

Faculty of Engineering of the University of Porto



FEUP

Operations Research Modeling Environment for an ERP System

Joaquim Carlos Figueiredo Rendeiro

Report of Project

Master in Informatics Engineering and Computing

Supervisor: João Carlos Pascoal de Faria (Auxiliar Professor)

July 2008

Operations Research Modeling Environment for an ERP System

Joaquim Carlos Figueiredo Rendeiro

Report of Project
Master in Informatics Engineering and Computing

v1.1

Approved in oral examination by the committee:

Chair:

External Examiner:

Internal Examiner:

31st July, 2008

Abstract

Modern organizations need to adapt their information and management systems with increasing frequency, in order to improve operational efficiency and respond better to market demands—in sum, to stay competitive.

Microsoft Dynamics AX is an ERP system that features extensive support for customization, via imperative programming, and for which the declarative modeling language ORML was created. ORML stands for Operations Research Modeling Language, a declarative language that supports database integration and aims at representing an operations research problem statement—a model—as closely as possible to its mathematical representation.

The opportunity to provide an integrated model development environment for the ORML language, targeted specifically at Dynamics AX, was identified and pursued. Such system, the ORML IDE, is integrated in the preferred customization environment for Dynamics AX—Dynamics AX itself. The ORML IDE provides assisted editing of ORML models and, by being integrated in Dynamics AX, allows easy customization of ORML models that can be shared across the organization, quickly tested and used in Dynamics AX.

Target user categories for the ORML IDE were identified, from which the concrete project requirements were set. The results of planning, designing and building the ORML IDE, integrating it with the ORML Interpreter and with Microsoft Dynamics AX, are also presented.

The result of the project was a fully functional IDE for the ORML language, its user interface designed to take advantage of users' habits and expectations regarding Windows applications and its core integrated with the Dynamics AX database and rich client user interface. Overall, the produced system responds to all the initially established objectives.

Concluding the work done on the project, it is noted that managing expectations and internal promotion of the project in the company are very important factors in the life cycle of the project, because engaging interested parties in the subject often yields valuable feedback.

Resumo

As organizações modernas necessitam de adaptar os seus sistemas de informação e gestão com cada vez mais frequência, de modo a aumentar a sua eficiência operacional e melhor responder às exigências do mercado—em suma, para se manterem competitivas.

O Microsoft Dynamics AX é um sistema ERP com extensas capacidades para customização, por via de programação imperativa, para o qual a linguagem de modelação declarativa ORML foi criada. ORML significa *Operations Research Modeling Language*, é uma linguagem de modelação orientada a problemas de investigação operacional que suporta integração com bases de dados e tem como objectivo representar problemas de investigação operacional—modelos—o mais próximo possível da sua representação matemática.

A oportunidade de criar um ambiente integrado de desenvolvimento dos modelos em ORML especificamente destinado ao Microsoft Dynamics AX foi identificada e levada a cabo. O sistema produzido, denominado ORML IDE (ou simplesmente IDE), integra-se no ambiente de customização do Dynamics AX—o próprio Dynamics AX. O IDE proporciona a edição assitida de modelos na linguagem ORML e por estar integrado no Dynamics AX permite a fácil customização de modelos que podem ser partilhados por toda a organização, rapidamente testados e usados no Dynamics AX.

Foram identificadas as categorias the utilizadores alvo deste projecto, a partir das quais os requisitos concretos do projecto foram estabelecidos. São também apresentados os resultados do planeamento, desenho e construção do IDE, integração com o interpretador de ORML e com o Microsoft Dynamics AX.

O resultado deste projecto foi um IDE completamente funcional para linguagem ORML, com um interface gráfico desenhado de modo a tirar vantagem dos hábitos e expectativas dos utilizadores de aplicações para Windows e funcionalidades integradas com servidor e aplicação cliente do Dynamics AX. No geral, o sistema produzido respone a todos os objectivos inicialmente estabelecidos.

Concluído o trabalho de projecto, é notado que a gestão de expectativas e promoção interna do projecto na empresa são factores muito importantes no ciclo de vida de um projecto, pois o envolvimento de pessoas interessadas no projecto é uma boa fonte de *feedback*.

Acknowledgements

To MIEIC (at the time represented by Prof. Raul Vidal), AlumniEI (represented by Pedro Castro Henriques) and Microsoft Development Center Copenhagen (represented by Hans Jørgen Skovgaard), thank you for providing this opportunity.

To IAESTE Denmark, thanks for the diligent support during my relocation.

Prof. Raul Moreira Vidal, thanks for the work and life lessons almost subliminally passed during classes and conversations.

Prof. Augusto Sousa, thanks for being understanding and not making our lives more difficult.

Hans Jørgen Skovgaard, thanks for kicking my brain around for a while at the interview and for the very useful book.

Prof. João Pascoal de Faria, thanks for the guidance and for the liberty.

Per Vikkelsøe, thanks for having a clear vision of the objectives of the project but not constraining us with that.

Scott McDonald, thanks for good advice at the right times.

Bruce Benton, thanks for the warm reception in the first day and for being an accessible (manager's) manager.

André Lamego, João Magalhães and Tiago Silva, thanks for the smooth integration in Microsoft and in Denmark (especially the parties). Special thanks to André Lamego for putting up with interrupting questions every once in a (short) while these five months.

My dear colleagues and friends (not too many, but still I fear that all your names wouldn't fit here), thanks for all these years (this is not a "goodbye", but a "see you soon!").

My most close family, Joaquim Carlos Fernandes Rendeiro, Maria Emília Figueiredo e Silva Rendeiro, André Figueiredo Rendeiro, a big thank you for making me grow (yes, even my "little" brother!).

Joaquim Carlos Figueiredo Rendeiro

Index

CHAPTER 1	INTRODUCTION	1
1.1	CONTEXT	2
1.2	PROJECT.....	3
1.3	MOTIVATION AND OBJECTIVES.....	4
1.4	REPORT OVERVIEW	5
CHAPTER 2	LITERATURE AND TECHNOLOGY REVIEW.....	7
2.1	INTEGRATION WITH MICROSOFT DYNAMICS AX 2009	7
2.1.1	User Interface Level	8
2.1.2	Database Level	9
2.2	INTEGRATED DEVELOPMENT ENVIRONMENTS (IDES)	10
2.2.1	IDES for Operations Research Modeling Languages	11
2.2.2	General Purpose IDEs for Programming Languages.....	16
2.3	GRAPHICAL USER INTERFACE DEVELOPMENT FRAMEWORKS	17
2.3.1	Windows Forms	17
2.3.2	Windows Presentation Foundation.....	18
2.4	GRAPHICAL USER INTERFACE AUTOMATION FRAMEWORKS	19
2.5	OTHER TECHNOLOGIES	19
2.5.1	Microsoft Math 4.....	19
2.5.2	Microsoft Office Primary Interop Assemblies for .NET	20
2.5.3	Office Open XML SDK	20
2.6	SUMMARY.....	21
CHAPTER 3	REQUIREMENTS OF THE ORML IDE.....	23
3.1	DOMAIN-SPECIFIC TERMS.....	23
3.2	ASSUMPTIONS AND CONSTRAINTS	24
3.3	PROJECT REQUIREMENTS.....	24
3.3.1	Overview	25
3.3.2	Integrated Development Environment.....	26
3.3.3	Common Components	29
3.3.4	Non-functional Requirements.....	30
3.4	OBJECTIVES BESIDES THE REQUIREMENTS.....	30
3.5	SUMMARY.....	31
CHAPTER 4	SOLUTION SPECIFICATION	33
4.1	OVERVIEW	33
4.1.1	Declarative Language	33
4.1.2	Complete System Architecture Overview	37
4.2	ACTIVE X WPF HOST.....	38
4.3	ORML IDE.....	39
4.3.1	Technologies	40
4.3.2	User Interface	41
4.3.3	Architecture.....	43
4.3.4	Testing Methodology.....	48

Index

4.4	COMMON COMPONENTS ARCHITECTURE	49
4.4.1	Common Data Layer Library.....	49
4.4.2	Models Management Library.....	50
4.5	DESIGN OF OTHER FEATURES	51
4.5.1	Convert ORML Model to Word Document.....	51
4.6	SCHEDULE.....	52
4.6.1	Joint Project Overview	53
4.6.2	ORML IDE Project Details.....	53
4.7	SUMMARY	55
CHAPTER 5	IMPLEMENTATION	57
5.1	WPF ACTIVE X CONTROL HOST.....	57
5.2	INTEGRATED DEVELOPMENT ENVIRONMENT	58
5.2.1	Code Editor.....	58
5.2.2	Visual Editor.....	60
5.2.3	Model Form	61
5.2.4	Main Form	62
5.2.5	IDE Utilities Library.....	65
5.3	OTHER FEATURES	66
5.3.1	Common Data Layer.....	66
5.3.2	Models Management Library.....	67
5.4	EVALUATION OF THE SOLUTION.....	67
5.4.1	Tests and Integration.....	67
5.4.2	Usage Experience	68
5.5	SUMMARY	70
CHAPTER 6	CONCLUSIONS AND FUTURE WORK	71
6.1	SATISFACTION OF THE OBJECTIVES.....	72
6.2	FUTURE WORK.....	73
REFERENCES		75
APPENDIX A	DECLARATIVE MATHEMATICAL LANGUAGES: STUDYING AN OPERATIONS RESEARCH PROBLEM (PRODUCTION PLANNING).	81
APPENDIX B	LIST OF FEATURES OF THE ORML IDE	84
APPENDIX C	PROTOTYPING RESULTS: WPF CONTROLS IN DYNAMICS AX.....	90
APPENDIX D	PROTOTYPING RESULTS: WPF USER INTERFACE AUTOMATION AND TESTING.....	92
APPENDIX E	AOT-LIKE MODELS TREE STRUCTURE DEFINITION.....	95

List of Figures

Figure 1: Dynamics AX Forms Designer.	8
Figure 2: The AIMMS IDE user interface.	12
Figure 3: The ILOG OPL Development Studio IDE user interface.	13
Figure 4: The LINGO IDE user interface.	14
Figure 5: The MPL IDE user interface.	15
Figure 6: The Mathematica IDE user interface.	16
Figure 7: User interface of Microsoft Math 4.	20
Figure 8: High-level system components diagram.	25
Figure 9: ORML IDE use case diagram.	26
Figure 10: Blocks diagram illustrating the technology stack of the joint project.	38
Figure 11: Components diagram of the joint project.	38
Figure 12: The Decorator design pattern conceptual diagram [44].	39
Figure 13: Class diagram of the ORML ActiveX WPF Host.	39
Figure 14: The .NET Framework 3.5 technology stack [52].	40
Figure 15: Early IDE UI sketch.	42
Figure 16: Abstract UI sketch for the IDE (showing the code editor, database explorer and model error list tabs).	43
Figure 17: Abstract UI sketch for the IDE (showing the visual editor, models explorer and model output tabs).	43
Figure 18: Initial version of the ORML IDE components architecture.	44
Figure 19: The architecture of the ORML IDE—class diagram.	45
Figure 21: UI sketch of the Visual Editor control.	46
Figure 20: The <code>TokenStyleInfo</code> structure.	46
Figure 22: Documentation example for the automated tests.	48
Figure 23: Documentation example for the manual tests.	48
Figure 24: Adapter design pattern conceptual diagram.	49
Figure 25: Commn Data Layer adapter structure and support classes.	50

List of Figures

Figure 26: Models Management Library classes.	51
Figure 27: Conversion from ORML to Word (OpenXML) sequence diagram.	52
Figure 28: Schedule of the joint project.....	53
Figure 29: Detailed activities schedule of the IDE project.	54
Figure 30: The FDD project lifecycle.....	55
Figure 31: ActiveX WPF Host—class and interfaces in the assembly.	58
Figure 32: <code>Orml.Ide.CodeEditor</code> API, organized by functional sections.	59
Figure 33: <code>Orml.Ide.ModelForm</code> showing code editor, model outline and error list.	61
Figure 34: The ORML IDE user interface.....	62
Figure 35: The Dynamics AX Application Object Tree (AOT).	63
Figure 36: IDE screenshot featuring the context menu for a model tree item and the parameters editing window.....	64
Figure 37: Common Data Layer class diagram (excl. adapters).	67
Figure 38: Margin of error for testing with various numbers of users [47].	68
Figure 39: Dialog box presenting the choices available when conflicts arise while saving a model.	70

Glossary

ActiveX

ActiveX is a Component Object Model (COM) (see the definition of ***Component Object Model*** below) developed by Microsoft that incorporates functionality previously made available by Object Linking and Embedding (OLE), an inter-process communication mechanism available to Windows applications [53]. This technology is generally used to build visual controls meant to be used as components of other applications' graphical interfaces (a program that displays Web pages may use Microsoft Internet Explorer as an ActiveX control) or as enhanced controls in websites (Microsoft Internet Explorer supports ActiveX objects; these can be used to provide additional security when accessing sensible Web sites, like banking sites, for example).

Application Object Tree (AOT)

See ***Dynamics AX Application Object Tree***, in the next page.

Application Programming Interface (API)

A set of declarations of the procedures, objects, methods, events and other types of programmable elements provided by a system, program, library or service.

Component Object Model (COM)

COM is a Microsoft technology that aims at being a language-independent way of implementing objects that can be combined and re-used. It has been present in Microsoft Windows since Windows 3.1 and while it has been superseded by ActiveX, it is still supported to this date. Although it may be considered that COM has been made obsolete by .NET, those are separate but complementary technologies [54].

Deserialization

See ***Serialization / Deserialization***, in page xiii.

Dynamics AX Application Object Tree (AOT)

The Application Object Tree is the collection of programmable presentation elements, business logic elements, data model elements, application resources, documentation entries and other application-specific elements (like menus and configuration keys), hierarchically organized [20].

Ecma International

Ecma International is an industry association dedicated to the standardization of Information and Communication Technology and Consumer Electronics [37].

Hypertext Markup Language (HTML)

HTML is a markup language that serves to describe the structure and content of a document that contains multiple element types (paragraphs, headings, lists and tables, to cite a few) and references to sections in the same document or even in other documents.

Integrated Development Environment (IDE)

“Since the early 1980s, increasingly sophisticated tools have been built to aid the software developer and to automate as much as possible the development process [1].” In the current age of software development, IDEs are quite common tools.

An IDE can be defined as program that covers a broad set of needs of the developer, from the most basic features like text editing and compiling the code or project to more advanced functionality like syntax highlighting, debugging, integration with source control systems, database navigation, among others [2]. In sum, an IDE should aim to be as complete as possible in order to provide support for all activities that the user will need, so that the user’s productivity suffers the least possible from switching between different tools to accomplish any task.

Language- Integrated Query (LINQ)

“The LINQ Project is a codename for a set of extensions to the .NET Framework that encompass language-integrated query, set, and transform operations. It extends C# and Visual Basic with native language syntax for queries and provides class libraries to take advantage of these capabilities. [...] .NET Language-Integrated Query defines a set of general purpose standard query operators that allow traversal, filter, and projection operations to be expressed in a direct yet declarative way in any .NET-based programming

language. The standard query operators allow queries to be applied to any `IEnumerable<T>`-based information source” [27].

Mathematical Markup Language (MathML)

“A product of the W3C Math working group, MathML is a low-level specification for describing mathematics as a basis for machine to machine communication” based on XML [55].

Model View Controller (MVC) design pattern

“The essential purpose of MVC is to bridge the gap between the human user's mental model and the digital model that exists in the computer. The ideal MVC solution supports the user illusion of seeing and manipulating the domain information directly” [56].

Off-the-shelf

In the context of this document, the term off-the-shelf refers to a product or technology that meets most of the requirements searched for and/or has a good available functionality vs. price ratio.

Primary Interop Assembly (PIA)

A Primary Interop Assembly (PIA) is a regular .NET assembly—it contains a collection of types that are deployed, versioned and configured as a single unit. It has the particularity of containing only the type definitions (and not implementations) of types that have already been defined in COM, allowing .NET applications to use the counterpart COM objects as regular .NET objects, because these types provide static compile- and run-time information to the .NET environment [40].

RecordSet

A RecordSet “represents the entire set of records from a base table or the results of an executed command” [57]. A record can be defined as collection of name-value tuples. Typically, the records in a RecordSet all contain the same collection of “names”, differing only on the “values” associated with those “names”.

Serialization / Deserialization

Serialization is the process of saving an object (commonly including other objects contained in the original object to be serialized) in textual or binary form in order to store

that object or transmit it across a network. Deserialization is the inverse operation: reconstructing an object from its textual or binary representation.

Window Handle (HWND)

A handle is a kind of pointer that is used by an application to reference objects managed by other applications or systems. A Window Handle is a handle for a window managed by the Windows operating system.

Extensible Markup Language (XML)

“The Extensible Markup Language (XML) is a simple, very flexible text format [...]. Designed to meet the challenges of large-scale electronic publishing, XML is also playing an increasingly important role in the exchange of a wide variety of data” [58]. Due to its very generic nature, XML has been used as the base for the specification of many data formats, called XML dialects.

XML dialect

See *Extensible Markup Language* above.

Extensible Stylesheet Language (XSL)

XSL is a set of recommendations for defining XML document transformations and presentation, composed by the XSL Transformations (XSLT, a language for transforming XML into various formats), the XML Path Language (XPath, an expression language used by XSLT to refer to parts of an XML document) and the XSL Formatting Objects (XSL-FO, “an XML vocabulary for specifying formatting semantics”) [59].

X++ Programming Language, The

X++ is an object-oriented language with similarities to C++ and C#. Programmers using X++ can access existing Microsoft Dynamics AX classes that provide functionality that ranges from basic I/O and inter-process communication to modifying controls in the graphical user interface at run time [20] [60].

List of Acronyms

AOT	Application Objects Tree
API	Application Programming Interface
COM	Component Object Model
ERP	Enterprise Resource Planning
IDE	Integrated Development Environment
FEUP	Faculty of Engineering of the University of Porto, Portugal (<i>Faculdade de Engenharia da Universidade do Porto</i>)
HTML	Hypertext Markup Language
HWND	Window Handle
LINQ	Language Integrated Query
MathML	Mathematical Markup Language
MDCC	Microsoft Development Center Copenhagen
MVC	Model View Controller
OLE	Object Linking and Embedding
PIA	Primary Interop Assembly
SDK	Software Development Kit
XAML	Extensible Application Markup Language
XML	Extensible Markup Language
XSL	Extensible Stylesheet Language

List of Acronyms

Chapter 1

Introduction

This thesis is the result of a project that ran over the course of sixteen weeks, with the objective of producing an application that would allow easily editing operations research problem models written in a declarative language with data-binding capabilities (named ORML, for Operations Research Modeling Language), supporting that editing process with features and tools that would provide ease of integration with the data of an existing ERP system—Microsoft Dynamics AX. The project was developed jointly with another project—specifically, the creation of the referred declarative modeling language and its interpreter (the ORML Interpreter)—in Microsoft Development Center Copenhagen, Denmark.

The project involved researching the state-of-the-art on existing IDEs for mathematical and programming languages, selecting the appropriate manner of interacting with Dynamics AX database (both data and metadata) and client user interface, designing and developing the ORML IDE once the requirements were defined, integrating it with the ORML Interpreter and participating in the creation of some libraries shared by both the Interpreter and the IDE.

This dissertation covers the background, motivation, research done, planning and executing steps, achieved results and evaluation of the project.

This dissertation does not cover the creation and optimization of models for operations research problems, the justification for the language constructs or the detailed architecture of the ORML Interpreter.

1.1 Context

Since their adoption by organizations, information systems have grown in many levels (functional areas that they cover, volume of information they must handle, user expectations, to cite a few) and have come to support organizational processes with increasing levels of complexity (from transaction-processing systems to decision-support systems, management information systems and executive support systems). The goal of an organization when implementing these systems is to achieve full integration horizontally—different management levels in the same functional unit—and vertically—different functional areas in the same management level—, creating information flows and workflows that lead to the best operational efficiency [34].

Vertically, systems for specific functional areas (like sales, finance, manufacturing and others) are usually very good at supporting the operations of those areas because of their specificity and the years of evolution these systems have gone through. However, systems that are horizontal to an organization—the case of ERP systems—often need to be adapted to the reality of each organization in order to match the individual organic and business processes of the organization, integrating their information and processes with eventual specific vertical systems of that organization and even with external systems and processes [34]. This adaptation of the ERP system to specific organizational processes, optimizing those same processes, if possible, is one of the objectives of operations research.

“Operations research is not a science itself but rather the application of science to the solution of managerial and administrative problems, and it focuses on the performance of organized systems taken as a whole rather than on their parts taken separately” [33].

These problems, after being carefully studied, are often defined in the form of a mathematical model, with input data, variables, constraints and goals (like minimizing resource waste or maximizing revenue). With increasing frequency, there is the need to formally model problems based on Linear and Quadratic Programming models (LP, QP), Constraint Satisfaction problems (CSP) and Satisfiability problems (SAT).

The mathematical formulation of these problems tends to be compact and formally verifiable, but there is the need to easily take these models and translate them into code that interacts with the systems used by the organizations that want to solve these problems. While the models have quite succinct formal definitions, they require much more lines of code to implement when it comes to not only implementing the models themselves but also taking into account the database interactions (both inbound and outbound).

Introduction

By having a declarative language simple but capable of expressing these kinds of problems, it is possible to drastically reduce both the time to implement such models into the product and the errors that arise from very complex implementations done with imperative programming languages. For that purpose, the domain specific language ORML (Operations Research Modeling Language) was created. This language provides a syntax that aims to be very close to the mathematical formulation of the referred models and, at the same time, allows referring to data from Microsoft Dynamics AX.

Dynamics AX is a very large and complex ERP system, not only in terms of the underlying technology but also in terms of available functionality. It is targeted at mid-sized and larger organizations, providing a broad set of features for business areas like Trading, Finance, Production/Manufacturing, Sales, Human Resource Management, Supply Chain Management, among others. Microsoft Dynamics AX 2009 was released to public on June 2008, together with documentation and tools both for developers and end-users [36].

