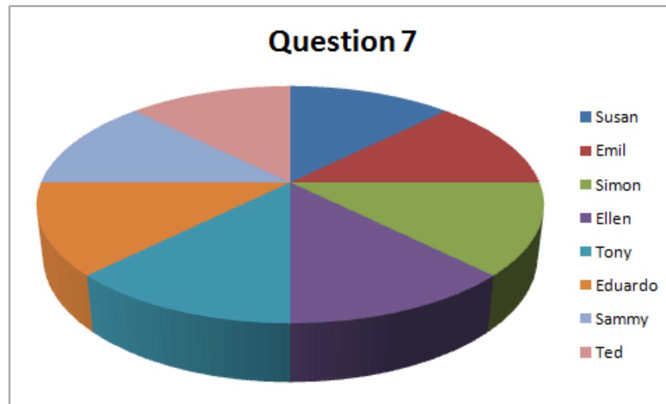

Figure A.10: Question 7 answers

Appendix B

Project Requirements

ORML Interpreter Features List

Perform Lexical Analysis (*Int-UC01*)

Purpose Identify the different types of tokens in a source code text.

Inputs The source code text.

Processing Use regular expressions to identify the different types of tokens and supply them to the Syntax Analyzer.

Outputs The meaningful types of tokens identified.

Priority High

Perform Syntax Analysis and AST construcion (*Int-UC02*)

Purpose Match the identified tokens against the specific language grammar rules and produce an AST.

Inputs Types of tokens, identified one by one, or the termination character when there is no more token to be processed.

Processing Match identified token sequences against the specific grammar rules in a bottom-up approach.

Outputs An AST that fully represents the code processed.

Priority High

Perform Semantic Analysis (*Int-UC03*)

Purpose Add semantic information to the parse tree and builds the symbol table. This phase performs semantic checks such as type checking (checking for type errors), or object binding (associating variable and function references with their definitions), or definite assignment (requiring all local variables to be initialized before use), rejecting incorrect programs or issuing warnings.

Inputs The AST that was previously build by the parser.

Processing Create the semantic table to represent the different symbols in the AST, as well as, doing the required checking and signaling the errors to the specific handlers.

Outputs A symbols table and, if it is the case, the identified warnings/errors.

Priority High

Interpret Model (*Int-UC04*)

Purpose Translate the AST and symbols table into instructions and actually process them in order to obtain the results for the model.

Inputs The previously built AST and symbols table.

Processing Navigate through the AST using the symbols table and translating them to instructions.

Outputs The model results or, otherwise, a list of warnings/errors.

Priority High

Color Highlight service (*Int-UC05*)

Purpose Identify the different types of tokens within a text to be processed.

Inputs String with the source code to be processed.

Processing Process the input text through the scanner to identify the different types of tokens.

Outputs List of identified tokens together with their classification.

Priority Medium

Autocomplete service (*Int-UC06*)

Purpose Based on some current input text supplied by the user through the IDE identify possible future sequences of characters. It speeds up software development by reducing the amount of name memorization needed and keyboard input required. It also allows for less reference to external documentation as interactive documentation on many symbols (i.e. variables and functions).

Inputs Current user's input.

Processing Based on the input, using the possible tokens to be formed and possible grammar rules, identify future code possibilities.

Outputs A list of code possibilities to predict a word or phrase that the user wants to type in without actually having to type it in completely.

Priority Low

Export model to MathML (*Int-UC07*)

Purpose Export an ORML model to the MathML document format.

Inputs The ORML model text.

Processing Convert the ORML model to the MathML format section by section and then to a single file.

Outputs An MathML document that corresponds to the original ORML model.

Priority Medium

Export model to OMML (*Int-UC08*)

Purpose Export an ORML model to the OMML document format.

Inputs The ORML model text.

Processing Convert the ORML model to the MathML format section by section and then convert each of the sections to OMML using the correspondent office's XSLT. If the desired output is a file, use the OpenXML library to write the results.