As the name indicates, Microsoft Dynamics AX is produced by Microsoft Corporation, an American multinational computer technology company. Despite being present in the peripherals and consumer devices markets, it was founded (by Bill Gates and Paul Allen, in 1975) to be very targeted at the software market and still keeps that focus in the enterprise and consumer markets—the Microsoft Windows operating system and Microsoft Office productivity suite both have very high market shares. On the enterprise level, Microsoft has several technological platforms that are used by organizations to build their systems (e.g.: Microsoft Windows Server, Microsoft SQL Server, Microsoft SharePoint, to name a few) but also offers almost off-the-shelf management solutions (that usually are customized by partners for the specific needs of customers) that have been grouped under the “Dynamics” branding, like Dynamics C5, Dynamics CRM, Dynamics GP, Dynamics NAV and Dynamics AX.

Microsoft Dynamics AX has a large development force at the Microsoft Development Center in Copenhagen (MDCC)—the largest development center of Microsoft outside the U.S., employing about 900 staff in total, from 44 different nationalities. The global product management for Dynamics NAV and Dynamics AX are also located at the MDCC.

1.2 Project

In order to achieve a tighter integration between the production and maintenance of the models referred in the previous section and the environment where these models will be used—Microsoft Dynamics AX—it is desirable to have a model development environment

Introduction

that is tightly integrated with Dynamics AX data and user interface. That integrated model development environment is the focus of this project (herein after referred to as ORML IDE).

This project is part of a joint project to create both the ORML IDE and the ORML Language and Interpreter. The ORML Interpreter project runs in parallel with this project, the elements of both projects working closely and communicating at all times.

1.3 Motivation and Objectives

Having a model development environment tightly integrated with Dynamics AX will not only allow Microsoft to easily develop the models against Dynamics AX data and business logic but also empower partners and customers to make customizations of those models in order to better fit a particular business model (since the models expose business logic), by making the models readily available.

This project was also motivated by the need to have a model editing experience that supports the user when editing the models and integrating them with Dynamics AX data sources, by opposition to editing a text file in a simple text editor.

The objectives of the project are the following:

1. Research and document the state of the art on methodologies and technologies regarding the type of problems specified and how they are modeled and solved, and on current IDEs and other solutions to the referred kind of functionalities;
2. Define the requirements for the ORML IDE;
3. Prototype a simple user interface that is integrated into Microsoft Dynamics AX's graphical user interface and interacts minimally with the database and ORML Interpreter to solve a simple LP problem, in order to prove the feasibility of the project as it was defined initially or review the project definition, if necessary;
4. Design and implement the ORML IDE;
5. Provide integration with the ORML Interpreter;
6. Provide integration with Microsoft Dynamics AX database, both in terms of data and metadata;
7. Provide the feature of exporting the model to a file, in a human-readable, textual and mathematical representation;
8. Use state-of-the-art and in-house ("house" here being Microsoft) technologies, when possible;

Introduction

9. Provide a good user experience and overall responsiveness of the ORML IDE.

In order to achieve these objectives, an iterative software development approach was followed, with the purpose of having more frequent deliveries of “pieces” of the product and getting early feedback. There were weekly project meetings, as well as defined milestones (detailed in Section 4.6, Schedule).

1.4 Report Overview

Chapter 1, Introduction, contextualized the project by presenting the background of the project and its motivation and objectives.

Chapter 2, Literature and Technology Review, illustrates the research done regarding previous work and available technologies in the context of this project.

Chapter 3, ORML IDE—Detailed Problem Description, gives detailed information on the problem, what were the specific goals of the project and what methodologies were used to achieve the objectives.

Chapter 4, Solution Specification, specifies the solution that was planned in order to achieve the objectives of the project, presenting the detailed system design, the architecture, the testing methodology and the project schedule.

Chapter 5, Implementation, consists of the solution implementation details. In this chapter are focused the noteworthy details of the system implementation and interaction logic. It is also part of this chapter an evaluation of the solution, both in terms of tests implemented and usage experience.

Chapter 6, Conclusions and Future Work, presents a review of the project in general, considering the purpose of the work and achieved results.

Introduction

Chapter 2

Literature and Technology Review

This chapter presents a summary of previous works related to the subject of this project and an analysis of available technologies that may be relevant to solve the problem briefly described in the previous chapter.

2.1 Integration with Microsoft Dynamics AX 2009

ERP systems nowadays generally follow the client-server architecture, often with thin clients (the Web browser, for example) and rich clients (full-fledged programs). Dynamics AX makes available its functionality through both these kinds of clients—the Dynamics AX Enterprise Portal (EP) is an ASP.NET application that can be accessed through the browser and the Dynamics AX Client Interface is a first-class desktop application that has to be installed on the local system. However, the target of this project is to achieve integration with the server and with the rich client, not the thin client.

Customizing Dynamics AX or adding extra functionality can be done directly from Dynamics AX itself—not only providing more automation or business rules but also creating new tables, queries and even forms—in a separate layer from the Dynamics AX core and system layers, which means that the customizations remain even if the application is re-installed/upgraded. This approach requires the developers to learn the X++ programming language and to use the built-in form designer.

There is however the possibility of loading external controls via ActiveX and interacting with them, even if that interaction is always limited to the least common denominator, imposed by the COM interoperability limitations. While the integration might not be easy, this is an option to take into account when developing relatively isolated components, since

Literature and Technology Review

the development effort may be smaller and there is still the possibility to interact with the Dynamics AX database, via the Business Connector (that is available both by .NET and COM).

If the extensions don't involve graphical elements, it's possible to directly import .NET libraries and use .NET classes inside Dynamics AX [20].

2.1.1 User Interface Level

For the purpose of creating an IDE that is integrated into the user interface of Dynamics AX, it would be very hard to do it using only the internal designer plus native X++ code and deliver a good user experience, since the integrated form designer is limited to the most common control types (that are typically not easy to extend or whose internal behavior is not easily modifiable) and some specialized ones [20].

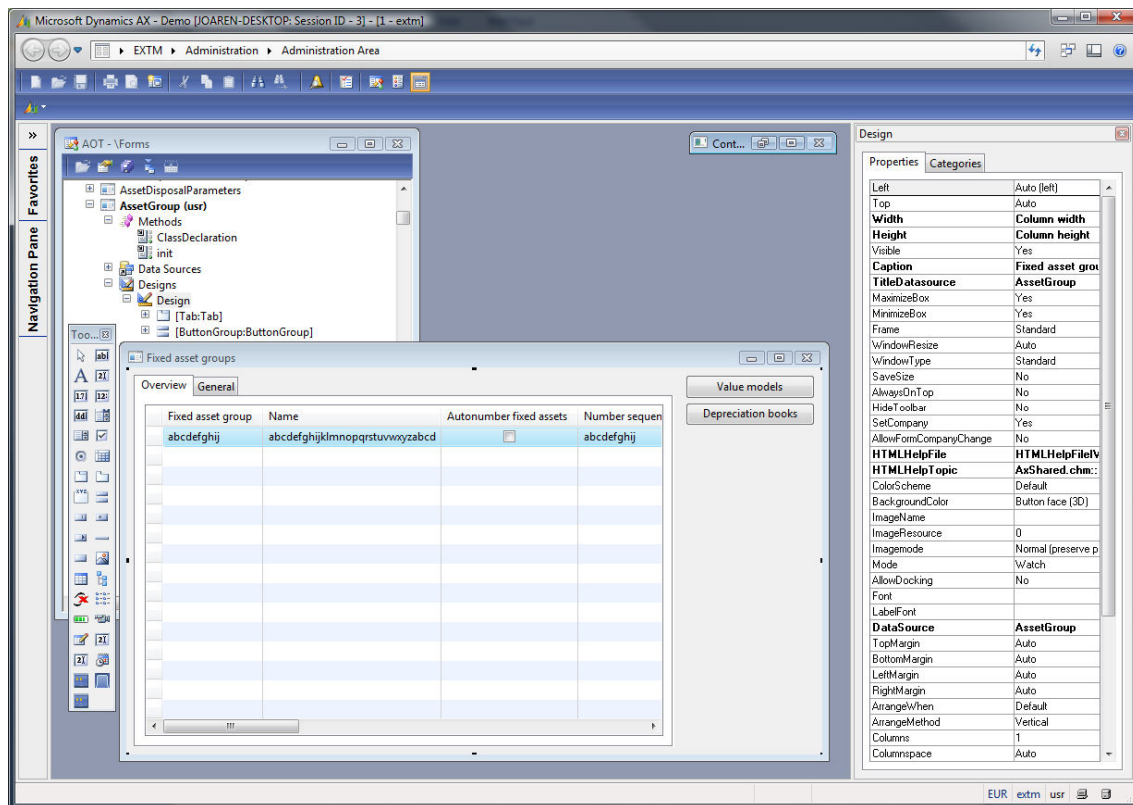


Figure 1: Dynamics AX Forms Designer.

However, since there is the possibility to use ActiveX controls, the developer has more freedom for choosing the most suitable development environment as long as the components are exported and visible by Dynamics AX. Things like event handling from the control to the host form and vice-versa have to be implemented in a custom case-by-case basis if needed, but the control is normally usable otherwise.

2.1.2 Database Level

There are two purposes when it comes to interact with the database from the IDE:

- The main one is to get the list of all the tables, their fields and respective types, in order to provide the user with a quick reference (possibly hierarchical, grouped by business area) to the Dynamics AX database fields that can be used in the models;
- The secondary objective is to provide to the user a preview of the contents of the table, which can help to become more familiar with the data itself.

The three possible ways to interact with the Dynamics AX database are the following [20]:

1. Natively in Dynamics AX, using X++;
2. In .NET, through the Dynamics AX Business Connector Library; and
3. In pure .NET, using only the framework's classes.

When accessing the database natively in Dynamics AX, through the facilities X++ provides and through the Dynamics AX extensive class library, it's possible to use forms, custom classes, classes that represent database tables, views and much more. This alternative abides by all the business logic encoded in the database entities.

However, it's not trivial to pass complex data like complete RecordSets or custom data types to the outside environment or to use the data objects in a statically typed extern environment, because X++ is an interpreted language, which means that the structure of the objects is not known beforehand.

In .NET, through the Dynamics AX Business Connector library, the second alternative presented above, it's possible to mirror most of the data access capabilities that are available natively. This method also does not circumvent any business logic possibly attached to the database objects—the Business Connector internally translates the calls to X++ calls.

This second option provides more support to the .NET environment when dealing with RecordSets and respective operations, but still keeps some level of abstraction over the data model and is database-agnostic;

Finally, in pure .NET it's also possible to access the database server directly, via ADO.NET (or even using LINQ); however, it's necessary to take into account that the abstraction over the database type¹ provided by the two alternatives mentioned above is not available, which means that more specific commands may not be compatible with all types

¹ Dynamics AX currently supports Oracle Database 10g and Microsoft SQL Server 2000 and 2005.

of databases. Besides that, any business logic attached to database entities is ignored when accessing the database directly, because business logic and even table constraints reside in the application classes and not in the database.

2.1.2.1 Enumerating database elements

Listing all the tables, views or queries (kind of stored procedures) together with their associated fields and field types is possible from native X++, pure .NET (direct database access) and by using the Business Connector from .NET. Moreover, the most desirable options are to use native X++ development or to use the Business Connector, since they abstract the database type.

When it comes to obtaining the relations between tables, it's a kind of information that only exists completely in the application layer of Dynamics AX, which makes it impossible to obtain when accessing the database directly from pure .NET [20].

2.1.2.2 Data Access

When it comes to regularly accessing data from Dynamics AX database, it's easy to do it either natively (with X++) or from .NET (with the Business Connector) and take advantage of the database abstraction provided [20].

The restrictions referred in section 2.1.2 still apply when accessing data directly from the database server: any business logic attached to the database entities is bypassed, which can result in inconsistent application data. This condition alone pretty much excludes the option of directly accessing the database, not to mention that Dynamics AX supports different kinds of databases that would also have to be supported via direct access.

2.2 Integrated Development Environments (IDEs)

The purpose of analyzing existing IDEs was mainly to gather a set of useful features and characteristics (and features or characteristics to avoid) that are present separately in IDEs for operations research modeling languages and IDEs for programming languages.

Nevertheless, in order to identify possible candidates that could be used or re-used almost off-the-shelf—i.e., that would provide a broad base to solve the problem at hand—special attention was paid to existing mathematical IDEs that would support plugging different languages and to general-purpose programming IDEs that might be easily extendable (and that conform to other important requirements of the project, like integration with Dynamics AX user interface and data).

2.2.1 IDEs for Operations Research Modeling Languages

The following sections refer only to five IDEs—these are the few IDEs that actually are developed for operations research or optimization modeling languages. Most solutions provide only a compiler or interpreter and an API that can be used to harness the power of the optimization library in another application.

A more exhaustive list of optimization software was found on [11] and [12]; however, many solutions seem to focus a lot on the strength and completeness of their solvers, APIs and languages, relegating to a lesser level the support tools that aid the model development and maintenance processes.

2.2.1.1 *AIMMS 3.8*

AIMMS stands for Advanced Interactive Mathematical Modeling Software. It's a comprehensive software package for creating decision support and advanced planning applications, divided into multiple layers of components that support custom GUI construction, model building based on different data sources and integration with many solvers for actually solving the models [3]. The 3.8 version was released in 2008.

The AIMMS user interface is more than a coding IDE; it allows for the user to have a custom visualization of the model solving results (e.g.: pie charts, 3D-plotting) or even to build a complete standalone application customized for a certain model (e.g.: planning the flows of products within a manufacturing unit) [4]. However, the basic functionality hasn't been disregarded altogether—the editor provides syntax highlighting for its custom language and a number of controls that improve the user experience while creating models and using AIMMS, some of which are noted here:

- The model explorer control is a hierarchical view of the model sections that provides an overview of the model and allows quick navigation to its sections;
- The identifier selector control provides a more in-depth perspective of the model's variables, constraints and other named identifiers, whose filter can be adjusted to find identifiers with common attributes and possibly performing operations on them;
- The data manager helps the user to define how (and where from) the model data is retrieved and how results are stored depending on different scenarios specified for the model, by providing controls and dialogs that ease the task of defining data interactions with the model.

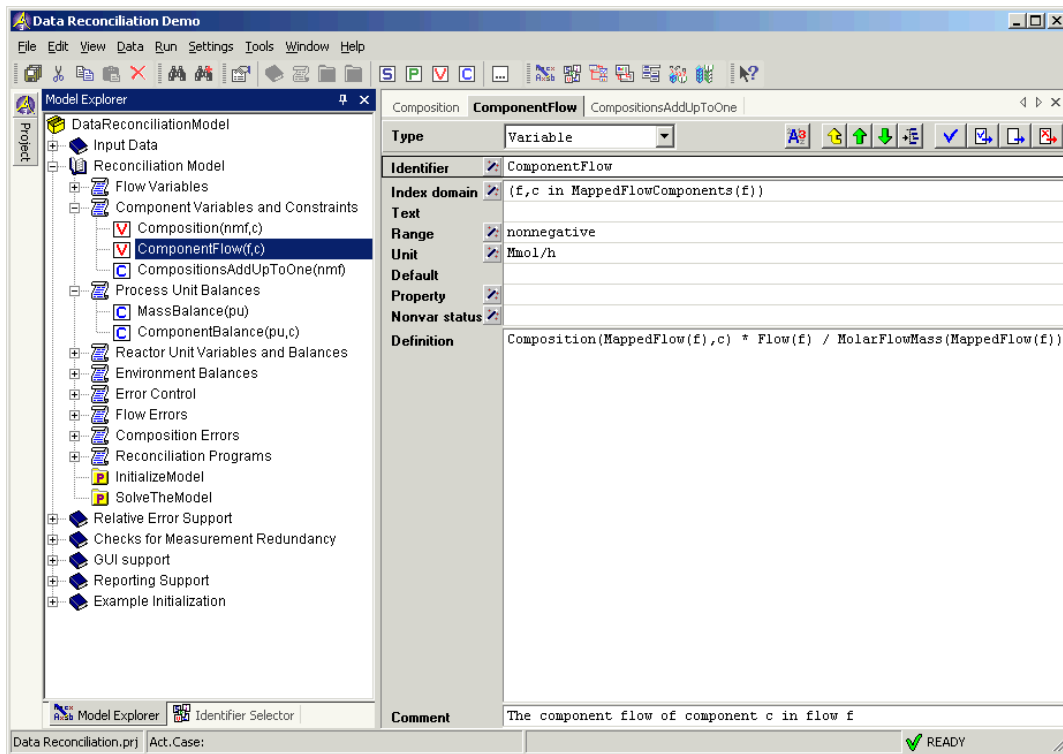


Figure 2: The AIMMS IDE user interface [4].

2.2.1.2 ILOG OPL Development Studio 6.0

ILOG's development studio, supported on their Optimization Programming Language (ILOG OPL), has an IDE that aims at supporting the user on evaluating different modeling approaches to a problem, integrating external data and reviewing results [5]. It was released in 2008 and it's based on the Eclipse Rich Client Platform (Eclipse RCP), which will be referred in section 2.2.2.

ILOG OPL Development Studio's IDE also provides syntax highlighting in the text editor. Other noteworthy features regarding the IDE are the following [6]:

- A hierarchical view of the model structure that can be navigated through;
- Contextual help constantly updated during the interaction of the user with the IDE;
- The ability to solve models in debugging mode.

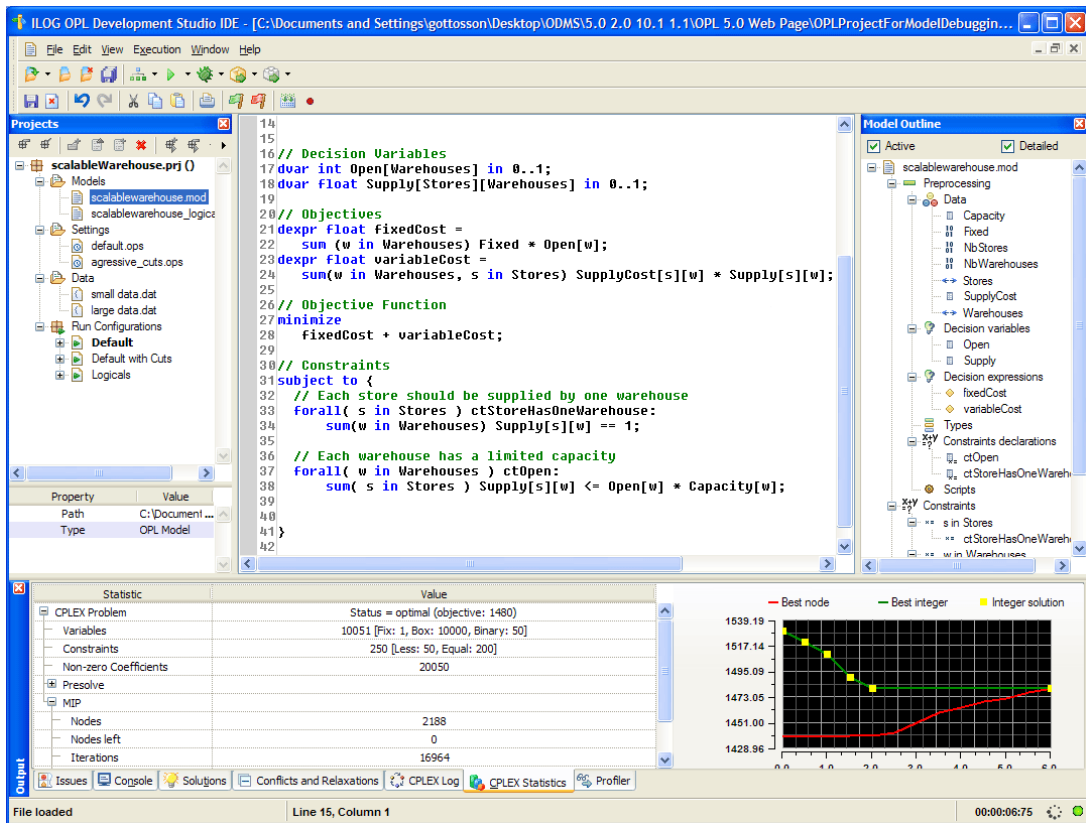


Figure 3: The ILOG OPL Development Studio IDE user interface [5].

2.2.1.3 LINGO 11.0

LINGO is both a modeling language and an IDE very integrated with the LINDO set of solvers. The IDE specifically supports the building, solving and analyzing activities of modeling optimization problems [7]. LINGO 11.0 was released in 2008.

The LINGO IDE provides syntax highlighting in the model code editor, but, other than that, the model edition experience is only supported by error reports when the user tries to solve the model (if there are any syntax or semantics errors, for example). No tools are available to assist with managing large models or exploring data sources (just to use as an example two of the most common features in these types of IDEs).

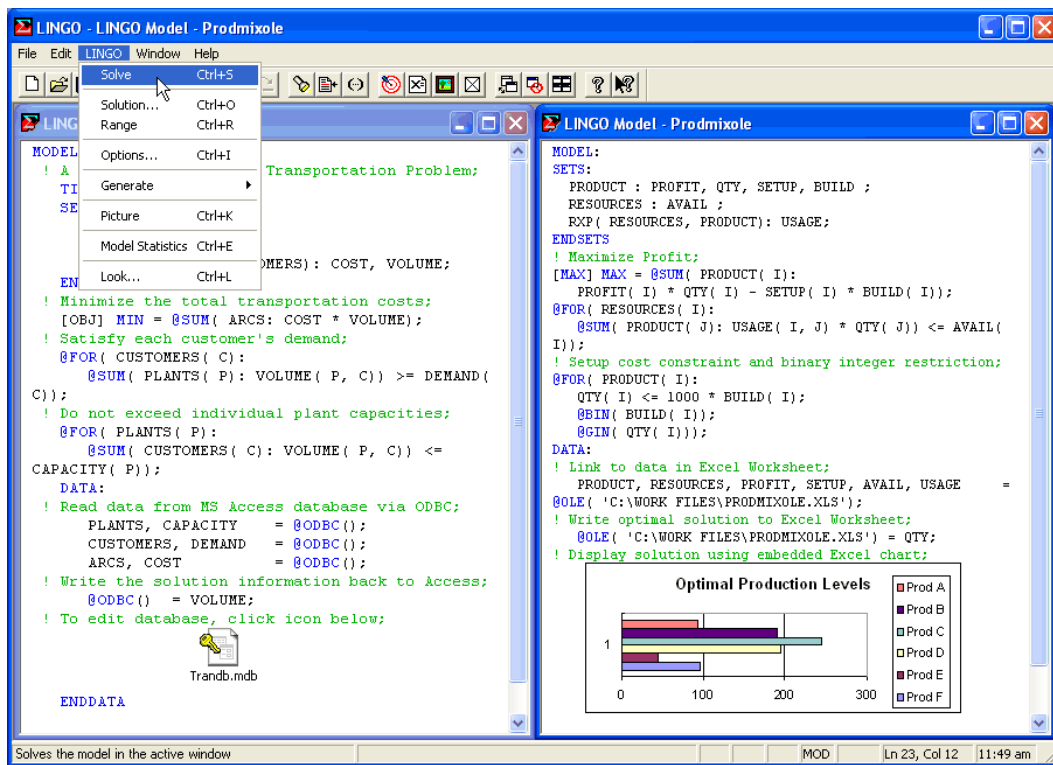


Figure 4: The LINGO IDE user interface [7].

2.2.1.4 MPL 4.2

Maximal's MPL (Mathematical Programming Language) is not simply a modeling language, but a complete modeling system that can use several external solvers to actually solve the models. The IDE itself aims at supporting the user throughout model formulation, model data management, model optimization and solution analysis phases [8]. Version 4.2 is the most recent, released in 2002.

The IDE provides a hierarchical navigable view of the model elements like data sources, variables, constraints and others. It also provides more user-friendly views of the solution of some models as 2D graphics.

However, the model creation process itself could be more interactive—the IDE doesn't provide syntax highlighting when editing the model text and only provides information about the correctness of the model once the user compiles it.

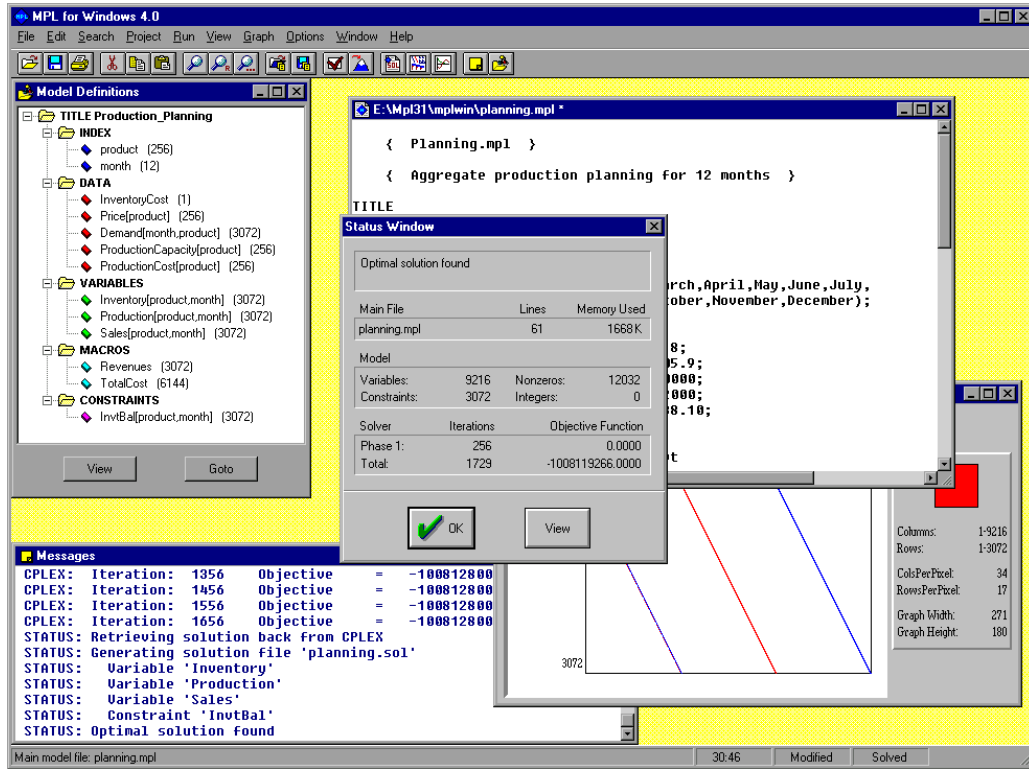


Figure 5: The MPL IDE user interface [8].

2.2.1.5 Wolfram Mathematica 6

Mathematica is a general-purpose solution that supports computing, modeling, simulation, visualization, development, among other activities, backed by a high-level interpreted language [9]. It was released in 2006 and it's based on the Eclipse Rich Client Platform (Eclipse RCP), which will be referred in section 2.2.2.

Being more general-purpose than the previously presented IDEs, it's more close to an IDE for a programming language, having features like support for projects, tracking tasks and debugging. The expressiveness of the mathematical language and the back-end engine provide support for solving many kinds of mathematical problems, Linear Programming and Constraint Programming models being just a few among many [10]. Precisely because of this, Mathematica is much more limited when it comes to providing specialized support for the user that is trying to model optimizations problems and provides little integration with external data.

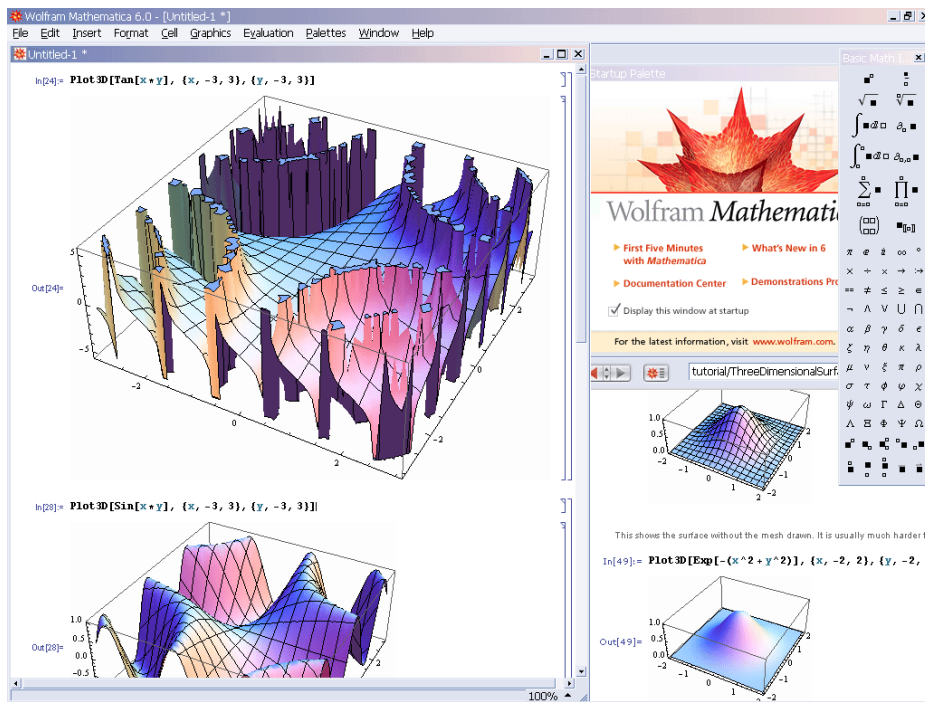


Figure 6: The Mathematica IDE user interface [9].

2.2.2 General Purpose IDEs for Programming Languages

Analyzed from a basic perspective, most modern IDEs for programming languages share a common set of traits [13] [14] [15] [16] [17] [18], the most common being the following:

- Code editor with syntax highlighting and auto-completion;
- Code refactoring facilities;
- Managing whole projects (as opposed to working with loose files);
- Create-edit-compile-run-debug cycle fully supported;
- Automated continuous error checking (both syntax and compilation errors);
- Integration with unit testing frameworks;
- Generating documentation from project code;
- Pluggable architecture that allows extra functionality to be developed by third-parties;
- Integration with external systems (e.g.: importing libraries, using Web services) and data sources (from simply connecting and navigating through databases to generating bindings to work with those data sources, like Hibernate or LINQ bindings).

In addition to these features, some of these IDEs have taken advantage of their modularity and pluggable architecture to form a Rich Client Platform (RCP)—a base for application development that provides basic functionality that complex applications nowadays may find useful (not only software development applications, but any kind). Examples of such are the Eclipse RCP and the Microsoft Visual Studio Shell.

The Eclipse RCP and Visual Studio Shell are both, in their essence, composed of a core platform, a plug-in system and a collection of graphical controls/widgets that can be easily re-used and extended. Eclipse RCP's UI logic has the particularity of being based on the Model View Controller pattern [13]. Another common trait is that both platforms must run as a first class application, and their components can't be easily embedded in other applications.

2.3 Graphical User Interface Development Frameworks

It's a definite need to integrate the IDE for the ORML language with the user interface of Microsoft Dynamics AX (which is a Windows application). This leads to the possibility that a solution might have to be created specifically for this problem. In order to study this possibility, the most relevant Graphical User Interface (GUI) frameworks available for the construction of complex GUIs were analyzed.

2.3.1 Windows Forms

Windows Forms (released in 2002 and later updated in 2005), or WinForms, started as an evolution of the Microsoft Foundation Classes (MFC) library, building new features and capabilities and keeping the support for some low-level interaction. For example, all the controls have a Window Handle (HWND), which means that they directly receive and process messages from the operating system and that any control can be exported as an ActiveX control [23].

However, while WinForms was a big leap in terms of productivity and ease of use for the application developer when compared to MFC, used in .NET 1.0, 1.1 and 2.0 (and still supported in .NET 3.0 and 3.5), it's still not very far away from WIN32 in terms of amount of work necessary to customize the aspect and, more importantly, the behavior of controls, when compared to the flexibility and customization required by today's standards.

Applications that use this graphical library often end up tightly coupling their logic and visual aspects, simply because, to this date, there were not many alternatives to designing the application's UI in the Visual Studio forms designer. Some efforts were by made

Microsoft to invert that situation, by releasing a MVC-based framework initially targeted at WinForms in .NET 2.0 [24], but the community participation points to a low rate of adoption and many difficulties encountered during development [25].

2.3.2 Windows Presentation Foundation

Windows Presentation Foundation (WPF) is the new-generation UI framework that ships as part of Microsoft .NET Framework 3.0 and 3.5 (released in 2006, updated in 2007). It can be used programmatically, similar to WinForms, but its main cornerstone is the usage of Extensible Application Markup Language (XAML, an XML dialect) to declaratively define the windows, controls, layout, properties and styles (and even the bridge to event handling in code) in a very flexible way.

Contrary to WinForms, this framework is not based on the traditional WIN32 HWNDs (only the topmost windows have a Window Handle), which allowed great freedom when defining the component model and other features like [21] [22]:

- Great control over the layout of the controls—from fixed pixels to automatic stretching, scaling and positioning based on grids with different layout settings;
- Almost no limitation to which kinds of elements can be contained in containers (not only panels or grids, but also buttons, list/menu items, text boxes, ...);
- Styles and control templates that can apply a custom design and add controls to many elements (and even different kinds of elements) at once;
- More fine-grained event control, with tunneling events (from containers down to contained elements), bubbling events (from contained elements up to their containers) and direct events (fired only at the specific target element);
- Data binding (between controls and also with objects), dependency properties (to ease data binding, value change notifications and allow property values inheritance) and attached properties (properties defined in one element that child elements can use to provide additional information to the parent control).

This incomplete but representative list of WPF features is a good example of how WPF is aiming at getting the best of the desktop programming model (e.g.: rich controls and events, static code, data binding) and also the best of the web programming model (e.g.: good layout/auto-layout control, scaling, easy control composition, styles, templates), adding even multimedia capabilities (e.g.: 2D and 3D animations, audio, video) [26].

2.4 Graphical User Interface Automation Frameworks

To test the interface, available user interface automation frameworks were analyzed. In the Microsoft Windows environment, until recently there was only one user interface automation (and accessibility) framework developed and recommended by Microsoft—Microsoft Active Accessibility.

Microsoft Active Accessibility, as the name hints, was designed mainly with accessibility in mind. It's a COM-based technology available since 1997 that exposes a useful but limited set of functions to navigate, inspect and interact with graphical applications [28]. Because of its simplicity when dealing with test automation, motivated research work discussing testing patterns, test harness architectures and even complete test frameworks construction (e.g.: [31], [32]).

Microsoft UI Automation, being released more than ten years after Active Accessibility, draws a lot from that same research—being a much more feature-rich framework, it provides a true abstraction layer over graphical Windows applications. The whole graphical system is represented as a tree, with the desktop at its root, each window as a first-level child and controls on the following levels [29].

While with Active Accessibility all the controls were exposed, UI Automation exposes only the controls that provide some information to the user or can be the target of interactions, leaving behind decorative controls, for example. UI Automation also provides an abstraction over the particular control types that are found in the user interface, via the concept of control patterns. These patterns depend not only on the type of controls—buttons and checkboxes may both support the “Toggle” pattern, but buttons support the “Invoke” pattern while checkboxes don't—, but can also depend on the state of the control—a disabled button does not support the “Invoke” pattern [30].

2.5 Other Technologies

The following technologies were analyzed in the context of exporting a model to a file in a human-readable, textual and mathematical representation.

2.5.1 Microsoft Math 4

Microsoft Math (MS Math) is an application targeted at the classroom: it's most recent version, released in 2006, provides functionalities like graphing, step-by-step math solutions, unit conversion, handwriting recognition and visual edition of equations [45]. It is

precisely this last feature that is most interesting for the project at hand. MS Math not only provides a component for editing equations that can be used in .NET, but it is able to import and export these equations between several formats, including but not limited to MathML [46]. By using the visual editor and plugging into the supported formats, it is possible for an application to re-use the equation editing abilities provided by MS Math.

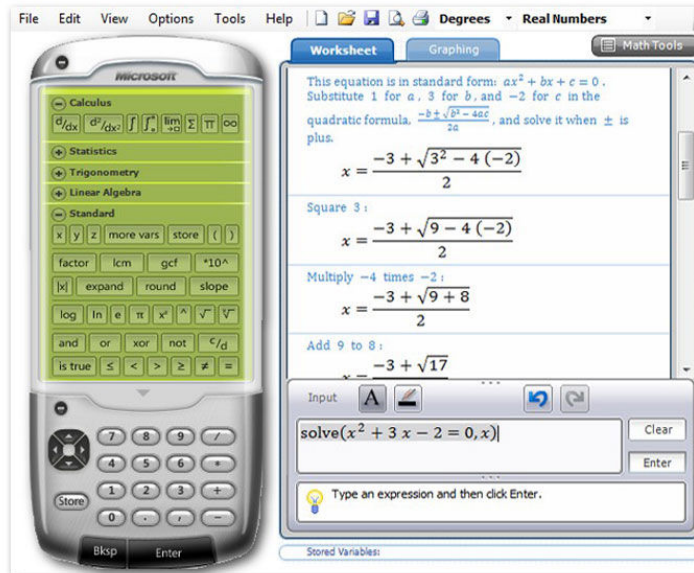


Figure 7: User interface of Microsoft Math 4.

2.5.2 Microsoft Office Primary Interop Assemblies for .NET

The Office 2007 Primary Interop Assemblies (PIAs) put at the disposal of developers the full API of the Microsoft Office programs. However, this very complete object model is also very complex: between methods, properties and events, some interfaces have up to two hundred members.

That complexity offers total control over the document, but the PIAs don't act directly over the physical representation of the documents; instead, they require the target programs to be present in the system (for example, spreadsheets require Microsoft Excel) in order for them to do the work on the documents. This also means that an application that wants to manipulate an Excel spreadsheet requires the additional memory load of loading an Excel instance.

2.5.3 Office Open XML SDK

The Office Open XML (OpenXML) is a free and open Ecma International standard file format specification for representing presentations, spreadsheets, charts and word

processing documents. This format is based on dialects of the XML markup language and it has been adopted as the standard file format for Microsoft Office 2007 [38].

The OpenXML SDK is a .NET library released by Microsoft in 2008 that supports the developer when creating or manipulating OpenXML files programmatically. It creates an abstraction over the packaging and structure of the referred files' structure and contents, by providing an object model for those documents' distinct parts.

Despite being fairly recent (version 1.0 was released in June 2008), this library affords the developer much more simplicity when manipulating Office documents when compared to the Office Interop Assemblies API, at the cost of not supporting completely the OpenXML format. It also has the advantage of not being dependant on Microsoft Office itself—it depends only on the .NET Framework [39].

2.6 Summary

This chapter presented the result of the research and study of relevant subjects, products and technologies related to the problem introduced in Chapter 1. None of the analyzed IDEs covers the two main objectives of this project—support a custom declarative language and being able to work transparently as a part of another software's user interface—so it was clear that there is an opportunity for developing a custom solution for this problem. From this principle, relevant technologies for pursuing that objective were researched and documented as well.

Literature and Technology Review

Chapter 3

Requirements of the ORML IDE

The ORML IDE aims at covering the identified gap between the built-in Dynamics AX business logic customization abilities and the process of creating and maintaining operations research problems modeled in a concise declarative language, by providing an integrated component for managing those models and easily editing and testing them with support from business data and metadata.

The following sections explain in more detail the scope of the project and its interaction points with the ORML Interpreter and Microsoft Dynamics AX.

3.1 Domain-specific terms

Based on what has been presented so far, this section defines terms and concepts specific to the context of this project that will be used in the rest of this document.

Microsoft Dynamics AX: The specific ERP System targeted by this project is Microsoft Dynamics AX, introduced in the previous chapters.

ORML IDE: Henceforth, the Operations Research Modeling Environment shall be referred to as the ORML IDE, or simply the IDE, and is the main topic of the rest of this document.

Models: The models for operations research problems, in their declarative language (ORML) textual format, when used in the context of the IDE (or IDE's interactions with the ORML Interpreter or the Dynamics AX Server), may be referred to simply as “models” when the context is clear enough; otherwise, the term “ORML model” shall be used.

Model parameters: ORML models may specify input parameters—for example, the number of days in advance for which the production must be planned / scheduled.

Model instances: The combination of a model with parameters and the values that will be used as arguments shall from now on be referred to as a “model instance”. A model instance does not contain a model; instead, it references a model

3.2 Assumptions and Constraints

The IDE has to be fully compatible with Dynamics AX 2009. This implies that even if supporting external data sources or standalone mode are not completely ruled out, the main objective is to plan, execute, and test that integration as early as possible in the project. The IDE’s user interface has to be as seamlessly as possible integrated in Dynamics AX’s client user interface and the server settings (for data access and possibly other interactions) should not need to be configured when running inside the client.

The IDE should perform in a way that doesn’t impair user interaction or interferes with the rest of the application (Dynamics AX itself). That being said, it is not a goal to have an optimized performance, which implies that performance tests are not in the scope of the project.

Regarding libraries, tools or other elements that the project will use or depend on, it is preferable to look first and foremost for in-house solutions, to reduce both expenses and external dependencies on products that Microsoft can’t customize or correct if needed. Since Microsoft is a company with more than thirty years of history producing software—being known for its extensive and regularly updated portfolio of platforms, frameworks, libraries and applications—and one of the market principals in many sectors, this constraint is not expected to impair the quality or scope of the solution.

3.3 Project Requirements

In the beginning of the project, three target categories of users were identified:

1. Internal MDCC program managers and developers that make a big effort translating mathematical models of operations research problems to application code (and further efforts when maintaining them);
2. Partners that do customizations of Dynamics AX for clients and need to change the code that solves the referred optimization problems; and
3. Customers that may want themselves to customize their solution or simply to do little adjustments in the model.

Given the short time frame of the project and at-hand availability of the first category of target users (and, by contrast, the difficulty of access and low availability of the target users in the second and third categories), the requirements elicitation process was focused in the first category. This process consisted mainly of individual interviews and workshops with some feature teams (namely, the Inventory and Manufacturing teams).

The other stakeholders on this project are the team designing, developing and supervising the joint project.

3.3.1 Overview

Before going in detail into the requirements of the IDE, a brief overview of the joint project (IDE & Interpreter) must be presented. The joint project aims at providing the complete flow of managing, editing and solving the models with data from Dynamics AX, without ever requiring the users to leave their normal customization environment for Dynamics AX—Dynamics AX itself.

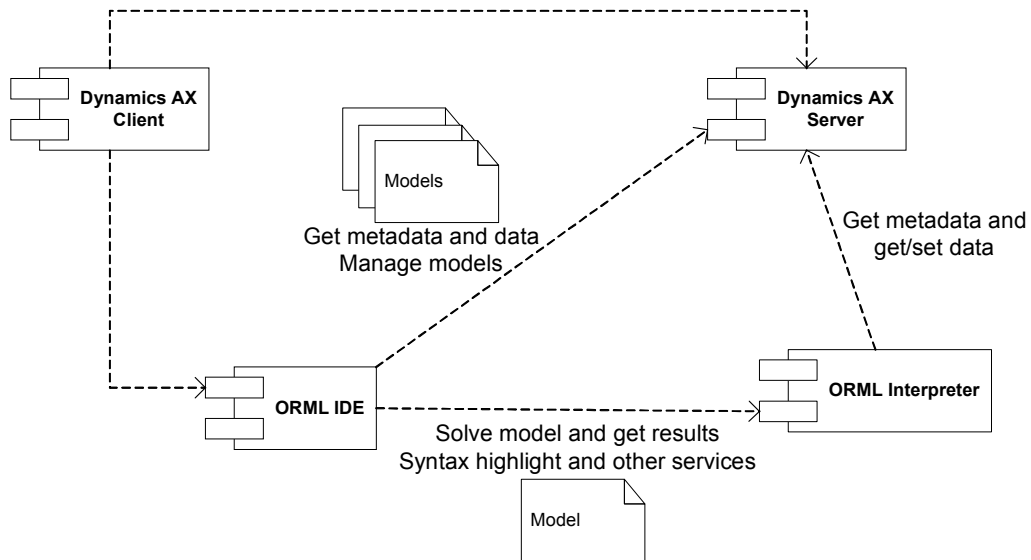


Figure 8: High-level system components diagram.