Outputs An OMML document that corresponds to the original ORML model.

Priority Medium

Appendix C

Project Schedule

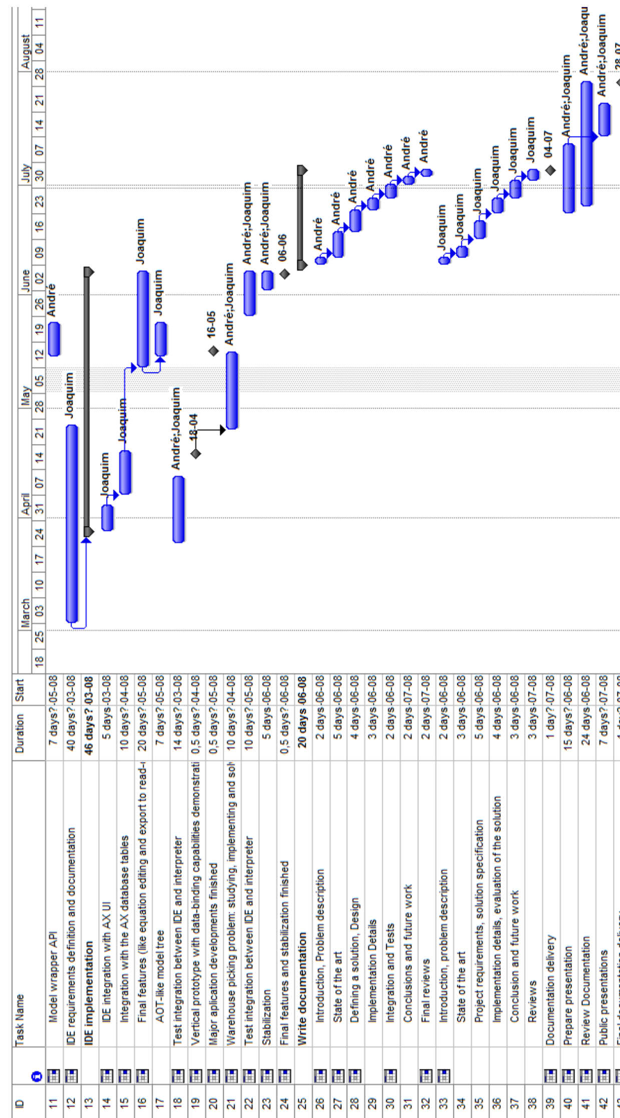

Figure C.1: Project Schedule

Project Schedule

Appendix D

Microsoft Math Supported Formats

Format	Description
Formula	This format is not very human-readable, but best for machine processing and unit test. The meaning of an expression is precise, there is no need for parentheses, and it's easy to parse. While other serializers may contain little tweaks to make the UI component work as expected, this format doesn't change often because no real UI uses it. Example: `Sum[1,Divide[2,x]]`
InvariantFormula	Similar to Formula, except the variables are in the format of Var(0), Var(1), ... instead of in their names. Used in serialization only (CasContext.FormatOptions) but not in parsing (CasContext.ParsingOptions). Example: `Sum[1,Divide[2,Var(0)]]`
FormulaWithoutAggregate	Similar to Formula, except the aggregates (grouping parentheses) are silently removed during serialization. The primary usage for this format is in internal test cases. When used in parsing, it's identical to Formula. Example: `Sum[1,Divide[2,x]]`
Linear	This format is the most human-readable one. Can be used in command line applications or application that accept and displays linear syntax. Also, the RichEdit wrapper returns linear syntax and it should be parsed with Linear format type. Example: `1+2/x`
LinearInput	This format is similar to linear format, but during serialization it uses ASCII characters only (except for variable named specified by the user) and can roundtrip back to the linear parser. When used in parsing it's identical to Linear.