Figure 8 illustrates a high-level view of the components of the joint project, in regard to dependencies and main reasons for those dependencies. From this figure it's already visible that while there is a clear separation of concerns between the IDE and the Interpreter, there is also some common functionality (getting database metadata from the Dynamics AX server, for example).

3.3.2 Integrated Development Environment

The following use cases were identified for the IDE (see Figure 9). For the purpose of making the work more manageable, they were broken down in smaller units—features. This division into features allowed for common functionality between use cases to be factored out, reducing repeated features early in the design, and helped splitting bigger tasks in work items of some hours (or a couple of days at maximum), so that development could be better planned and executed.

While the list of features derived from the use cases is presented in Appendix B, the use cases identified for the IDE will be presented next.

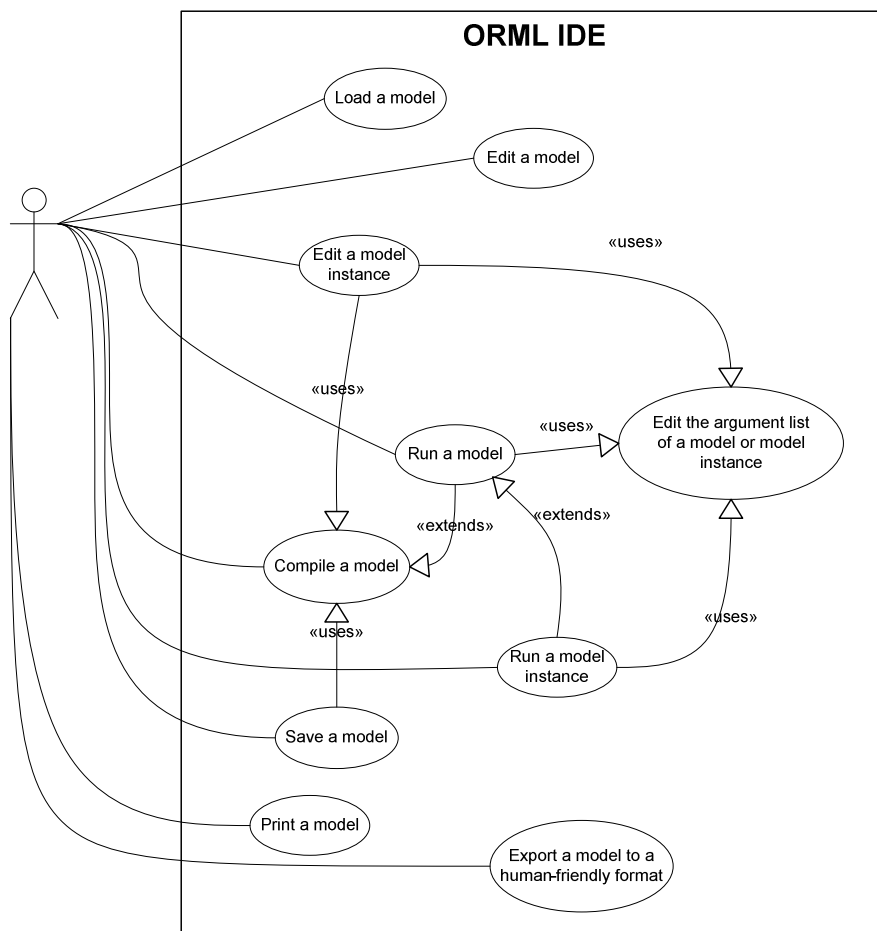


Figure 9: ORML IDE use case diagram.

IDE-UC01: Load a model

It must be possible to load a model into the IDE, either from a pre-defined source folder, from a user-specified one or from the database. The user must select whether the model comes from the database (considered a normal open action) or from the file system

Requirements of the ORML IDE

(considered an import action). The IDE must distinguish between a regular ORML code file and a code+documentation XML file and open both seamlessly.

Goal: Make the model available for editing and other tasks within the application.

Priority: High.

Post-conditions: If the IDE was not able to open the file, the user must be informed of the reason.

IDE-UC02: Edit a model (ORML code)

It should be possible to edit a model's ORML code.

Goal: Allow the user to make any changes to a model.

Priority: High.

Pre-conditions: The model must first be loaded in the code editor.

IDE-UC03: Edit a model (code+documentation)

It should be possible to edit a model in a visual editor that displays the model sections and expressions separately, allowing for visual editing equations and adding arbitrary text between the model's sections and/or expressions.

Goal: Allow the user to make controlled changes to a model, making sure that the model is syntactically and semantically valid.

Priority: Medium.

Pre-conditions: The model must first be loaded in the visual editor.

IDE-UC04: Edit a model instance

It should be possible to edit the arguments of a model instance, saving the instance immediately.

Goal: Save the arguments to make the model ready to be run.

Priority: High.

Post-conditions: If the IDE was not able to save the instance, the user must be informed of the reason.

IDE-UC05: Compile a model

It must be possible to compile the model currently being edited.

Goal: Validate the correctness of the model.

Priority: High.

Pre-conditions: If the ORML code view is out of sync with the code+documentation view, they should be synchronized taking into account the most recently edited view.

Requirements of the ORML IDE

Post-conditions: The user was informed of the results of the compilation.

IDE-UC06: Run a model

It must be possible to run (solve) the model currently being edited or a saved model.

Goal: Solve a model and get a solution, if it's possible to obtain one.

Priority: High.

Pre-conditions: If the ORML code view is out of sync with the code+documentation view, they should be synchronized taking into account the most recently edited view.

Post-conditions: The user was presented with the solution of the model and the interpreter has inserted (or updated, depending on the model) the results in the database, if any of the variables are mapped to the database.

Dependencies: This use case uses IDE-UC08: Edit the argument list of a model or model instance.

IDE-UC07: Run a model instance

It must be possible to run (solve) a saved model with the parameter values specified by the instance's saved argument values.

Goal: Solve a model and get a solution, if it's possible to obtain one.

Priority: High.

Post-conditions: The user was presented with the solution of the model and the interpreter has inserted (or updated, depending on the model) the results in the database, if any of the variables are mapped to the database.

Dependencies: This use case uses IDE-UC08: Edit the argument list of a model or model instance.

IDE-UC08: Edit the argument list of a model or model instance

Before a model that has parameters can be run, the user must input the parameter values. Also, it's important to be able to fix some arguments and save them (with a reference to the model) as a model instance, while still being able to run that instance later with different arguments.

Goal: Have a dialog that allows the user to edit the argument values for running a model or an instance, validating the data types against those specified in the model.

Priority: High.

Triggers: Running a model that has parameters or a model instance; editing the arguments of an instance.

Pre-conditions: The model must compile without errors.

Requirements of the ORML IDE

Post-conditions: The collection of argument values is valid if all the arguments have valid values and the user chooses “Ok” or invalid if the user chooses “Cancel”.

Dependencies: This use case depends on IDE-UC05: Compile a model.

IDE-UC09: Save a model

In order to preserve the changes to the model, the model must be saved.

If the model is being saved over a different model (or an earlier version of this model with a different parameter signature), the user must select whether to replace the previous model (deleting any existing instances) or save the model with a different name.

Goal: Save the new version of the model permanently, either on the filesystem or on the database.

Priority: High.

Dependencies: This use case depends on IDE-UC05: Compile a model.

IDE-UC10: Export a model to a human-friendly format

It should be possible to export a model to a format like MS Word that can be shared and edited, both text and formulas.

Goal: By exporting a model to a document format like Word, it is provided a way for users to publish, discuss and review the model definition.

Priority: Medium.

Post-conditions: If the editor was not able to save the file, the user must be informed of the reason.

IDE-UC11: Print a model

As a complement to exporting a model to a human-friendly format, it is useful to provide the functionality to print a model’s text, with formatting.

Goal: Send a formatted (and paginated, if necessary) document to a printer chosen by the user.

Priority: Medium.

3.3.3 Common Components

3.3.3.1 Common Data Layer

The ORML Common Data Layer was the first common library to be identified in the joint project, very early in the requirements elicitation phase. Since it was believed that it could be useful to support external databases besides the Dynamics AX database, both the IDE and the Interpreter would achieve more easily their respective objectives and the

common objectives by accessing the database data and metadata through a layer of abstraction that didn't involve writing SQL statements or database-specific coding.

Therefore, this library's essential requirements were to provide information about the database schema—tables, views, queries, and their attributes and data types—and to provide read and write access to the different database types that may be considered relevant.

3.3.3.2 *Models Management Library*

The need for this library was identified a bit later, in the first milestone of the project. Its basic requirements were to support the Create/Read/Update/Delete (CRUD) operations over models and model instances stored in the database.

This library was also required to support operations like comparing the parameter signature of two models and solving a model instance (either already existing or newly created).

3.3.4 Non-functional Requirements

The IDE is expected to be always responsive to user interaction and to be resilient, i.e., recover from unforeseen failures derived from its own components or from external factors (unexpected user interaction or network failures, for example).

The common libraries are essential elements for both the IDE and the Interpreter, so they are expected to be robust, reliable and to have a good performance.

3.4 Objectives besides the Requirements

Besides addressing all the requirements of the IDE that the project time would allow (choosing which ones to eventually drop based on the priorities of use cases and features), other objectives were considered on this project:

- Always work concerted with the ORML Interpreter/Language project, keeping a proactive attitude towards communicating and supporting the joint project activities when necessary; and
- Provide not only a good support to the user that is editing a model (in terms of additional information about the model itself or about the data source) but also a good user experience—the IDE should be designed in a way that doesn't require users to “fight” it, i.e., designed to take advantage of the user's habits when using other applications and their knowledge of common Windows applications behavior patterns, and behave in a predictable manner; this may seem to be implied by the usability non-functional requirement, but is stated here because it

was not a requirement directly driven by the users but driven by the author of the project.

3.5 Summary

Based on a more extensive description of the problem being addressed, this chapter presented in details the requirements of the IDE, under the form of a list of use cases that has been updated throughout the project as priorities shifted, together with the goals and required features for the common components of the joint project. The next chapter ties directly into these requirements, describing the solution that was planned.

Requirements of the ORML IDE

Chapter 4

Solution Specification

Based on the requirements for the project and on prototyping work done during the research and technology evaluation phase, a specification of the proposed solution was produced.

Firstly, an overview of the joint project's specification will be presented. It will be followed by the specification of the common components and features implemented in collaboration by the authors of the two projects, ending with a detailed specification of the IDE.

4.1 Overview

4.1.1 Declarative Language

The ORML IDE is targeted specifically at the ORML language, whose definition and development was outside the scope of this project.

The ORML language is a declarative language with data-binding capabilities aiming at having a syntax as simple and straightforward as possible. It's an interpreted language, backed by the ORML Interpreter that makes the actual integration with the solving engine in order to solve the models [41].

The following is an example of a problem modeled in the ORML language.

```

1 Model (String RouteId, String StartLocation, String EndLocation) {
2   Indexes:
3     Locations = Classes.WMSGetRouteItemLocations(RouteId, StartLocation, EndLocation).location;
4
5   Inputs:
6     LocationNum = Count(Locations);
7     Costs = Tables.WMSLocationDistance.Distance[WMSLocationOrigin, WMSLocationDestination];
8 }

```

Solution Specification

```
9 Variables:
10 Integer in [0..1] PathFromLocationToLocation[Locations,Locations];
11 Integer in [1..LocationNum] SequenceLocationVisited[Locations];
12
13 Minimize:
14 Sum(i in Locations;
15     Sum(j in Locations | i != j; Costs[i,j] * PathFromLocationToLocation[i,j])
16 );
17 Constraints:
18 Sum(j in Locations | i != j && Exists(Costs[i,j]) ; PathFromLocationToLocation[i,j]) == 1
19     where(i in Locations | i != EndLocation);
20
21 Sum(i in Locations | i != j && i != EndLocation && Exists(Costs[i,j]) ;
22     PathFromLocationToLocation[i,j]) == 1
23     where (j in Locations | j != StartLocation);
24
25 SequenceLocationVisited[i] - SequenceLocationVisited[j]
26 + LocationNum * PathFromLocationToLocation[i,j] <= (LocationNum - 1)
27     where(i in Locations, j in Locations | j != i && i != StartLocation && j != StartLocation);
28
29 Outputs:
30 Classes.WMSUpdateOrderTrans.sortcode[location,routeId=RouteId] = SequenceLocationVisited;
31 }
```

This model is actually the result of a study conducted by both authors of the joint project in order to try to solve an optimization problem existing in Dynamics AX, namely the Warehouse Picking Routes problem.

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

Line 1 starts by declaring the model parameters and their types. Parameters can be of any of the supported basic types in ORML (Integer, Real or String).

```
Indexes:
```

In line 2 the Indexes section starts. This section contains symbols that represent sets of values over which variables or inputs declared as Vectors will be iterated.

```
Locations = Classes.WMSGetRouteItemLocations(RouteId, StartLocation, EndLocation).location;
```

Line 3 declares an index named Locations whose values correspond to the values of the location column from the records that are read from the database query-like class WMSGetRouteItemLocations. Because Queries in Dynamics AX are very different from stored procedures in regular databases, a wrapper class allows the developer to return data from those queries (or, bottom line, from any data source).

```
Inputs:
```

In line 5 the Inputs section starts. In this section will be declared data elements that are constant during the solving process, but may be variable prior to starting to solve the model, since they can be read from the database.

```
LocationNum = Count(Locations);
```

Line 6 declares a simple scalar called LocationNum that is equal to the number of elements in the Locations index that was created in the Indexes section.

Solution Specification

```
Costs = Tables.WMSLocationDistance.Distance[WMSLocationOrigin, WMSLocationDestination];
```

Line 7 declares a vector called `Costs` whose values are read from the `Distance` column of the `WMSLocationDistance` database table. This vector is indexed by the `WMSLocationOrigin` and `WMSLocationDestination` fields of the same table, which means that in run-time a query will be generated that groups the rows of that table by those three fields, getting the distinct rows and filling the vector.

```
Variables:
```

In line 9 the `Variables` section starts. This section is where the decision variables of the model are declared.

```
Integer in [0..1] PathFromLocationToLocation[Locations,Locations];
```

Line 10 declares a variable called `PathFromLocationToLocation` that is a vector of `Integers` doubly indexed by the `Locations` index previously declared. The domain of each element is between 0 and 1 inclusive, which in an integer space effectively means that each element will be 0 or 1. This variable can be compared to a matrix where lines and columns both represent locations and the cell value is 1 or 0 depending on whether the solution requires the path from one location to another to be taken or not, respectively.

```
Integer in [1..LocationNum] SequenceLocationVisited[Locations];
```

Line 11 also declares a vector of `Integers`, this time with only one index (`Locations` again) and a different domain: the `LocationNum` input that was defined before. This variable represents the sequence by which the locations are visited.

```
Minimize:
```

Line 13 starts a `Goal Function` section—in this case, the goal is to minimize only one expression, but it is possible that a model specifies several expressions to be minimized and maximized simultaneously.

```
Sum(i in Locations;
    Sum(j in Locations | i != j; Costs[i,j] * PathFromLocationToLocation[i,j])
);
```

Lines 14 to 16 constitute the expression that will be minimized, introducing the `Sum` function. The clearest explanation of the expression is the following mathematical expression:

$$\sum_{i \in \text{Locations}} \sum_{j \in \text{Locations}} Costs_{i,j} \times PathFromLocationToLocation_{i,j}, i \neq j$$

This expression translates to the goal of minimizing the sum of the costs of each path taken, or, in other words, minimizes the travelled distance.

Solution Specification

Constraints:

In line 17 the Constraints section starts. While the Goal Function sections were only arithmetic expressions, expressions in this section are equations (either equalities or inequalities).

```
Sum(j in Locations | i != j && Exists(Costs[i,j]) ; PathFromLocationToLocation[i,j]) == 1
where(i in Locations | i != EndLocation);
```

Lines 18 and 19 constrain the solver to go from one location to exactly only one other location, except for the final location, that is the end of the route. This constraint introduces two other language elements:

- The Exists function returns true if a Vector contains a value indexed by the given indexes—this is useful in cases where not all the paths between some locations exist, originating a sparse adjacency matrix;
- The where keyword is intended to be read as in a mathematical formula—the Sum line alone contained the undefined variable i, that the where clause specifies as being each of the values contained in the Locations index, except EndLocation.

```
Sum(i in Locations | i != j && i != EndLocation && Exists(Costs[i,j]) ;
PathFromLocationToLocation[i,j]) == 1
where (j in Locations | j != StartLocation);
```

Lines 21 to 23 don't contain any new concepts. This constraint is similar to the previous one, but specifies that each location must be visited exactly from one other location.

```
SequenceLocationVisited[i] - SequenceLocationVisited[j]
+ LocationNum * PathFromLocationToLocation[i,j] <= (LocationNum - 1)
where(i in Locations, j in Locations | j != i && i != StartLocation && j != StartLocation);
```

Despite not introducing any new concept, lines 25 to 27 show a little more of the flexibility offered by the ORML language. This constraint instructs the solver to avoid loops while searching for a solution.

Outputs:

Line 29 starts a new section: the Outputs of the model. In this section, decision variables can be bound to database tables or queries in order to insert or update data to the database.

```
Classes.WMSUpdateOrderTrans.sortcode[location,routeId=RouteId] = SequenceLocationVisited ;
```

Finally, line 30 specifies that the values of the SequenceLocationVisited Vector shall be used as parameters for invoking the WMSUpdateOrderTrans query-like class, once for each location that indexes the Vector. The values of the vector itself are mapped to the parameter called sortcode. The class needs one additional parameter called routeId that is fetched directly from the RouteId model parameter.

```
}
```

This concludes the introduction to the ORML language. For more details, the language full specification is defined in [41].

4.1.2 Complete System Architecture Overview

Since the beginning of both projects, planning has been a concerted activity, in order to start integration of both projects as early as possible. Starting with the ORML language requirements and based on the IDE and Interpreter requirements, the activities that require interaction between IDE and Interpreter were identified, together with the data that is exchanged in those interactions.

4.1.2.1 Workflows and data flows

The basic workflow of the joint project is to allow the user to write a model in ORML, get feedback about possible errors in the model, solve the model and get information about the solution. For all these actions, the IDE supports itself on the Interpreter.

Because both the IDE and the Interpreter need to access the database, the Interpreter can receive a reference to an instance of the `DatabaseAdapter` used by the IDE, so that only one database session is used for both.

For the purpose of checking a model for errors and solving it, the model text is passed to the Interpreter. The Interpreter signals output lines and errors through events that are subscribed by the IDE. These events also indicate if the model has valid syntax and semantics.

To support syntax highlighting, the Interpreter has a specific method that given the text of a model outputs a list of tokens associated with their types (identifiers, keywords, punctuation, etc.) and positions (stating line and column, ending line and column).

In order to display an outline view of the model, the parse tree that the Interpreter returns is traversed by a visitor class that creates the tree items displayed to the user.

4.1.2.2 Components and Technologies

In light of the four top-level components that have been identified so far—IDE, Interpreter, Common Data Layer and Models Management Library—the block diagram in Figure 10 illustrates the technology stack envisioned for the joint project (excluding implicit .NET Framework elements upon which IDE, Interpreter and shared libraries all depend), while Figure 11 provides a view more oriented at components and dependencies between them.

Solution Specification

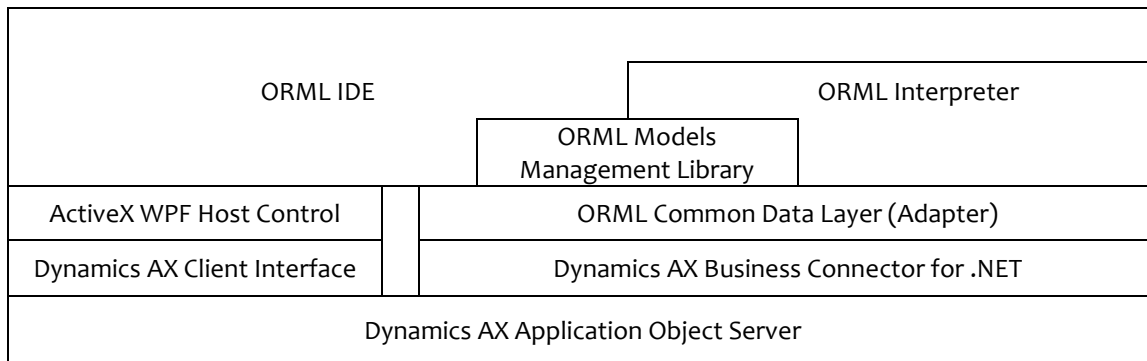


Figure 10: Blocks diagram illustrating the technology stack of the joint project.

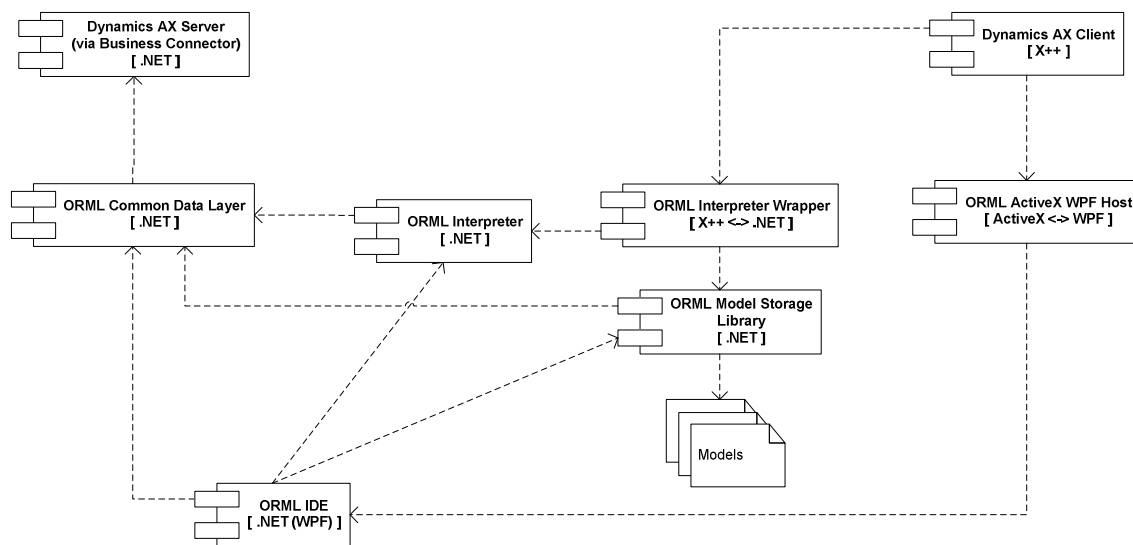


Figure 11: Components diagram of the joint project.