MathML	The standard MathML format. Note that the math engine can only parse a subset of the syntax in the MathML specification. Example: <math><mn>1</mn><mo>+</mo><mfrac><mn>2</mn> <mi>x</mi></mfrac></math>
MathMLNoWrapper	Same as MathML, except this format type won't generate the root element ¡math¿. Used in serialization only (CasContext.FormatOptions) but not in parsing (CasContext.ParsingOptions). Example: <mn>1</mn><mo>+</mo><mfrac><mn>2</mn><mi>x</mi> </mfrac>
InlineMathRichEdit	This format type is for the RichEdit wrapper to render math in RichEdit. It's same with MathRichEdit format with one exception: fractions are rendered horizontally instead of vertically. This is a better choice if the display area is confined with a small height. Used in serialization only (CasContext.FormatOptions) but not in parsing (CasContext.ParsingOptions). Example: 1␣+␣x/(2␣a)
Binary	The most compact format. Uses binary data. Like Format, it guarantees roundtrip between parsing and serialization.
InvariantBinary	Similar to Binary, except variableIds are used instead of variable name.
Base64	This is the base-64 encoded Binary format.
InvariantBase64	This is the base-64 encoded InvariantBinary format.

Table D.1: Microsoft Math Engine Supported Formats

Appendix E

Language Definition

The lexical grammar will be presented using regular expressions. By using them in a scanner generator tool, it is possible to generate a scanner that will successfully identify the different grammar productions in the language. For the syntactic grammar, grammar productions will be used. Each grammar production defines a non-terminal symbol and the possible expansions of that non-terminal symbol into sequences of non-terminal and terminal symbols. In grammar productions, non-terminal symbols are shown in italic type, and terminal symbols are shown in a fixed-width font.
The first terminal line of a grammar production is the name of the non symbol being defined, followed by a colon. Each successive indented line contains a possible expansion of the non-terminal given as a sequence of non-terminal or terminal symbols.
When there is more than one possible expansion of a non-terminal symbol, the alternatives are listed on separate lines. For example, the production:

```
1 statement-list:
2 statement statement-list
3     statement
```

defines a statement-list that either consists of a statement or a statement followed by a statement-list.

Lexical Grammar

Line Terminators

A line terminator can be a Line feed character (U+000A); a Carriage return character (U+000D); a Carriage return character (U+000D), followed by line feed character (U+000A); a Next line character (U+0085); a Form feed character (U+000C); Line separator character (U+2028) or a Paragraph separator character (U+2029). The equivalent regular expression is:

```
1 [\U000A\U000D\U0085\U000C\U2028\U2029]|\U000D\U000A
```

White Spaces

A white space is defined as a space character, a horizontal tab character, a vertical tab character or a form feed character. The corresponding regular expression:

```
1 [ \t\f\v]
```

Comments

For the two types of comments available (single-line comments and delimited comments), the regular expression translates into:

```
1  CmntStart    \/\*
2  CmntEnd      \*\/
3  ABStar       [^\*\n]*
4
5  \/\/.*
6  {CmntStart}{ABStar}\**{CmntEnd}
7  {CmntStart}{ABStar}\**
8  <COMMENT>\n
9  <COMMENT>{ABStar}\**
10 <COMMENT>{ABStar}\**{CmntEnd}
```

Tokens

For the different types of tokens, these are the regular expressions:

Identifiers

[language=none]

```
1 letter    [A-Za-z]
2 digit     [0-9]
3 (_|{letter})*{letter}(_|{letter}|{digit})*
```

Keywords

ORML's keyword can be expressed by the following regular expressions:

```
1  Integer
2  Real
3  String
4  Model
5  Variables
6  Constraints
7  Inputs
8  Indexes
9  Outputs
10 Tables
11 Views
12 Classes
13 in
14 Infinity
15 where
16 Sum
```

```
17 Count
18 Max
19 Exists
20 Abs
21 Unequal
22 Minimize
23 Maximize
```

Literals

ORML's literals can be expressed by the following regular expressions:

```
1 digit      [0-9]
2 0|[1-9]{digit}*
3 ([1-9]{digit}*)\.
4 \.({digit}*[1-9])
5 [1-9]{digit}*\.({digit}*[1-9])
```

Operators and Punctuators

ORML's operators and punctuators can be described by the following regular expressions:

```
1 /* Punctuation tokens */
2 ;                        // Semi colon
3 ,                        // Comma
4 :                        // Colon
5 \.                       // Dot
6 \.\.                     // Double Dot
7 \[                       // Left bracket
8 \]                       // Right bracket
9 \(                       // Left Parentheses
10 \)                      // Right Parentheses
11 \{                      // Left Curly Brace
12 \}                      // Right Curly Brace
13 \|                      // Pipe
14 /* Arithmetic operator tokens */
15 \+                      // Operator Plus
16 -                       // Operator Minus
17 \*                      // Operator Multiplication
18 \/                      // Operator Division
19 \%                      // Operator Module
20
21 /* Compare operator tokens */
22 !=                      // Operator Not Equal
23 ==                      // Operator Equals
24 \>=                     // Operator Equal or Greater Than
25 \<=                     // Operator Equal or Less Than
26 \>                      // Operator Greater Than
27 \<                      // Operator Less Than
28 \^                      // Operator Power
29
30 /* Boolean operator tokens */
31 \&\&                    // Binary Operator And
32 \|\|                    // Binary Operator Or
33
34 /* Special operator tokens */
35 =                       // Operator Attribution
```

Grammar Rules

An ORML model's general structure can be described by the following rules:

```
1  ModelStatement:
2    Model { NamedStatementList }
3    Model ( ParameterDeclarationList ) { NamedStatementList }
4
5  NamedStatementList:
6    NamedStatement NamedStatementList
7    NamedStatement
8
9  NamedStatement:
10   IndexSectionStatement
11   InputSectionStatement
12   VariableSectionStatement
13   FunctionSectionStatement
14   ConstraintSectionStatement
15   OutputSectionStatement
16
17 ParameterDeclarationList:
18   ParameterDeclaration , ParameterDeclarationList
19   ParameterDeclaration
```

Indexes

An Index section can be described by the following grammar rules:

```
1  IndexSectionStatement:
2    Index : IndexDeclarationDecls
3
4  IndexDeclarationDecls:
5    IndexDeclaration ; IndexDeclarationDecls
6    IndexDeclaration ;
7
8  IndexDeclaration:
9    Identifier = AxaptaRecord // database index
10   Identifier = SetExpression // numeric and named indexes
```

An axapta record can be expressed in the following way:

```
1  AxaptaRecord:
2    Tables . Identifier . Identifier
3    Views . Identifier . Identifier
4    Classes . Identifier . Identifier
5    Classes . Identifier ( ExpressionList ) . Identifier
```

Concerning the set expressions, their syntax is defined as follows:

```
1  SetExpression:
2    { SequenceElementList }
3    [ Integer .. Integer ]
4    [ Integer , Integer ]
5
6  SequenceElementList:
7    SequenceElement , SequenceElementList
8    SequenceElement
9
10 SequenceElement:
```

```
11    Literal
12
13  Literal:
14    Integer
15    Real
16    String
```

Inputs

An input section may be described by the following grammar rules:

```
1   InputSectionStatement:
2     Input : InputDeclarationDecls
3
4   InputDeclarationDecls:
5     InputDeclaration ; InputDeclarationDecls
6     InputDeclaration ;
7
8   InputDeclaration:
9     Identifier = AxaptaRecord
10    Identifier = AxaptaRecord [ AxaptaRecordColumnList ]
11    Identifier = SetExpression
12    Identifier = Expression
13    Identifier [ IndentifierList ]= SetExpression
14    Identifier [ IndentifierList ]= Expression
15
16  IdentifierList:
17    Identifier , IdentifierList
18    Identifier
19
20  AxaptaRecordColumnList:
21    AxaptaRecordColumn , AxaptaRecordColumnList
22    AxaptaRecordColumn
23
24  AxaptaRecordColumn:
25    Identifier == Identifier
26    Identifier
```