4.2 ActiveX WPF Host

This control is the link between the IDE and the Dynamics AX Client. The `ActivexWpfHost` control inherits from `ElementHost`, a Windows Forms control released as part of the .NET Framework 3.0 that can be used to host a WPF element [44] (`UIElement` is the base class for all the controls and many of the visual elements in WPF). Since the `ElementHost` class cannot be changed, `ActivexWpfHost` can be considered to be a decorator over `ElementHost`—it provides all the functionality of the base element and additional properties and operations.

Solution Specification

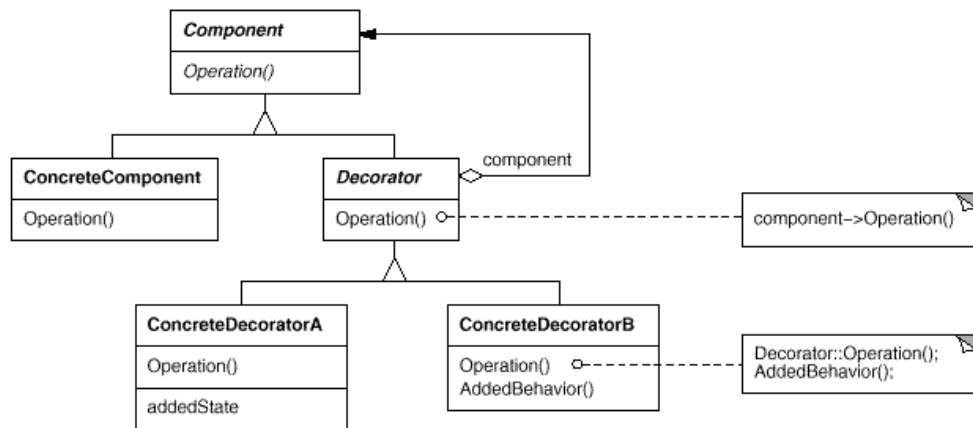


Figure 12: The Decorator design pattern conceptual diagram [44].

The referred additional properties and operations are related to the specific purpose of this control: besides hosting a WPF control in Dynamics AX, it provides contextual information about the connection between client and server to the controls that may be interested in knowing it, i.e., to controls that implement the `IManagedAxControl` interface.

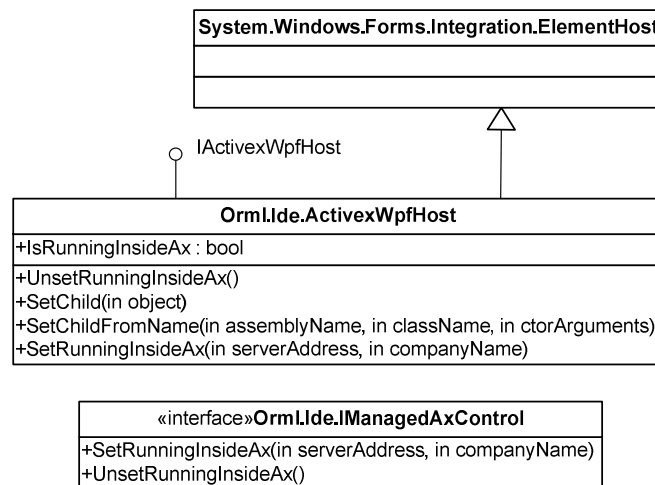


Figure 13: Class diagram of the ORML ActiveX WPF Host.

4.3 ORML IDE

From the concept of the joint project, the ORML IDE was planned to be tightly coupled with the ORML language and with Microsoft Dynamics AX, consequentially being tightly coupled with the ORML Interpreter and Common Data Layer.

That being said, the design of the IDE also took into account that the IDE should not be a completely closed application, but should be designed to support extensibility. As presented in Chapter 2, extensibility is a more and more valued trait of modern IDEs—even

if the time available for the planning and execution of the ORML IDE does not allow incorporating extensibility in the form of plug-ins or scripts, for example, the IDE should at least be designed to be extensible.

The technological base and architecture of the ORML IDE will be presented in the following sub-sections.

4.3.1 Technologies

This project relies heavily on the Microsoft .NET Framework 3.5—from the joint project common libraries to the IDE user interface, all the components were planned to be built upon the .NET Framework (or using .NET 2.0 components, in the case of Microsoft Math).

The .NET Framework 3.5 is not a replacement of versions 3.0 or 2.0, but a superset of them, supporting all the core features of .NET 2.0 — like WinForms, serialization, reflection, Web services, among others — and .NET 3.0 — Windows Cardspace, Windows Workflow Foundation (WF), Windows Communication Foundation (WCF) and Windows Presentation Foundation (WPF)—, adding some new technologies like ASP.NET Asynchronous JavaScript and XML (ASP.NET AJAX) and Language Integrated Query (LINQ).

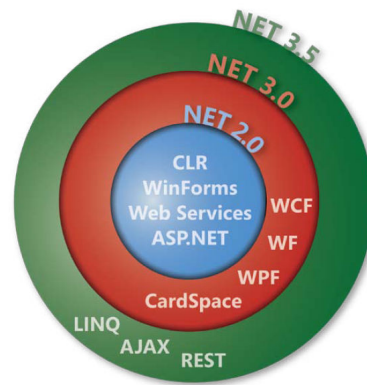


Figure 14: The .NET Framework 3.5 technology stack [52].

NET 3.5 also introduced version 3.0 of the C# programming language, with support for anonymous types (static, inferred at compile-time), extension methods (that allow a method to be added to a class just by importing the namespace where the extension was defined) and taking functional programming aspects of C# a little further with lambda expressions and expression trees. More importantly, it's binary compatible with C# 2.0, which means that it runs on the same version of the Common Language Infrastructure (CLI).

On the visual side, the combination of XAML and the WPF libraries is very strong: XAML is essentially a very sophisticated serialization form of the object model defined by WPF that allows a very flexible composition of visual objects, comparable to HTML. Events in WPF are also similar to events in HTML: until handled, they bubble from the element that originated the event up to the root of the visual tree—these are called bubbling events. WPF also supports tunneling events that generally are the counterpart to bubbling

events and follow the reverse path, i.e., from the topmost element in the tree to the element that originated the event (e.g.: bubbling event `KeyDown` is preceded by tunneling event `PreviewKeyDown` that can cancel the former or modify its arguments) and direct events (targeted at a specific control and none other in the visual tree). These kinds of events are WPF routed events.

4.3.2 User Interface

The ORML IDE's user interface was not something that can be directly deduced only from the use cases and other requirements—its characteristics and organization draw much from the analysis of other IDEs for modeling and programming languages.

One of the essential threads in modern IDEs is the ability to work with multiple documents simultaneously, presenting those documents either in internal dialogs (applications that follow this approach are known as Multiple Document Interface applications, or simply MDI applications) or as tabs of a central tab control. The ORML IDE follows the latter strategy.

Using a multiple tabs interface (MTI) to show multiple documents in an application is a more recent trend that has been adopted by IDEs and Web browsers, so it is expected to be a user interface pattern that users of the IDE will recognize and start using immediately, virtually without any learning effort.

A characteristic present both in MDI and MTI applications that the ORML IDE was also planned to have is the clear separation between areas in the editor that serve different purposes—the database explorer, the documents area, the model outline of each document and other areas should be easily distinguishable one from the other.

However, the flexibility of customization of those areas (something that is common place in other IDEs) is not present in the ORML IDE. This does not mean that the IDE couldn't be made to have a very flexible UI—in fact, doing so wouldn't be very hard, taking into account the capabilities of the underlying technology that were referred in the previous section. This decision was based on the notion that many tools and applications benefit from having a consistent look and behavior across an organization, allowing the users to collaborate and interact easily.

This consistency was also observed with regard to application visuals and terminology used: since one of the main targets of the ORML IDE was to be as seamlessly as possible integrated into the Dynamics AX client interface, the icons for toolbar, database explorer and models explorer, for example, are from Dynamics AX itself, used in similar situations. Other icons in the IDE are used in the same sense they would be used by other applications

Solution Specification

(e.g.: error list, output and run are very similar to Microsoft Visual Studio's). An example of adapting the IDE terminology to the “user environment” where it is executing is the text that appears on the topmost item of the database explorer: when running in standalone mode it displays “Database: X”, while when running inside Dynamics AX it displays “Company: X”.

Another trait very common that was adopted as a design goal by the IDE is displaying visual cues as to whether or not a functionality activated by a button or menu item is available, in order to spare the user unnecessary dialog messages or “empty” clicks, i.e., clicking a button and “watching” the button get clicked normally without anything happening.

When planning the UI some sketches were made, depicted in Figure 15, Figure 16 and Figure 17.

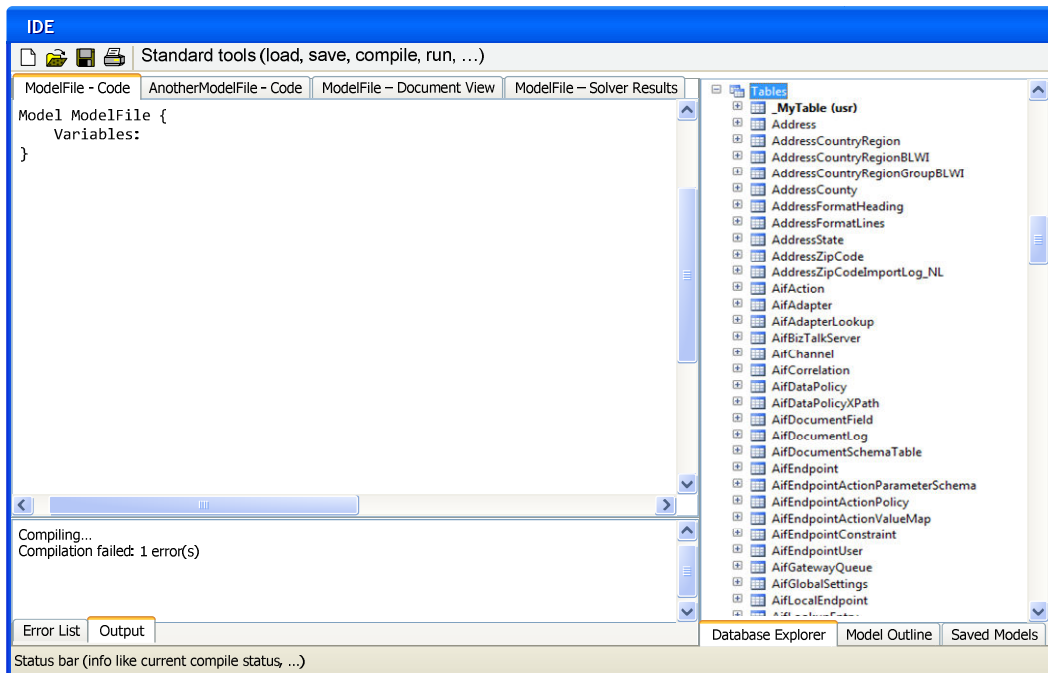


Figure 15: Early IDE UI sketch.

Solution Specification

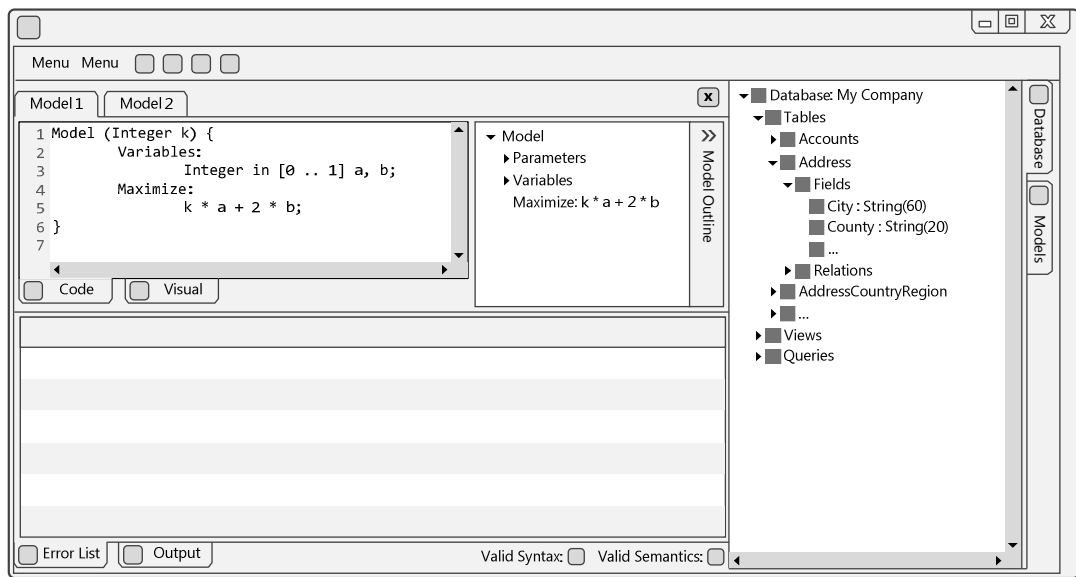


Figure 16: Abstract UI sketch for the IDE (showing the code editor, database explorer and model error list tabs).

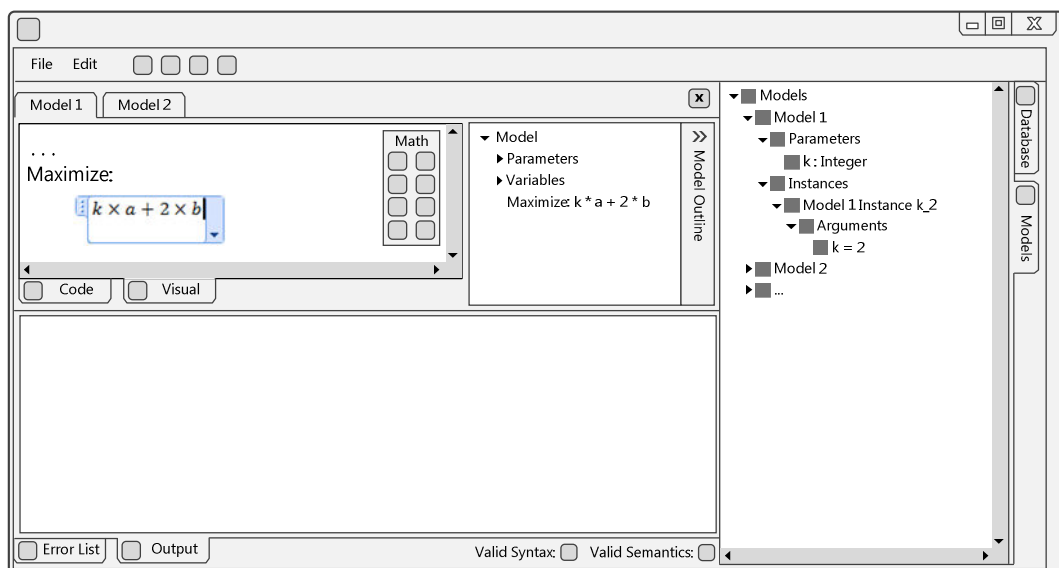


Figure 17: Abstract UI sketch for the IDE (showing the visual editor, models explorer and model output tabs).

4.3.3 Architecture

The initially planned high-level architecture for the IDE is depicted in Figure 18. The components identified in the diagram follow the organization of the visual controls identified early on (see Figure 15). While the components themselves were kept, some changes in the UI organization had repercussions in the way these controls were nested. For

Solution Specification

example, the Output and Errors panels and the Model Hierarchical Tree (or simply, Model Outline) were moved to be part of the Editor Area (henceforth referred to as Model Form).

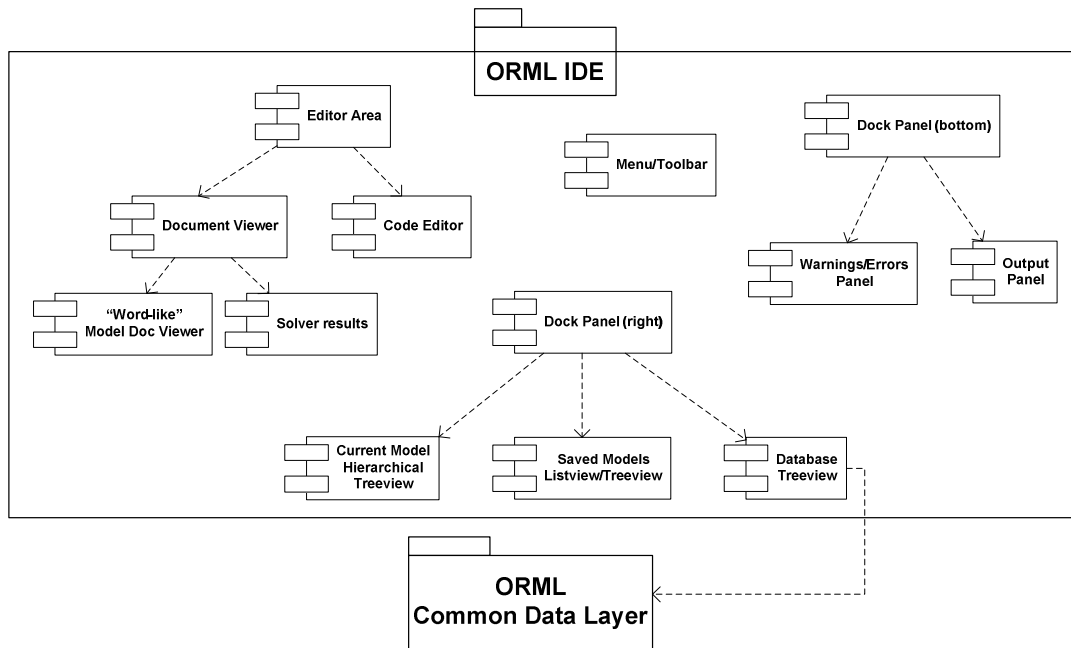


Figure 18: Initial version of the ORML IDE components architecture.

An updated version of the IDE architecture can be seen on Figure 19. This diagram already contains references to the Interpreter, to the Models Management Library and to Microsoft Math, but does not present possible dependencies between those packages.

Solution Specification

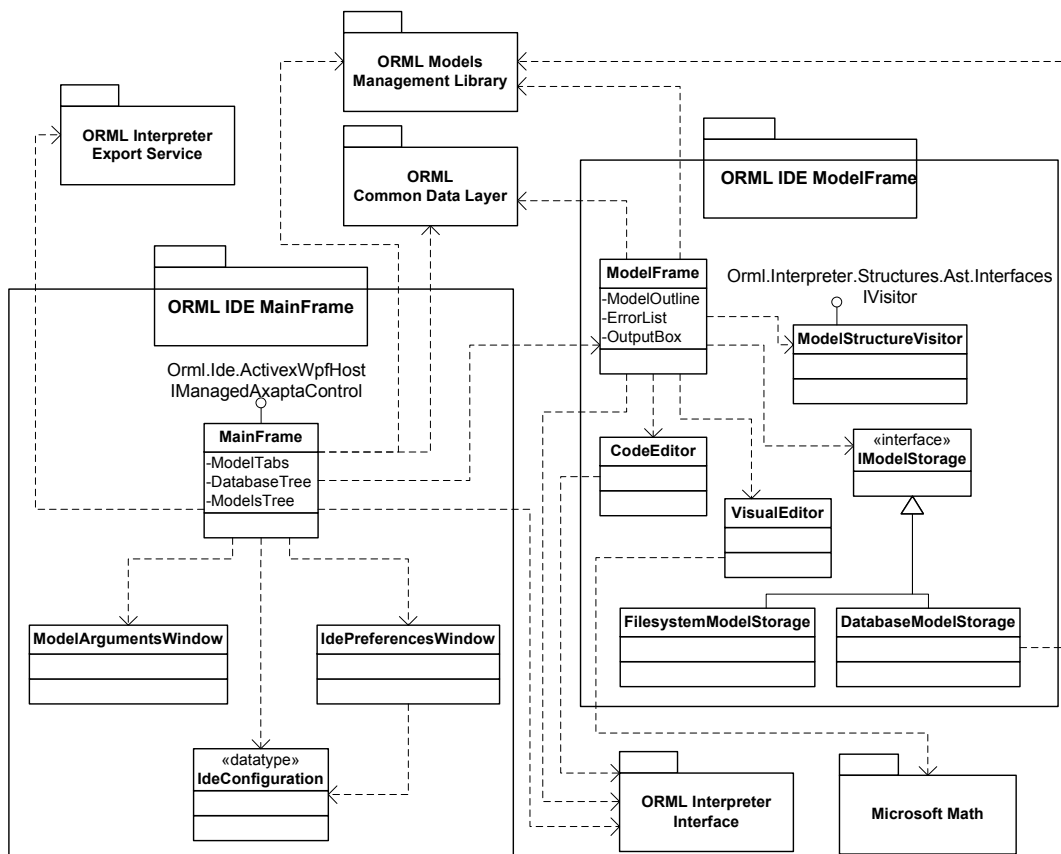


Figure 19: The architecture of the ORML IDE—class diagram.

The following sections expose the architectural details of the IDE grouped by major components, starting with the component that has the least dependencies (the Code Editor) and finishing with the one that has the most dependencies (the Main Frame).

4.3.3.1 Code Editor

The `CodeEditor` class is composed of a simple `TextBox` for displaying the line numbers side by side with a `RichTextBox` for allowing the user to edit the model text. The `RichTextBox` formatting features are used to provide syntax highlighting of the model text, being dependant on an `Interpreter` service specifically designed for this.

The syntax highlighting will support itself on a `TokenStyleProvider`, which associates a `TokenStyleInfo` object to the corresponding `TokenCategory` (the latter is an enumeration defined by the `Interpreter` library, which groups tokens in “punctuation”, “identifier”, “keyword” and other types). So, the list of tokens received from the `Interpreter` that contains information about each token’s position and type is complemented with information about the style that must be applied to that token.

Solution Specification

The `TokenStyleInfo` structure (depicted on Figure 20) declares the most common types of formatting but also supports a collection of custom text decorations. These custom decorations can, for example: change the color of the text background; create a custom underline with a different color or an image; or use a different font for the text being decorated. Effectively, any formatting available to the `RichTextBox` can potentially be specified in the custom decorations.

TokenStyleInfo
+Color
+IsBold
+IsItalic
+IsUnderlined
+CustomDecorations

Figure 20: The `TokenStyleInfo` structure.

4.3.3.2 Visual Editor

The visual editor allows the user to abstract him or her from the modeling language and actually visualize and edit the model expressions. It will rely on the equation editor from Microsoft Math (MS Math) to display and edit the equations, requiring some level of translation between the representations of equations in MS Math to their respective representations in ORML. A sketch for the possible UI of the visual editor is presented on Figure 21.

The UI sketch shows a control titled "Model". Inside, there are two main sections: "Inputs:" and "Minimize:". The "Inputs:" section contains a complex expression: $\text{Costs} = \text{Tables} \cdot \text{Paths} \cdot \text{Distance} \left[\text{From}, \text{To}, + \right]$. The "Minimize:" section contains a summation expression: $\sum_{i \in \text{Locations}} \text{Costs}_{ij} \times \text{PathTaken}_{ij}$, followed by a constraint: $j \in \text{Locations}$. Both sections have expand/collapse icons (+/-) and a vertical scrollbar on the right.

Figure 21: UI sketch of the Visual Editor control.

4.3.3.3 Model Frame

The `ModelFrame` control will be responsible for displaying the editing controls referred in the previous sections, along with other controls that provide additional information about the model being edited (and solved). This control must “know” how to perform any necessary conversions between the formats used by the editing controls to represent the models, in order to keep them synchronized.

The `ModelFrame` control will also be responsible for managing loading and saving models from various storage sources, signaling events like changes to the model so that

Solution Specification

eventual visual cues may be displayed by the container control (in the IDE's case, the container of `ModelFrame` instances will be a `TabControl` in a `MainFrame` instance).

One of the additional controls contained in the `ModelFrame` is a `TabControl` that contains one tab with the list of errors in the model and another tab with a simple `TextBox` for displaying the output of the Interpreter when solving a model. The error list is populated based on the errors raised by the Interpreter when checking the validity (what can be called “compiling” a model) or interpreting a model.

Finally, `ModelFrame` will also contain a `TreeView` control that provides an outline perspective of the model, divided by sections and with each expression identified. This model outline shall be constructed based on the model tree returned by the Interpreter's parser (specifically, an Abstract Syntax Tree, or AST).

4.3.3.4 Main Frame

The `MainFrame` control will be the IDE itself. Its first point of integration with Dynamics AX is through the ActiveX WPF Host control. The `MainFrame` implements the `IManagedAxControl` interface, in order to distinguish running in standalone mode from running as a control hosted in Dynamics AX.

This control's main component is a `TabControl` where instances of the `ModelForm` control are displayed. While responsibilities like loading and saving models are delegated to the `ModelForm`, it's the `MainForm` that controls nearly all the user interactions. The `MainForm` control handles application commands (like opening a model) and some commands specific to the current model being edited (like solving or printing a model, or closing the current model being edited), but the actual execution of the actions associated with those commands may be delegated to the corresponding `ModelForm` instance. For example, saving a model will be delegated to `ModelForm`, but solving a model is handled by `MainForm`, because it may require prompting the user for argument values to pass to the Interpreter.

The `MainFrame` control will also contain a `TreeView` control to display the models currently saved in the database, together with their parameters and any existing instances; its structure and planned context menus are documented in Appendix E. Interaction with this control will allow the users to solve a model or a model instance from the IDE without the need to actually open the model for editing.

Finally, the `MainFrame` control will contain another `TreeView` to allow the user to navigate through the database elements and will support interaction between them and the model being currently edited—for example, the user may select a table field that is inserted in the text of the model.

4.3.4 Testing Methodology

Whether planning automated or manual testing of the IDE, the general approach will be, for each feature or use case to test:

1. Document the valid states of the IDE when the feature or use case is being exercised;
2. Document what actions are necessary to trigger the state transitions;
3. Verify that when those actions are taken, the state transitions are successful.

Regarding automated testing, it's required of the IDE to behave almost completely the same way when instantiated as a control inside Dynamics AX or used in standalone mode. Based on this—and knowing that instantiation inside Dynamics AX can be simulated in an external environment—all automated testing will occur in the .NET 3.5 environment.

This allows for easy verification of the application state before and after each automated test is executed, because all the properties of the .NET controls are immediately accessible.

The results of the tests will be automatically documented in the following format:

Feature name	Test name	Passed (Y/N)	Initial state	Expected result	Actual result
Standalone IDE Initialization	Initialize From Configuration File	N	CurrentDatabase == null; CurrentServer == null; Configuration == null;	CurrentDatabase != null; CurrentServer != null; Configuration != null;	CurrentDatabase != null; CurrentServer != null; Configuration == null;
...					

Figure 22: Documentation example for the automated tests.

Manual testing will also follow the “cover state-transitions” pattern, using the IDE in standalone mode or inside Dynamics AX as required for the test (or set of tests) at hand.

The results of the tests should be documented in the following format:

Feature name	Test name	Passed (Y/N)	Initial state	Expected result	Actual result
Standalone IDE Initialization	Initialize From Clean Configuration Or Without Previous Configuration	Y	User is asked to input server name and company name	Application connects to the AOS and loads DB tree and saved models	Application connects to the AOS and loads DB tree and saved models
...					

Figure 23: Documentation example for the manual tests.

Besides the new .NET 3.5 UI Automation framework (for the automated tests), no tools or libraries will be necessary to conduct / run the tests. To document the results of the tests, it will be necessary either the access to the Microsoft Office Excel API or to the Open XML Format library.

4.4 Common Components Architecture

The common components were planned by both authors of the joint project—the ORML IDE project and the ORML Language and Interpreter project.

4.4.1 Common Data Layer Library

As mentioned in the previous chapter, this library was the first common library to be planned in the joint project. In order to fulfill its requirements of providing a good abstraction over the underlying database and still be as database-agnostic as possible, it implements the Adapter design pattern [44]. This simple design pattern, abstractly presented in Figure 24, was instantiated as it can be seen in Figure 25.

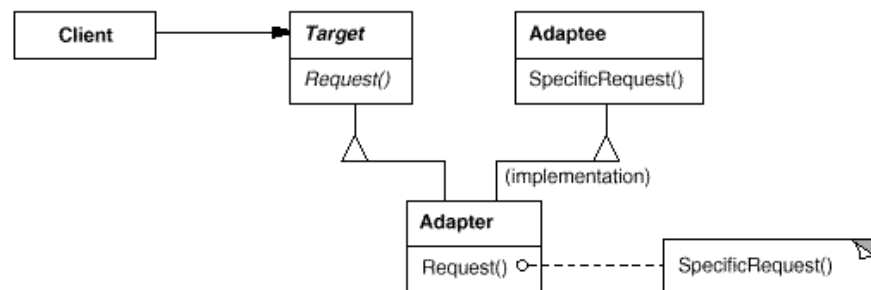


Figure 24: Adapter design pattern conceptual diagram [44].

By using this design pattern, it is guaranteed that the applications that use this library are prepared to use any type of database that implements the adapter to the program or library that is being adapted (the adaptee).

This library also provides a collection of related classes that represent database data and metadata—information about tables, their fields and field types, for example. A more detailed view over this object model is presented in Figure 25. In the case of queries, the fields are used to represent the parameters of the query.

Solution Specification

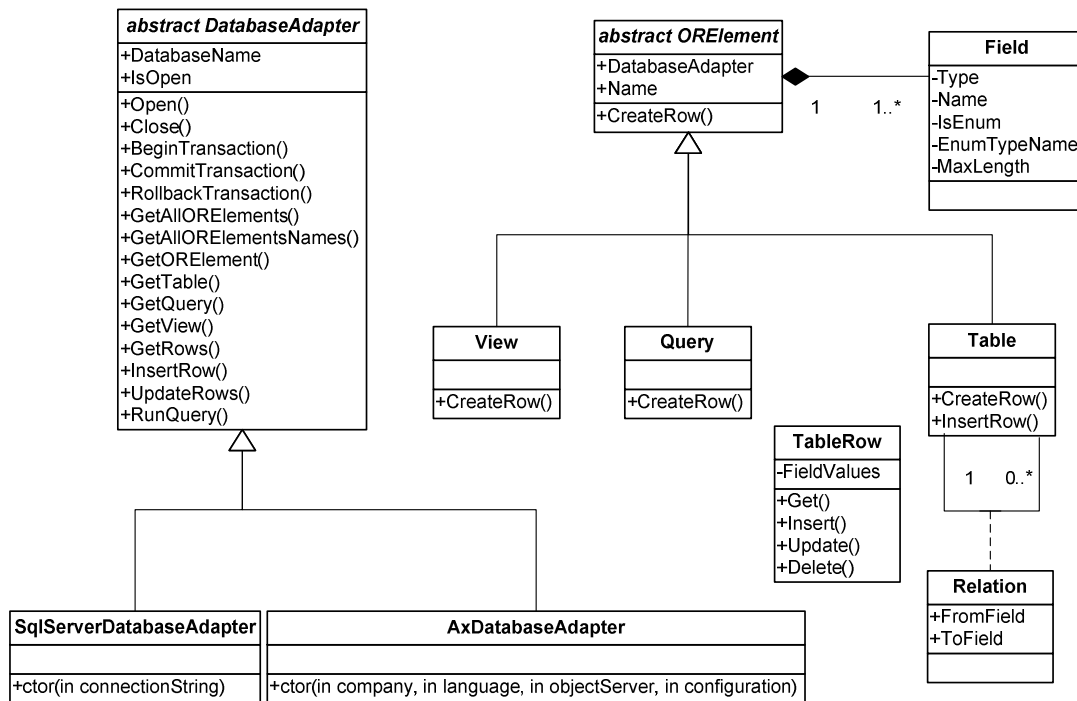


Figure 25: Commn Data Layer adapter structure and support classes.

4.4.2 Models Management Library

Dynamics AX follows the client-server architecture and there is the interest of making models available organization-wide. For this purpose, the models are stored in the database.

Since ORML models are typically not very large, this decision is not expected to negatively impact the system performance or back-up and restore operations, yielding the benefits of possibly versioning the models (either internally or making use of Dynamics AX infrastructure) and making the models readily available to everyone that needs to use them. This does not mean only the users who have to create and maintain models, but, more importantly, empowers the developers to use models as part of the business logic without the users ever realizing it.

The Models Management Library was designed to depend on the ORML Common Data Layer, which in turn makes this library effectively database-independent. This library also depends on the ORML Interpreter Interface for parsing the models in order to retrieve the list of parameters of the models and their types (in sum, the parameter signature of the model). This is necessary in order to support operations like saving a model replacing another existing model—if the models have different parameter signatures, the user must choose whether to overwrite the existing model and any existing instances or saving the model under a different name.

Solution Specification

This dependency on the Interpreter was explored further: the model instances have a method to solve them. Such a small utility method proves its usefulness making trivial the operations of solving models and instances:

- Solving a saved model is as simple as fetching it from the database, creating an instance, setting the arguments values and invoking `Run()`;
- Solving a saved model instance is as simple as fetching the model, fetching the required instance from that model and invoking `Run()`.

Invoking the `ModelInstance.Run()` method returns an `InterpreterResult` object that can be used to provide the user with feedback received from the Interpreter regarding the solution of the model, any errors detected and other output from the Interpreter.

Figure 26 illustrates in detail the planned architecture for the Models Management Library.

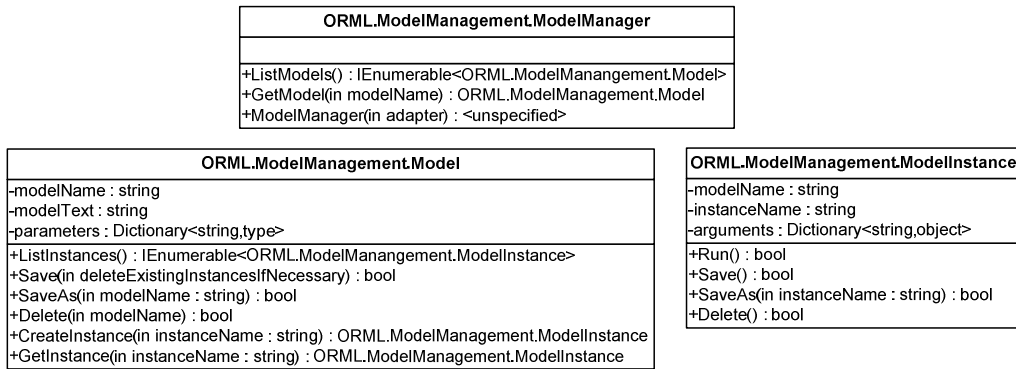


Figure 26: Models Management Library classes.

4.5 Design of Other Features

This section presents noteworthy design and architecture details regarding features that were partially planned by the author of the IDE project in the joint project.

4.5.1 Convert ORML Model to Word Document

The most recent Microsoft Word document file format is based on XML dialects for representing the various parts of the document, including equations. The format for equations, OMath, is similar to MathML, but has some simplifications and includes constructs to support word-processing documents specific features like comments and revisions. However, Microsoft Office 2007 ships with XSL documents that provide the conversion from OMath to MathML and vice-versa.

Solution Specification

From that standpoint, it's possible to traverse the syntax tree of an ORML model, build an MS Math internal expression tree and get the corresponding MathML that is then converted to OMath and included in a word document.

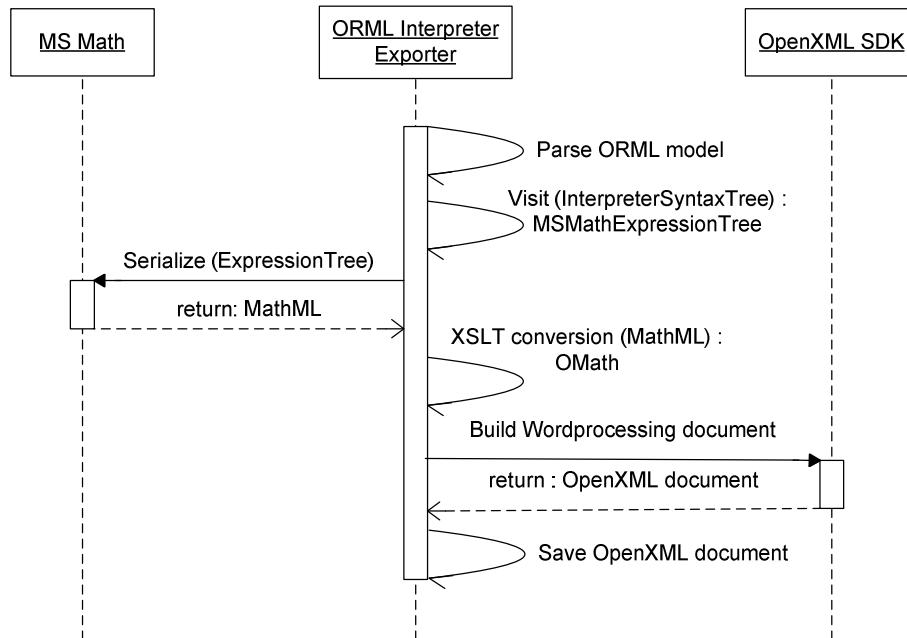


Figure 27: Conversion from ORML to Word (OpenXML) sequence diagram.

Figure 27 shows the sequence of converting an ORML model to a Word document. It's expected that each expression of a model (i.e., each goal function and constraint) will have to be converted individually to MathML, resulting in a cycle of calls to MS Math and conversions to OMath and doing a final aggregation of those expressions in a Word document.

The implementation of this feature was almost completely done by the author of the Interpreter project.

4.6 Schedule

The schedule of the joint project and of the IDE project is now presented, even if the design work documented in this section was not all done before the project schedule was established.

Solution Specification

4.6.1 Joint Project Overview

A schedule of the project, including the planned activities for writing this dissertation and other documentation about the project, is depicted on Figure 28.

The project research and development phases were planned to run over the course of two milestones:

1. The first milestone's objective was to build a vertical prototype with data-binding capabilities and minimal integration with the Dynamics AX client interface; This required doing research in the context of the project and building prototypes to assess feasibility of the features;
2. The second milestone involved not only finishing the planned features for the IDE and the Interpreter but also studying and solving optimization problems existing in Dynamics AX using the developed platform.

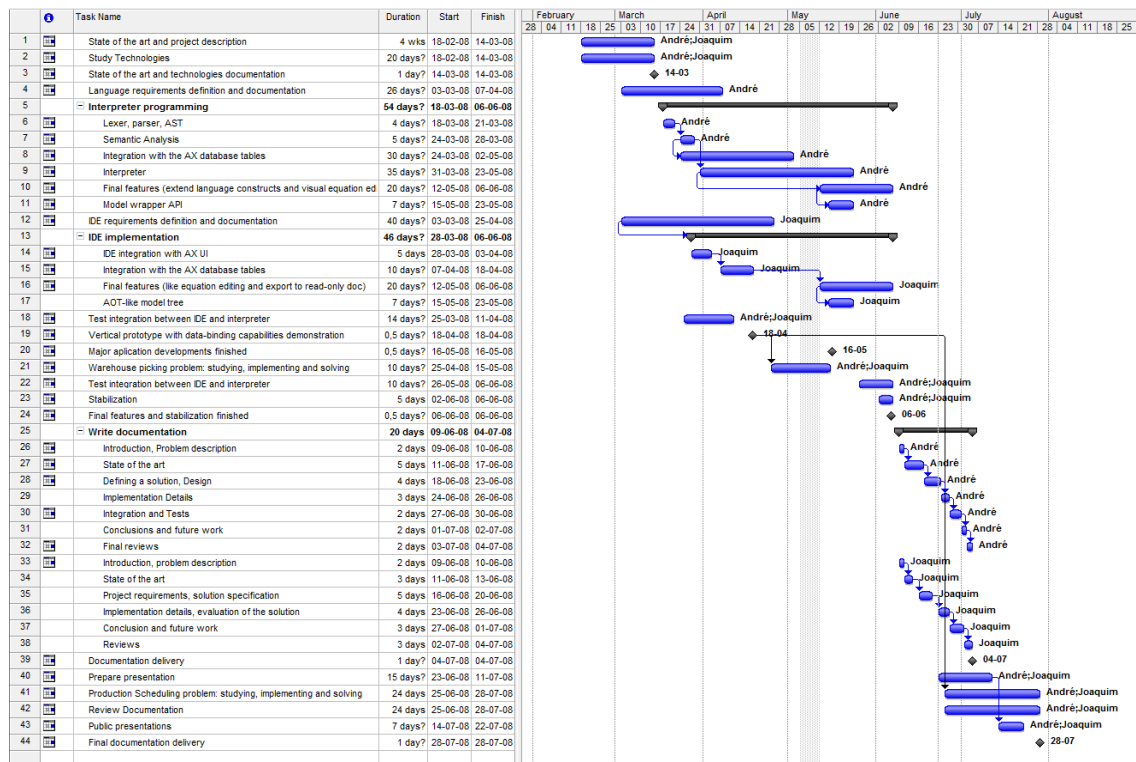


Figure 28: Schedule of the joint project.

4.6.2 ORML IDE Project Details

The schedule for the IDE project is not more detailed than the schedule for the joint project, in terms of planned tasks and their duration. Not all of the activities presented in Figure 29 (the same version of the Gantt chart in Figure 28 with only the IDE project's activities) correspond to the use cases or features of the project—they were estimated and

Solution Specification

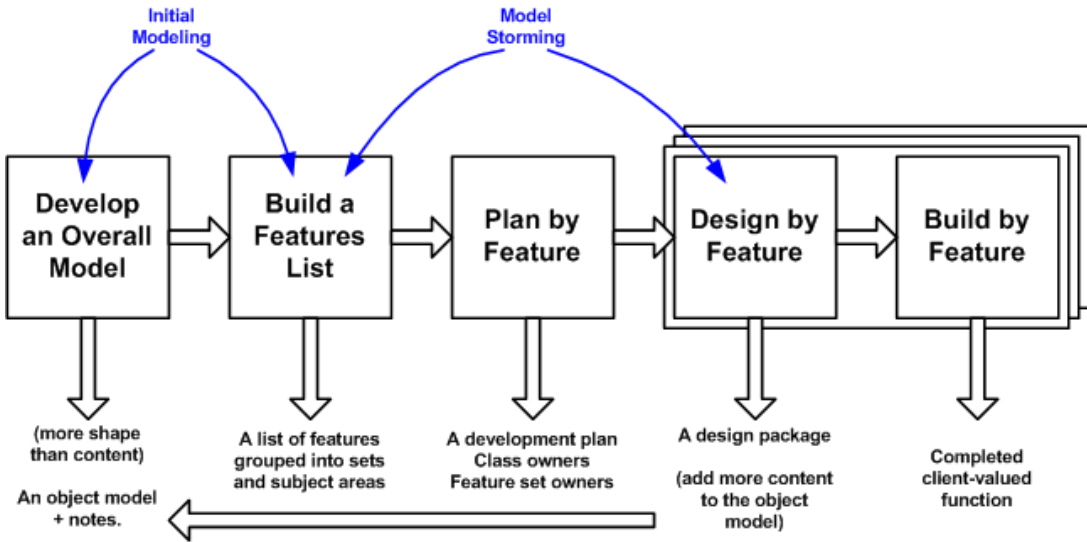
planned with the purpose of compartmentalizing and synchronizing the objectives of the joint project.