Variables

A Variables section may be described by the following grammar rules:

```
1   VariableSectionStatement:
2     Variables: VariableDeclarationDecls
3
4   VariableDeclarationDecls:
5     VariableDeclaration ; VariableDeclarationDecls
6     VariableDeclaration ;
7
8   VariableDeclaration: ValueType in Interval : VariableList
9
10  ValueType:
11    Integer
12    Float
13
14  Interval:
15    [Expression .. Expression]
```

```
16    [ Expression , Expression ]
```

Notice that the interval definition grammar for the variables is less strict than the one in the index declaration because, here, the user may want to use data already declared or compose expressions. The ORML grammar also supports multiple variable declarations in one line. The grammar that supports this can be described as follows:

```
1  VariableList:
2    Variable , VariableList
3    Variable
4
5  Variable:
6    Identifier
7    Identifier [ IdentifierList ]
```

Functions

A functions section may be described by the following grammar rules:

```
1  FunctionSectionStatement:
2    Goal : Expression ;
3
4  Goal:
5    Maximize
6    Minimize
```

Constraints

A constraints section may be described by the following grammar rules:

```
1  ConstraintSectionStatement:
2    Constraints : ConstraintDecls
3
4  ConstraintDeclarationDecls:
5    ConstraintDeclaration ; ConstraintDeclarationDecls
6    ConstraintDeclaration ;
7
8  ConstraintDeclaration:
9    BooleanExpression
10   BooleanExpression Where ( SetExpressionList )
11   BooleanExpression Where (SetExpressionList | BooleanExpression )
12
13 SetExpressionList:
14   SetExpression , SetExpressionList
15   SetExpression
16
17 SetExpression:
18   Identifier in Identifier
19   Identifier in Interval
20
21 Location:
22   Identifier
23   Identifier [ ExpressionList ]
24   AxaptaRecord
25
26 ExpressionList:
27   Expression , ExpressionList
```

```
28    Expression
```

Expressions

This section describes ORML expression's grammar rules:

Calls

An ORML's call may be defined by the following grammar rules:

```
1  Call:
2    Count ( Location )
3    Max ( Location )
```

Boolean Expression

An ORML's boolean expression may be defined by the following grammar rules:

```
1  BooleanExpression:
2    BooleanExpression && BooleanExpression
3    BooleanExpression || BooleanExpression
4    ComparisonExpression
```

Comparison Expression

An ORML's comparison expression may be defined by the following grammar rules:

```
1  ComparisonExpression:
2    Expression == Expression
3    Expression != Expression
4    Expression >= Expression
5    Expression <= Expression
6    Expression > Expression
7    Expression < Expression
```

Arithmetic and other expressions

Other types of ORML's expressions, including arithmetics, may be described by the following rules:

```
1  Expression:
2    Literal
3    Location
4    Call
5    ( Expression )
6    - Expression
7    Expression + Expression
8    Expression - Expression
9    Expression * Expression
10   Expression / Expression
11   Expression % Expression
12   Sum ( SetExpressionList ; Expression )
13   Sum ( SetExpressionList | BooleanExpression ; Expression )
```

Appendix F

Modeling Optimization Problems in Microsoft Dynamics AX

F.1 Traveling Salesman

```
1  Model(Integer NewPath) {
2    Indexes:
3      Cities = Tables.Cities.id;
4    Inputs:
5      Costs   = Tables.Paths.cost[from, to];
6      NumCities = Count(Cities);
7    Variables:
8      Integer in [0 .. 1] PathFromCityToCity [Cities, Cities];
9      Integer in [0 .. NumCities-1] SequenceCityVisited [Cities];
10   Minimize:
11     Sum(i in Cities ; Sum(j in Cities | i != j ; Costs[i, j] *
           PathFromCityToCity[i, j]));
12   Constraints:
13     Sum(j in Cities | i != j ; PathFromCityToCity[i, j]) == 1 where(i in Cities
           );
14
15     Sum(i in Cities | i != j ; PathFromCityToCity[i, j]) == 1 where(j in Cities
           );
16
17     (SequenceCityVisited[i] - SequenceCityVisited[j] + NumCities *
           PathFromCityToCity[i, j]) <= (NumCities - 1) where(j in Cities, i in
           Cities | j != i && i != 1 && j != 1);
18   Outputs:
19     Tables.CityPathSequence.ordernum[city, path=NewPath] = SequenceCityVisited;
20 }
```