		Task Name	Duration	Start	Finish
1		State of the art and project description	4 wks	18-02-08	14-03-08
2		Study Technologies	20 days?	18-02-08	14-03-08
3		State of the art and technologies documentation	1 day?	14-03-08	14-03-08
12		IDE requirements definition and documentation	40 days?	03-03-08	25-04-08
13		IDE implementation	46 days?	28-03-08	06-06-08
14		IDE integration with AX UI	5 days	28-03-08	03-04-08
15		Integration with the AX database tables	10 days?	07-04-08	18-04-08
16		Final features (like equation editing and export to read-only doc)	20 days?	12-05-08	06-06-08
17		AOT-like model tree	7 days?	15-05-08	23-05-08
18		Test integration between IDE and interpreter	14 days?	25-03-08	11-04-08
19		Vertical prototype with data-binding capabilities demonstration	0,5 days?	18-04-08	18-04-08
20		Major application developments finished	0,5 days?	16-05-08	16-05-08
21		Warehouse picking problem: studying, implementing and solving	10 days?	25-04-08	15-05-08
22		Test integration between IDE and interpreter	10 days?	26-05-08	06-06-08
23		Stabilization	5 days	02-06-08	06-06-08
24		Final features and stabilization finished	0,5 days?	06-06-08	06-06-08
25		Write documentation	20 days	09-06-08	04-07-08
33		Introduction, problem description	2 days	09-06-08	10-06-08
34		State of the art	3 days	11-06-08	13-06-08
35		Project requirements, solution specification	5 days	16-06-08	20-06-08
36		Implementation details, evaluation of the solution	4 days	23-06-08	26-06-08
37		Conclusion and future work	3 days	27-06-08	01-07-08
38		Reviews	3 days	02-07-08	04-07-08
39		Documentation delivery	1 day?	04-07-08	04-07-08
40		Prepare presentation	15 days?	23-06-08	11-07-08
41		Production Scheduling problem: studying, implementing and solving	24 days	25-06-08	28-07-08
42		Review Documentation	24 days	25-06-08	28-07-08
43		Public presentations	7 days?	14-07-08	22-07-08
44		Final documentation delivery	1 day?	28-07-08	28-07-08

Figure 29: Detailed activities schedule of the IDE project.

The features referred in Chapter 3 were designed and implemented during the implementation phase, following an iterative approach based on the Feature Driven Development (FDD) methodology. FDD is a “client-centric, architecture-centric, and pragmatic software process”, based on the concept of Feature as a small, client-valued item usually described in the form of <action><result><object> [42] [43]. An example would be “Display the solution of a model”. A concise lifecycle model of the FDD methodology is presented on Figure 30.

Solution Specification



Copyright 2002-2005 Scott W. Ambler
Original Copyright S. R. Palmer & J.M. Felsing

Figure 30: The FDD project lifecycle.

Features are the primary input to the planning and development activities of the projects that follows this methodology. In the case of the ORML IDE project, features were documented with additional attributes, some mandatory (title, description, priority) and others optionally used where applicable (relations to use cases or other features, dependencies on components external to the IDE project).

By using this methodology, there isn't the risk of doing unnecessary detailed architectural work until the feature is actually about to be implemented, sparing precious time in situations where features' descriptions and priorities change frequently and when the time for implementing is short. Both conditions (features changing and short development time) were met in the IDE project, but because of this methodology, dropping features didn't mean throwing away a lot of work.

4.7 Summary

This chapter is the result of combining the problem description, its objectives and the research done, in order to provide the specification of a solution that would suit the stakeholders identified in Section 3.3, presenting the design and architecture for the common libraries of the joint project and for the IDE project, the schedule that the joint project was submitted to and ending with a reference to the work breakdown methodology followed in the project.

Solution Specification

Chapter 5

Implementation

This chapter presents noteworthy details of the implementation work done in the project, with focus on the integration of the various components presented in the previous chapter and technologies used.

While a significant part of the implementation work was done according to the planned schedule, many adjustments were made in order to accommodate deviations from the estimated durations and better coordination with the ORML Interpreter project. This chapter, however, does not make reference to those or other project management-related topics.

5.1 WPF ActiveX Control Host

As referred before, the `ActivexWpfHost` control is the link between the Dynamics AX Client user interface and the IDE. However, while Figure 11 (Components diagram of the joint project.), in page 8, shows it has a dependency on the ORML IDE, this dependency is only present in the physical set-up of the system—the ActiveX WPF Host can be used to host any WPF control, because the dependency is only resolved at run-time:

```
WpfContainer.SetRunningInsideAx(session.AOSName()+":"+int2str(session.aosPort()), curext());  
WpfContainer.SetChildFromName(  
    "Orml.Ide.MainForm, Version=1.0.0.0, Culture=neutral, PublicKeyToken=26b92f5575cc3b9f",  
    "Orml.Ide.MainForm.MainFrame");
```

Probably due to its simplicity, this control was implemented almost exactly as designed. In terms of implementation, both the class and the interface (`IActivexWpfHost`) had to be exportable and registered for COM Interop, which was supported by the build process of

Implementation

Visual Studio 2008, provided that `ActivexWpfHost` exposed the static methods `RegisterClass()` and `UnregisterClass()`.

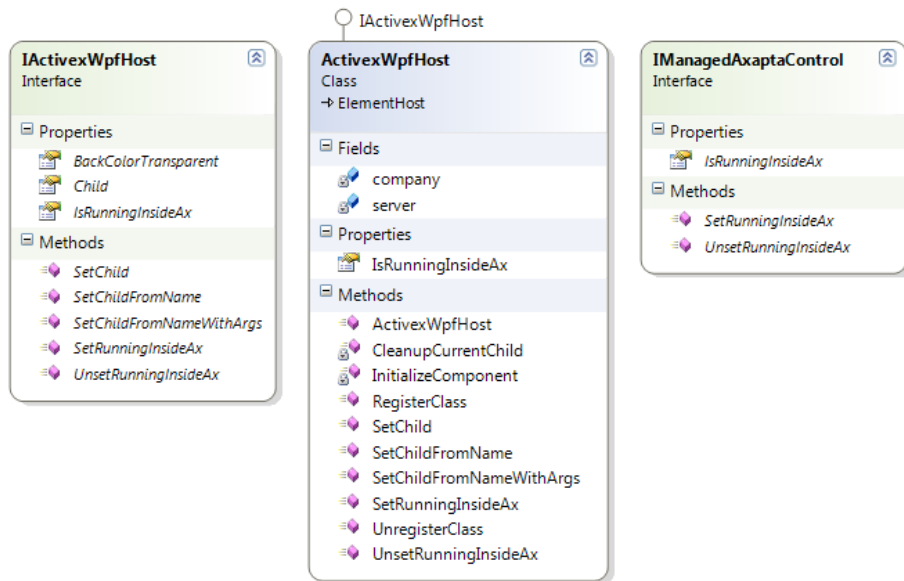


Figure 31: ActiveX WPF Host—class and interfaces in the assembly.

The `SetChild*()` methods were initially overloads with the same name—`SetChild()`. However, this is not supported in COM, so these methods had to be distinguished from each other by name. These three methods are the entry point for hosting a WPF control in Dynamics AX: when the object is set as child element of the `ActivexWpfHost` control, it is verified if the object implements the `IManagedAxaptaControl` interface, in order to “inform” it of the current environment (company and server currently being used by the Dynamics AX Client), calling the `SetRunningInsideAx()` method. Similarly, when the application is terminated, the child control is informed that it is no longer running inside Dynamics AX, via the `UnsetRunningInsideAx()` method.

5.2 Integrated Development Environment

5.2.1 Code Editor

The `CodeEditor` control provides text editing capabilities to the user. Its only contents are a `TextBox` and a `RichTextBox` placed side by side, the former to show the line numbers of the text that is displayed and editable on the latter. This control depends on an abstract `DatabaseAdapter` that it receives as a constructor argument, in order to re-use an already open database connection whenever one is necessary.

Implementation

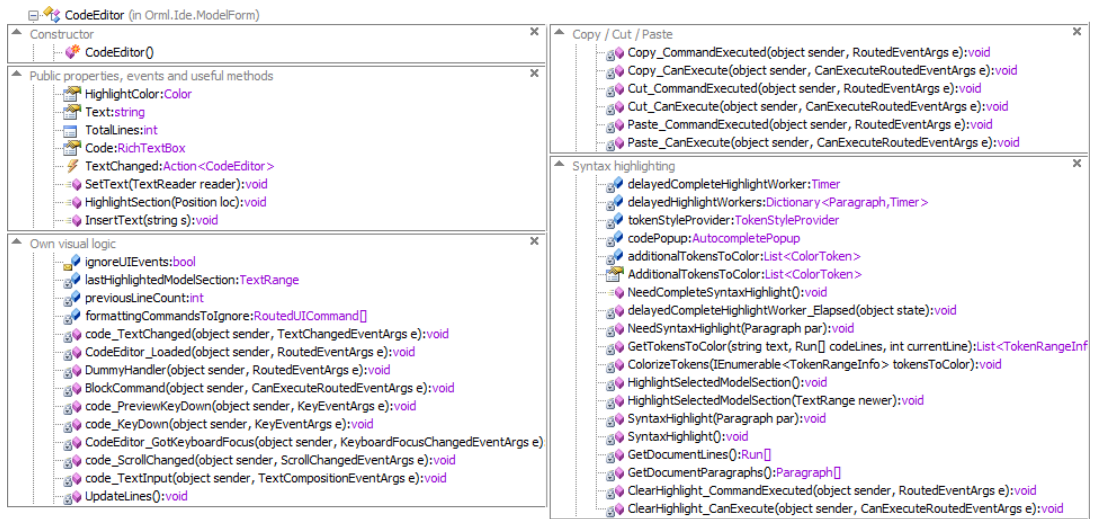


Figure 32: Orml.Ide.CodeEditor API, organized by functional sections.

This control makes use of the ORML Interpreter services in order to provide syntax highlighting as soon as a model is opened and continuously as the user makes changes to the model text. Because of the behavior of the RichTextBox control, its FlowDocument and derived, TextPointers and TextRanges and to do as little heavy lifting as possible in the UI thread, this must be done in a multi-pass process:

1. Pass to the Interpreter the text to be scanned (either all the document, or the line the user is editing) and wait for the results—this is done in a separate thread;
2. Clear current styles—from this step on, interaction with the user interface requires the operations to be performed on the UI thread;
3. For each token returned, build a TextRange from the start and end lines and columns, associating it with the token type;
4. For each TextRange, apply the style corresponding to the token type (accessed from a TokenStyleProvider object).

The model text is colored incrementally while the user is editing. While the syntax highlighting is triggered by the user changing the text, the highlight is only executed once the user stops editing for a moment (more precisely, one second). Coloring the text at exactly the same time that the user is editing would not only yield wrong results too often but, more importantly, would make the IDE appear to be less responsive. Also to enhance perceived responsiveness when editing, the coloring is only done after the last change, which means that if a user keeps typing for five seconds, the CodeEditor will not trigger the syntax highlighting process.

Implementation

Despite the continuous coloring, the presence of some kinds of tokens sometimes require the model text to be fully re-colored—namely, multi-line comments and strings. To correct the coloring of these kinds of tokens, a full re-coloring of the model text is triggered one and a half second after the last change.

A `RichTextBox` is a quite advanced control that supports more than formatted text—from paragraph indentation to images, a lot of content is supported by this editing control that is not the kind of content an ORML model can have. Because of those advanced capabilities of the WPF `RichTextBox` control, some “restraints” had to be applied:

- All formatting commands (e.g.: `CTRL+B` for bold, `CTRL+U` for underline) have been disabled;
- Some keys are pre-processed before they reach the `RichTextBox`,—the `TAB` key usually creates indentation, now only inserts spaces; the `ENTER` key usually creates a new paragraph, now only inserts a line break;
- The default context menu of the `RichTextBox` has been replaced for a simpler menu with only the Copy, Cut and Paste commands;
- In order to sanitize user input, every Paste operation converts the contents of the Windows clipboard to plain text—however, the Copy and Cut commands get the selected contents of the `RichTextBox` with formatting included; it would have been easy to convert it to text, but it may be useful to keep the formatting.

5.2.2 Visual Editor

Despite designed and its possible components identified, visual edition of the models’ equations and additional user text (previously referred to as code+documentation view) never got to be implemented, because it was not a high priority feature and the available time was devoted to other features. Before making this decision, research was conducted in order to find an off-the-shelf, low cost component that would serve as the base for the equation editor, but no component that could be re-used easily was considered a suitable alternative (free/open-source alternatives are still very limited and commercial ones are very expensive).

Later, turning to a in-house product—Microsoft Math (MS Math)—a possible fit to this feature was evaluated, but an analysis of the structure and behavior of the visual editor supplied by MS Math revealed that, despite this being a very complete math editing control, implementing this feature based on it would require adding support for custom constructs in the internal expression tree and serialization/deserialization procedures of the supported formats for representing equations. In face of this, this feature was chosen to be dropped in

Implementation

favor of the others that had the same priority (medium) and even in favor of some that had low priority.

5.2.3 Model Form

The `ModelForm` control was meant to be the host for the two planned controls for editing ORML models—the code editor and the visual editor—but ended up hosting only the code editor. However, the implementation of `ModelForm` was done having in mind a not too close integration with the `CodeEditor` control, which could result in making the `ModelForm` unsuitable for hosting other kinds of editing or visualization controls in the future.

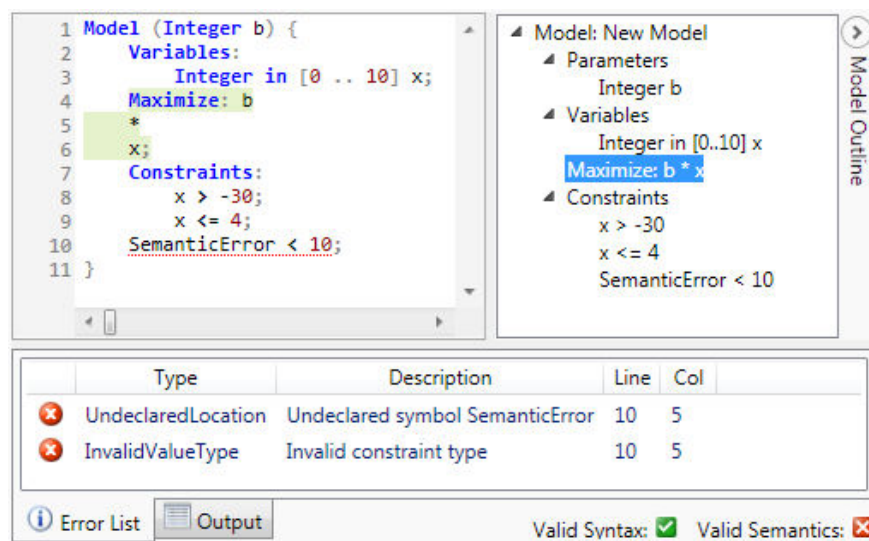


Figure 33: `Orml.Ide.ModelForm` showing code editor, model outline and error list.

The `ModelForm` control contains the text editing control (`CodeEditor`), the model outline panel, tabs for the list of errors and interpreter output and visual indicators of the syntactical and semantic correctness of the model.

The model outline tree is updated frequently, with a logic similar to that of the `CodeEditor` for the syntax highlighting—two seconds after the user stops editing, the Interpreter’s parser is invoked (on a different thread) and a `ModelStructureVisitor` object (a class that implements the `IVisitor` interface, defined in the Interpreter assemblies) traverses the tree in order to decompose the model in its sections and expressions.

In the same process, since the `OutputEvent` and `ErrorEvent` events of the Interpreter were subscribed by `ModelForm`, this control is able to set the correct icons for “Valid Syntax” and “Valid Semantics” (`OutputEvent` has special arguments for this) and populate the error list with information about any syntax or semantic errors.

Implementation

Both the model outline and error list provide some interactivity to the user: when the user double-clicks any line on the error list or leaf node in the model outline tree, the relevant section of the model is highlighted, as can be seen in Figure 33. This highlight is kept until another element is selected for outline or until the user presses the Escape key.

Since this control is not responsible for actually solving the models, the Output tab simply displays the text this control is requested to display (usually, by the MainForm).

5.2.4 Main Form

The MainForm class is the IDE itself. It's derived from a WPF UserControl and not from a WPF window in order to be included in other applications—in the specific case of this project it is used from Dynamics AX Client via the WPF ActiveX control host referred previously. That does not prevent the existence of a standalone ORML IDE application, but the user has to specify which server and company to use when running in standalone mode.

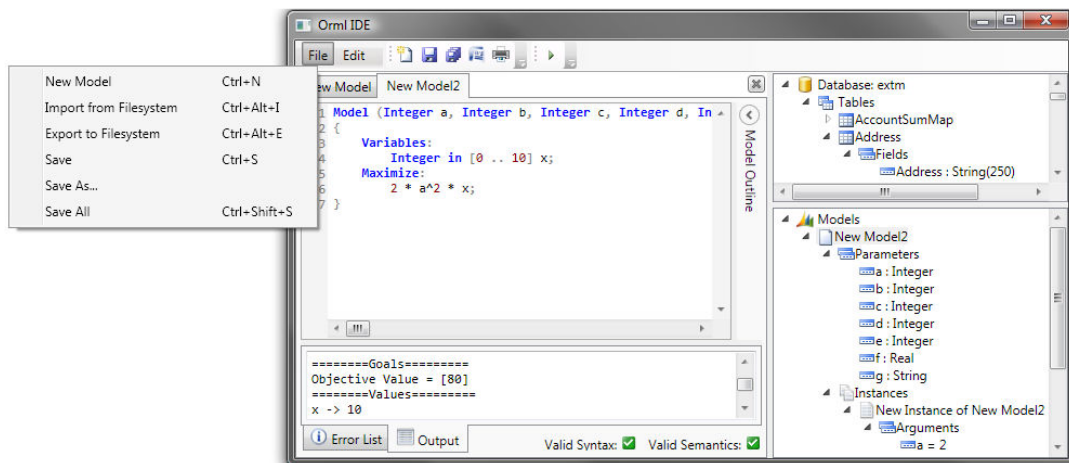


Figure 34: The ORML IDE user interface.

This control depends on the ModelForm control—instances of ModelForm are displayed in the models tab control and MainForm subscribes to events declared in ModelForm (e.g.: in order to be notified of changes in the model and display a “document changed” indicator in the heading of the tab of that document). Thanks to WPF’s event model, it is not necessary to attach any event handlers for commands issued by the user while the ModelForm has focus, because they will bubble up to the MainForm—for example, the focus may be on the CodeEditor’s RichTextBox, but when the user presses CTRL+N, it is the MainForm that handles the New Document command.

When the user issues the Run command, which means “solve the currently active model” and can originate from a keyboard combination or from the Run  button, it’s the MainFrame that determines if the user needs to supply the values for any arguments (asking them from

Implementation

the user if necessary, see Figure 36) and invokes the Interpreter, redirecting any textual output to the respective `ModelFrame`.

All the WPF commands are handled by this control independently of their source—they may originate in the business logic or by user input on the `MainForm` itself or any descendant control. If the commands require a specific target or context, it is either derived from the command itself or from the element that originated the command, for example:

- When handling the `CloseCurrentTab` command, the `MainForm` can easily “discover” the currently open tab;
- When handling the `RunModelOrInstance` command, the `DataContext` property of the control from where the command originated (usually a `TreeViewItem`, from the tree of models and instances stored in the database, but it makes no difference, since `DataContext` is a property shared by all WPF controls) is evaluated in order to get the `Model` or `ModelInstance` object that should be run.

There have been several references already to the `TreeView` that displays the models and instances stored in the database; this control is populated when the IDE starts and is kept in synchronization with the database while the user makes changes to models and instances. It was envisioned to follow the style of the Dynamics AX Application Object Tree (depicted on Figure 35)—having a similar structure and using icons familiar to Dynamics AX users—and supports a number of actions via the right-click context menu (see Appendix E). An example of one such action is creating a model instance, illustrated on Figure 36.

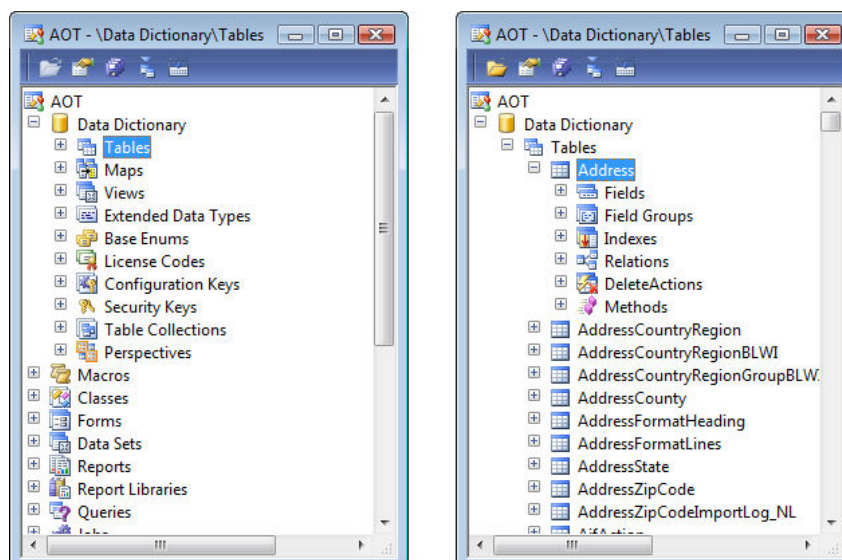


Figure 35: The Dynamics AX Application Object Tree (AOT).