Figure F.1: ORML Model - Traveling Salesman

F.2 Warehouse Picking Routes

```
1  Model(String RouteId, String StartLocation, String EndLocation) {
2      Inputs:
3          Costs = Tables.WMSLocationDistance.Distance[WMSLocationOrigin,
               WMSLocationDestination];
4      Indexes:
5          Locations = Classes.WMSGetRouteItemLocations(RouteId, StartLocation,
               EndLocation).location;
6      Inputs:
7          LocationNum = Count(Locations);
8      Variables:
9          Integer in [0 .. 1] PathFromLocationToLocation [Locations, Locations];
10         Integer in [1 .. LocationNum] SequenceLocationVisited [Locations];
11     Minimize:
12         Sum(i in Locations ;
13             Sum(j in Locations | i != j ; Costs[i, j] *
                   PathFromLocationToLocation[i, j]));
14     Constraints:
15         Sum(j in Locations | i != j && Exists(Costs[i,j]) ;
               PathFromLocationToLocation[i, j]) == 1
16             where(i in Locations | i != EndLocation);
17         Sum(i in Locations | i != j && i != EndLocation && Exists(Costs[i,j]) ;
                PathFromLocationToLocation[i, j]) == 1
18             where(j in Locations | j != StartLocation);
19         SequenceLocationVisited [i] - SequenceLocationVisited [j] + LocationNum
               * PathFromLocationToLocation[i, j] <= (LocationNum - 1)
20             where(i in Locations, j in Locations | j != i && i != StartLocation
                   && j != StartLocation);
21     Outputs:
22         Classes.WMSUpdateOrderTrans.sortcode[location, routeId=RouteId] =
               SequenceLocationVisited;
23 }
```

Figure F.2: ORML Model - Warehouse Picking Routes

Appendix G

Computer Specifications

G.1 Quad-core processor desktop

Number of processors : 4

Processor : Intel Xeon E5345 @ 2.33 GHz

Cache memory (per core pair) : 4 MB

Cache memory (global) : 8 MB L2

Memory : 4.0 GB

Front side bus speed : 1333 MHz

Architecture : 64 bits

Operating System : Windows Server 2008 x64 (64-bit version)

.NET Framework version : 3.5

G.2 Single-core processor laptop

Number of processors : 1

Processor : Intel Pentium M 750 @ 1.87 GHz

Cache memory : 2 MB L2

Memory : 2.0 GB

Front side bus speed : 533 MHz

Architecture : 32 bits

Operating System : Windows Vista x86 (32-bit version)

.NET Framework version : 3.5

Computer Specifications

Appendix H

ORML Models

H.1 Zebra

```
1  Model {
2    Variables:
3      Integer in [1,5]
4        English, Spanish, Japanese, Italian, Norwegian,
5        red, green, white, blue, yellow,
6        dog, snails, fox, horse, zebra,
7        painter, sculptor, diplomat, violinist,
8        doctor, tea, coffee, milk, juice, water;
9
10   Constraints:
11     Unequal(English, Spanish, Japanese, Italian, Norwegian);
12     Unequal(red, green, white, blue, yellow);
13     Unequal(dog, snails, fox, horse, zebra);
14     Unequal(painter, sculptor, diplomat, violinist, doctor);
15     Unequal(tea, coffee, milk, juice, water);
16
17     English == red;
18     Spanish == dog;
19     Japanese == painter;
20     Italian == tea;
21     Norwegian == 1;
22     green == coffee;
23     (green - white) == 1;
24     sculptor == snails;
25     diplomat == yellow;
26     milk == 3;
27     Abs(Norwegian - blue) == 1;
28     violinist == juice;
29     Abs(fox - doctor) == 1;
30     Abs(horse - diplomat) == 1;
31 }
```

Figure H.1: ORML Model - Zebra

Results

```
1  Microsoft (R) Orml Compiler version 0.2.1
2  Copyright (C) Microsoft Corporation 2008. All rights reserved.
3
4  Valid Syntax
5  Valid Semantics
6  ========Values=========
7  English -> 3
8  Spanish -> 4
9  Japanese -> 5
10 Italian -> 2
11 Norwegian -> 1
12 red -> 3
13 green -> 5
14 white -> 4
15 blue -> 2
16 yellow -> 1
17 dog -> 4
18 snails -> 3
19 fox -> 1
20 horse -> 2
21 zebra -> 5
22 painter -> 5
23 sculptor -> 3
24 diplomat -> 1
25 violinist -> 4
26 doctor -> 2
27 tea -> 2
28 coffee -> 5
29 milk -> 3
30 juice -> 4
31 water -> 1
```

H.2 Boeing

This model, due to its size, was decided to be omited from this report. If the reader wants to access it, please contact the author.

H.3 PetroChem

```
Model {
  Variables:
    Real SA, VZ;
  Minimize:
    20 * SA + 15 * VZ;
  Constraints:
    0.3 * SA + 0.4 * VZ >= 2000;
    0.4 * SA + 0.2 * VZ >= 1500;
    0.2 * SA + 0.3 * VZ >= 500;
    SA <= 9000;
    VZ <= 6000;
    SA >= 0;
    VZ >= 0;
}
```

Figure H.2: ORML Model - PetroChem

Results

```
Microsoft (R) Orml Compiler version 0.2.1
Copyright (C) Microsoft Corporation 2008. All rights reserved.

Valid Syntax
Valid Semantics
========Goals=========
Objective Value = [9250]
========Values=========
SA -> 200
VZ -> 350
```

H.4 WycoDoors

```
1  // wynder glass co. demo model for m2 presentation
2  Model {
3    Variables:
4      Real in [0, Infinity] x, y;
5    Maximize:
6      3 * x + 5 * y;
7    Constraints:
8      x <= 4;
9      2 * y <= 12;
10     3 * x + 2 * y <= 18;
11 }
```

Figure H.3: ORML Model - WycoDoors

Results

```
1  Microsoft (R) Orml Compiler version 0.1.4
2  Copyright (C) Microsoft Corporation 2008. All rights reserved.
3
4  Valid Syntax
5  Valid Semantics
6  ========Goals=========
7  Objective Value = [36]
8  ========Values=========
9  x -> 2
10 y -> 6
```

H.5 Traveling Salesman

```
1  Model {
2    Indexes:
3      Cities = Tables.Cities.id;
4    Inputs:
5      Costs   = Tables.Paths.cost[from, to];
6      NumCities = Count(Cities);
7      NewPath = 3;
8    Variables:
9      Integer in [0 .. 1] PathFromCityToCity [Cities, Cities];
10     Integer in [0 .. NumCities-1] SequenceCityVisited [Cities];
11   Minimize:
12     Sum(i in Cities ; Sum(j in Cities | i != j ; Costs[i, j] *
           PathFromCityToCity[i, j]));
13   Constraints:
14     Sum(j in Cities | i != j ; PathFromCityToCity[i, j]) == 1 where(i in Cities
           );
15
16     Sum(i in Cities | i != j ; PathFromCityToCity[i, j]) == 1 where(j in Cities
           );
17
18     (SequenceCityVisited[i] - SequenceCityVisited[j] + NumCities *
           PathFromCityToCity[i, j]) <= (NumCities - 1) where(j in Cities, i in
           Cities | j != i && i != 1 && j != 1);
19   Outputs:
20     Tables.CityPathSequence.ordernum[city, path=NewPath] = SequenceCityVisited;
21 }
```

Figure H.4: ORML Model - Traveling Salesman

Results

This model was applied, among others, to a 4 cities situation which, due to its smaller size, was chosen to be presented in this report.
The actual results of this model are only the information about a valid syntax and semantics analysis. However, for the purpose of understanding, these are the results for the same model without the "outputs" section.

```
1  Microsoft (R) Orml Compiler version 0.2.1
2  Copyright (C) Microsoft Corporation 2008. All rights reserved.
3
4  Valid Syntax
5  Valid Semantics
6  ========Goals=========
7  Objective Value = [12]
8  ========Values=========
9  PathFromCityToCity[1,1] -> 0
10 PathFromCityToCity[1,2] -> 0
11 PathFromCityToCity[1,3] -> 1
12 PathFromCityToCity[1,4] -> 0
13 PathFromCityToCity[2,1] -> 0
14 PathFromCityToCity[2,2] -> 0
15 PathFromCityToCity[2,3] -> 0
16 PathFromCityToCity[2,4] -> 1
17 PathFromCityToCity[3,1] -> 0
18 PathFromCityToCity[3,2] -> 1
19 PathFromCityToCity[3,3] -> 0
20 PathFromCityToCity[3,4] -> 0
21 PathFromCityToCity[4,1] -> 1
22 PathFromCityToCity[4,2] -> 0
23 PathFromCityToCity[4,3] -> 0
24 PathFromCityToCity[4,4] -> 0
25 SequenceCityVisited[1] -> 0
26 SequenceCityVisited[2] -> 1
27 SequenceCityVisited[3] -> 0
28 SequenceCityVisited[4] -> 2
```

Please note that, the way the chosen TSP model works, the original city (1) will have a sequencecityvisite equal to 0, as well as, the first actual city. The correct order is, therefore, to go from city 1, to city 3, to city 2, to city 4 and back to city 1. This can be verified looking at the paths to visit.

H.6 Warehouse Picking Routes

```
1  Model {
2      Inputs:
3          Costs = Tables.WMSLocationDistance.Distance[WMSLocationOrigin,
               WMSLocationDestination];
4          StartLocation = "001-01-0";
5          EndLocation = "001-04-0";
6          RouteId = "000069_103"; // will be parameter
7
8      Indexes:
9          Locations = Classes.WMSGetRouteItemLocations(RouteId, StartLocation,
               EndLocation).location;
10     Inputs:
11         LocationNum = Count(Locations);
12     Variables:
13         Integer in [0 .. 1] PathFromLocationToLocation [Locations, Locations];
14         Integer in [1 .. LocationNum] SequenceLocationVisited [Locations];
15     Minimize:
16         Sum(i in Locations ;
17             Sum(j in Locations | i != j ; Costs[i, j] *
                   PathFromLocationToLocation[i, j]));
18     Constraints:
19         Sum(j in Locations | i != j && Exists(Costs[i,j]) ;
               PathFromLocationToLocation[i, j]) == 1
20             where(i in Locations | i != EndLocation);
21         Sum(i in Locations | i != j && i != EndLocation && Exists(Costs[i,j]) ;
                PathFromLocationToLocation[i, j]) == 1
22             where(j in Locations | j != StartLocation);
23         SequenceLocationVisited [i] - SequenceLocationVisited [j] + LocationNum
                * PathFromLocationToLocation[i, j] <= (LocationNum - 1)
24             where(i in Locations, j in Locations | j != i && i != StartLocation
                   && j != StartLocation);
25     Outputs:
26         Classes.WMSUpdateOrderTrans.sortcode[location, routeId=RouteId] =
               SequenceLocationVisited;
27 }
```

Figure H.5: ORML Model - Warehouse Picking Routes