Implementation

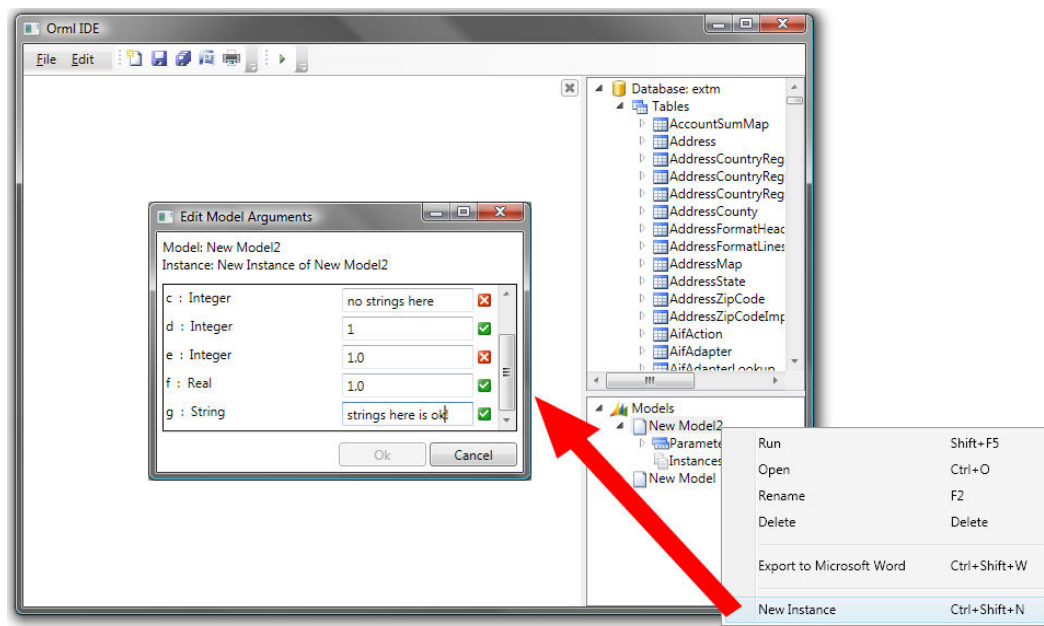


Figure 36: IDE screenshot featuring the context menu for a model tree item and the parameters editing window.

The Database TreeView is the other major control hosted in the MainFrame (others are menus and toolbars, that won't be referred). It shows the tables, views and queries (or in Dynamics AX's case, query-like classes) in the database that the user can use in models, allowing the user to, for example, insert a field in the model by double-clicking it.

This control is the greatest pain point of the IDE at the moment, because of the amount of data it has to handle: Dynamics AX has nearly 2800 tables that are displayed under the "Tables" tree node. The way that the WPF TreeViewItem control handles such an amount of items leads to a wait time of a few seconds when switching between the models explorer and the database tree tabs. To explore possibilities of circumventing this problem, two custom controls were prototyped (a TreeView based on a ListView that would only instantiate tree items that were visible, known as a virtualizing TreeView, and a custom TabControl that would keep all the tab items' contents visible in the same panel, changing the z-order when the user changed tabs), without success regarding to solving the problem.

The solution implemented to try to minimize this problem was to eliminate the tabs for the database explorer and models tree, displaying them side-by-side or one above the other, thus keeping both loaded and visible at all times. Although the response time of the database tree is still not immediate, this approach eliminated the delay previously existing when switching tabs. One possible solution to this delay problem would be to divide the presentation of database tables alphabetically, so that the "Tables" node would have only 26

Implementation

children nodes and each of those children would have at most a few hundred nodes. However, this approach was not completed at the time of writing this document; priority was given to fixing bugs and better supporting other features, since this performance problem was not impairing the overall quality of the IDE.

The model saving process of the IDE is also worth mentioning. When the user issues the `SaveAs` command or the `Save` command for a newly created document, the user is prompted for the new name for the model. If a model with that name already exists, the user is asked for confirmation on whether to save the model with a different name, overwrite the existing model or quit the save operation. If the `Save` command is issued for an already previously saved model, it is assumed that the user wants to save the model with the same name.

From this standpoint, it is evaluated whether the parameter signature of the new model matches the old one's (if there is a model to be replaced). If they match, the model is saved; on the contrary, if they don't match the user is prompted to choose whether to confirm saving the model and deleting all the existing instances of that model or to save the model with a different name.

As a final implementation note about `ModelForm`, it's worth mentioning that when the IDE is shutting down (either when running inside Dynamics AX or in standalone mode) the user is prompted whether to discard any unsaved changes made to open models or to save those changes (following the procedure described in the previous paragraphs for saving a model), before the IDE disposes of those changes.

5.2.5 IDE Utilities Library

This library contains utilities to aid mostly the user interface tasks that are common to the various controls of the IDE, like assigning the same handler to multiple commands' `CanExecute` and `Executed` events with very little code or controlling the logic of a delayed timer that has to be restarted when some event occurs.

One particular collection of utilities defined in this library is the collection of extensions to the `TreeViewItem` class. Nearly all the items in the various `TreeView` controls of the IDE have to be created programmatically; these extensions allow for standard and much cleaner and intuitive creation and manipulation of the `TreeViewItem` objects and particularly of their `Header` objects.

To give one example, instantiating a `TreeViewItem`, assigning it a `DataContext`, creating it's header with an icon, setting its style and associating a standard context menu would normally require the following code (without defining auxiliary variables):

Implementation

```
TreeViewItem modelNode = new TreeViewItem { DataContext = model };
modelNode.Header = new TextBlock();
((TextBlock)modelNode.Header).Inlines.Add(new Image());
((TextBlock)modelNode.Header).Inlines.Add(new Run(model.ModelName));
// wire the object to a parent that can find the styles, before assigning them
ModelsTreeRoot.Items.Add(modelNode);
((TextBlock)modelNode.Header).Style = modelNode.TryFindResource("NodeTextStyle") as Style;
((Image)((InlineUIContainer)((TextBlock)modelNode.Header).Inlines.FirstInline).Child).Style =
    modelNode.TryFindResource("ModelNodeIconStyle") as Style;
((TextBlock) modelNode.Header).ContextMenu = modelNode.TryFindResource("ModelNodeMenu") as ContextMenu;
```

With the extension methods implemented in this library, that code is reduced to five much more readable lines:

```
TreeViewItem modelNode = new TreeViewItem { DataContext = model }
    .CreateHeaderIconAndText(model.ModelName)
    .SetParent(ModelsTreeRoot) // wire the object to a parent that can find the styles, before assigning them
    .SetHeaderStyles("NodeTextStyle", "ModelNodeIconStyle")
    .SetHeaderContextMenu("ModelNodeMenu");
```

5.3 Other Features

This section presents noteworthy implementation details regarding features that were partially implemented by the author of the IDE project in the joint project.

5.3.1 Common Data Layer

Besides the functionalities of the Common Data Layer that have already been detailed, it is interesting to note that creating a `Table`, `View` or `Query` instance can only be done from the `DatabaseAdapter`, in order to wire the adapter to the object and seamlessly get metadata like field names and types. Similarly, creating a `TableRow` instance can only be done from the table, in order to transparently wire the table that originated the row.

This guarantees consistency when invoking methods that may require several objects to complete successfully—for example, the `Insert()` method on the `TableRow` element is actually just a shortcut method that has to access its parent `Table` and ultimately the `DatabaseAdapter` in order to actually insert the row in the database. The detailed object model of the Common Data Layer is depicted on Figure 37.

Another interesting implementation detail is the way metadata about Dynamics AX's database is collected by the `AxDatabaseAdapter` class. While initially this was done by navigating through the nodes of the Dynamics AX Application Object Tree (AOT), for example to find the list of tables and their fields and types, this procedure was later changed to take advantage of a system table that contains almost all of this information (only the field types have to be fetched from the AOT nodes now), which resulted in a ten times performance increase over iterating the AOT nodes to get all the information (both done via Dynamics AX's Business Connector for .NET).

Implementation

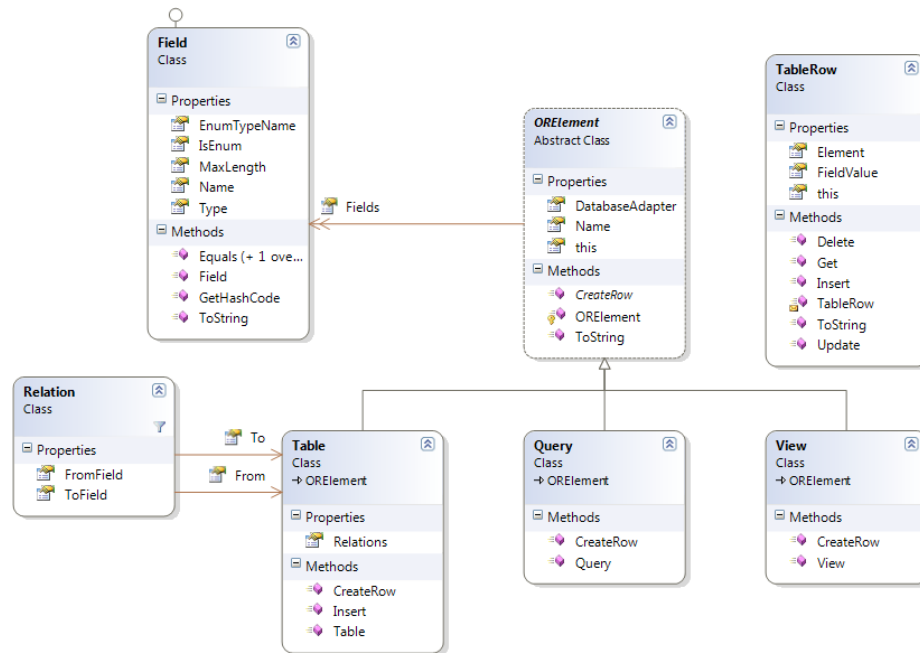


Figure 37: Common Data Layer class diagram (excl. adapters).

5.3.2 Models Management Library

While some of the particularities of this library have already been mentioned in previous sections, there is one aspect that is worth highlighting in this section: due to any possible failures in the process of saving a record to the database, the text to be saved is kept in a temporary variable (the actual `newText` parameter to the `Save()` or `SaveAs()` methods) until the database transaction is actually completed and only then assigned to the actual `ModelText` property of the model. This keeps a consistent state between what is in the object and what really is in the database, allowing consumer applications to be notified of any failures in the saving process.

5.4 Evaluation of the Solution

This section presents self-critical comments regarding the solution that was planned and implemented.

5.4.1 Tests and Integration

The IDE project was not planned for a test driven development approach and due to time constraints, the test plan for the IDE was never actually put to practice—it was necessary to create test design specification documents for the features and use cases that would be tested and implement the automation for those tests or execute them manually.

Implementation

This resulted in the not very comfortable situation of not having any documented tests for the IDE, despite the existence of tests for the common libraries and the extensive testing of the Interpreter. However, there was cohesion in the planning and development work of the IDE, Interpreter and common components. While the manual day-to-day undocumented tests don't guarantee the correctness of the IDE, that together with the continuous integration between the IDE and the rest of the components of the joint project provide some confidence when it comes to affirming that the IDE is stable now; to guarantee that the IDE stays stable in the future while new functionality is added and existing functionality may be modified or refactored, it would be important to plan and implement the tests for its features.

5.4.2 Usage Experience

Usability testing of the IDE was considered, in order to better assess the usage experience and usability of the IDE. However, studies point that in order to get an accurate result from a usability test with a good confidence interval (90%, in the case illustrated on Figure 38), it would be recommended to test with 20 users, which would yield a margin of error of about 19% when excluding the most deviated result (effectively considering 19 users) [47].

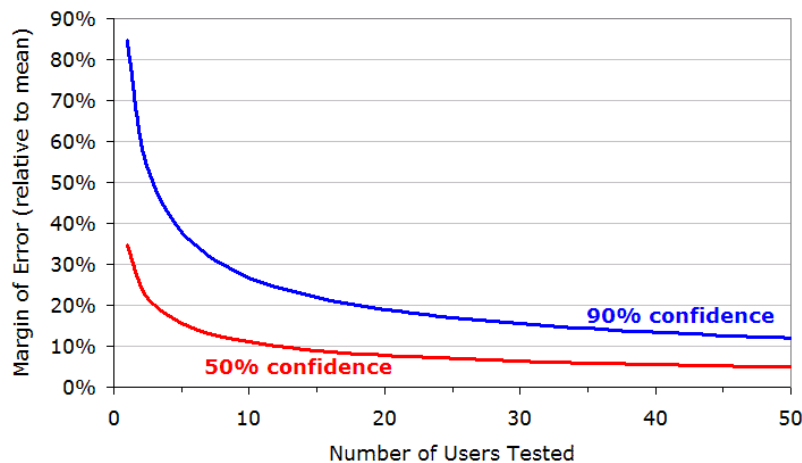


Figure 38: Margin of error for testing with various numbers of users [47].

This means that the time results of the usability tests would have to be considered to be 19% longer in the worst case, but can be 19% better in the best case. It is also pointed that from a qualitative point of view it may suffice to test with fewer users [49], but while testing with fewer users can expose usability problems, those tests should be repeated in order to yield accurate conclusions regarding the usability of the system being tested.

Implementation

From this standpoint, it was decided to devote the available time to identifying common patterns and behaviors present in many Windows applications, especially the ones more related to the ORML IDE (IDEs in general, IDEs for mathematical languages and the target application at which the ORML IDE is targeted—Dynamics AX 2009) and implement them in the IDE, when applicable.

Starting with the Use Cases, some hints for a better user experience can be pointed, like stating that “the IDE must distinguish between a regular ORML code file and a code+documentation XML file and open both seamlessly” and that the user must be presented with the solution to a model, even if the outputs of the model are bound to the database. However, most usability aspects of the IDE can be identified in the design of the user interface and implemented interactions:

- Multiple open documents are displayed on tabs in the central region of the IDE, following a pattern that IDEs and browsers widely support;
- Information that the users usually don’t need to see or use simultaneously is grouped in a tabs control (model error list vs. interpreter output log);
- When certain actions are taken, the controls that display the result of those actions are focused or made visible:
 - When the user double-clicks a table field, it is inserted in the code editor and the code editor is focused;
 - When the user runs an open model, the interpreter output tab is activated, but the focus remains in the original element (that may very well be the code editor, so it would be counterproductive to take the focus out of there);
- The context menu of the `TreeView` control that displays the modes and instances in the database adapts itself to the element that is being right-clicked, displaying the relevant operations that can be performed over that item (and contrarily to the default `TreeView` behavior, the item that was right-clicked is selected and highlighted, to give the user a clear visual hint of what element is being acted upon);
- Buttons and menu items that trigger commands that need an active document are disabled and “grayed-out” if no documents are available, following a very common trend among Windows applications;
- Dialog boxes with several choices clearly state what actions the IDE will take for the different available choices (depicted in Figure 39).

Implementation

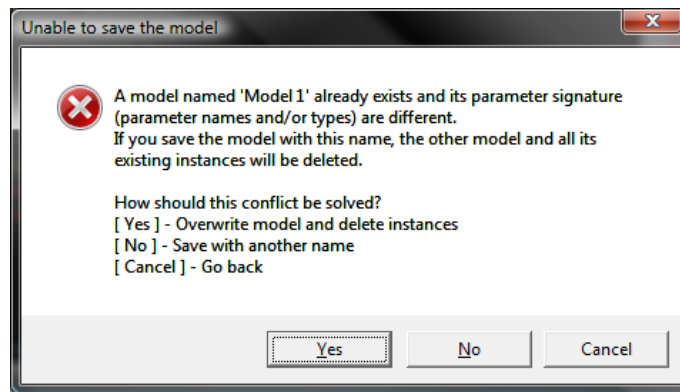


Figure 39: Dialog box presenting the choices available when conflicts arise while saving a model.

Critically analyzing these usability aspects of the IDE, it can be justly argued that following recognizable design and behavior without conducting and documenting usability tests does not prove that the IDE is actually easy to learn and use, but the IDE was actually designed and implemented based on the idea that good design should be invisible [50] [51], so it should perform well when used to create and manage models and instances.

5.5 Summary

This chapter presented details about the implementation of the planned solution, focusing on noteworthy specific program flows and features that otherwise would pass unnoticed and presenting in more detail some of the technologies that back the project. Finally, a self-critical evaluation of the solution elaborated on the testing, integration and usage experience aspects of the IDE, concluding the in-depth technical topics of this document.

Chapter 6

Conclusions and Future Work

Throughout the duration of the ORML IDE project, it was possible to go through all the phases of a software project, surpassing technical and social challenges that were essential to the good conclusion of the project. This thesis is the most important and, side by side with the system that was produced, the most tangible result of the project. Even if the IDE project team was only composed of one person, working in close collaboration with the author of the Interpreter project demanded documented decisions and concerted planning and executing.

The author of this project followed an iterative approach to the software development process (based on the Feature Driven Development software process): eliciting and refining the requirements before any work was done and during the design and development of the project; planning the architecture of each feature just when necessary, and not before; prototyping with the objective of evaluating the viability and value of using certain technologies and techniques; changing the architecture and implementation of previous features in an evolutionary cycle, when required by the architecture and implementation of other features; and regularly testing the system as a whole.

Despite being a one-person team, the team devoted to the IDE project required management: by the supervisors of Microsoft and FEUP devoted to this project, but also, first and foremost throughout the project, by the author of the project. Managing time, project scope, external expectations and participatory management in the planning, execution and promotion of the joint project were essential. It was also important to participate in analyzing and solving a real optimization problem in Dynamics AX: the warehouse picking routes problem started as a workshop held by the authors and supervisors

Conclusions and Future Work

of the joint project with the Dynamics AX Inventory team; a problem was defined (with several degrees of complexity) and actually solved using the system developed in the joint project (the less complex version of the problem, at least). These management and promotion activities remind us that it is not enough to build and deliver a product; it is very important to create interest around a project, which will provide valuable input from the environment and almost certainly have a positive influence on the project.

In the case of this project, it was also important to have the support of experienced elements in the company, which helped at bootstrapping the project and occasionally nudging it in the right direction. Despite consuming a seemingly precious amount of time, these “side-track” activities that promote the project can be very important to the degree of success (or simply the survival) of a project, especially in a company as large as Microsoft.

6.1 Satisfaction of the Objectives

The objectives of this project were previously stated in Chapter 1 and are re-stated here. Based on this list, a point-by-point evaluation of the satisfaction of the project’s objectives will follow, concluded by an assessment of the overall success of the project.

1. *Research and document the state of the art on methodologies and technologies regarding the type of problems specified and how they are modeled and solved, and on current IDEs and other solutions to the referred kind of functionalities:* while the time allotted for achieving this objective was not long, additional research needed during the project contributed to a fairly complete analysis of the state-of-the-art regarding the topics of the project;
2. *Define the requirements for the ORML IDE:* this objective was pursued during much of the time preceding the first milestone (in parallel with other activities) and the requirements were further refined until the second milestone was achieved;
3. *Prototype a simple user interface that is integrated into Microsoft Dynamics AX’s graphical interface and interacts minimally with the database and ORML Interpreter to solve a simple LP problem, in order to prove the feasibility of the project as it was defined initially or review the project definition, if necessary:* the possibility of integrating a custom control in Dynamics AX user interface that interacts with its database was proven (the first part of this prototype is documented in Appendix C, the database interaction part was done in a simpler prototype used by the first—what later evolved to be the Common Data Layer);

Conclusions and Future Work

4. *Design and implement the ORML IDE;*
5. *Provide integration with the ORML interpreter;*
6. *Provide integration with Microsoft Dynamics AX database, both in terms of data and metadata;*
7. *Provide the feature of exporting the model to a file, in a human-readable, textual and mathematical representation;*

Even if not all the planned features were implemented, the successful achievement of objectives 4, 5, 6 and 7 has already been exposed in previous chapters;

8. *Use state-of-the-art and in-house (“house” here being Microsoft) technologies, when possible:* the developed solution uses components released recently (Microsoft Math was released in 2006 and is expected to be updated in the future, the .NET Framework 3.5 was released in 2006 and updated in 2007) and does not depend on any non-Microsoft products;
9. *Provide a good user experience and overall responsiveness of the ORML IDE:* while the overall responsiveness is definitely an achieved goal, the quality of the usage experience may be argued, as was exposed in Section 5.4.2, Usage Experience.

Form an overview perspective it is safe to say that while some loose ends do exist, the project succeeded overall. Some features and planned activities had to be dropped—the uncertainty of the type of work that had to be done led to planning more than could be executed.

6.2 Future Work

Regarding the continuation of this project, there were identified some features and activities that would benefit the application.

Despite designed and its possible components identified, visual edition of a model's equations and additional user text (previously referred to as code+documentation view) never got to be implemented, because it was not a high priority feature and the available time was devoted to other features. This is perhaps the most interesting feature to implement in the future, because it would allow the user to have a great deal of support when editing a model and still collect the benefits of the model being generated in the ORML language.

In the continuum of the project, as new features are added and existing design and functionality may change, it would be important to put to practice the ORML IDE project test plan. This would probably require a continuous dedicated work stream—i.e., it would

Conclusions and Future Work

require the full attention of the development team on this task without implementing any new features—in order to analyze the IDE, document the intended state transitions when features are exercised and ultimately implement the corresponding automation tests (even if no automation was implemented, the tests could still be carried out manually).

References

- [1] Encyclopædia Britannica, Inc. *Software Engineering*. [On-line; accessed June 11, 2008] <http://www.britannica.com/eb/article-21951/computer-science>.
- [2] Jack Herrington. IBM DeveloperWorks. *Seven great PHP IDEs compared*. [Online; accessed June 11, 2008] <http://www.ibm.com/developerworks/opensource/library/os-php-ide/index.html>.
- [3] Paragon Decision Technology. *An AIMMS Overview*. [Online; accessed June 11, 2008] <http://www.aimms.com/aimms/product/overview.html>.
- [4] Paragon Decision Technology. *The AIMMS User Interface*. [Online; accessed June 11, 2008] http://www.aimms.com/aimms/product/user_interface.html.
- [5] ILOG, Inc. *ILOG OPL Development Studio*. [Online; accessed June 11, 2008] <http://www.ilog.co.uk/products/oplstudio/index.cfm>.
- [6] ILOG, Inc. *ILOG OPL Development Studio Features*. [Online; accessed June 11, 2008] <http://www.ilog.co.uk/products/oplstudio/features.cfm>.
- [7] LINDO Systems, Inc. *LINGO 11.0*. [Online; accessed June 11, 2008] <http://www.lindo.com/products/lingo/lingom.html>.
- [8] Maximal Software, Inc. *MPL Modeling System*. [Online; accessed June 12, 2008] <http://www.maximal-usa.com/mpl/>.
- [9] Wolfram Research, Inc. *Wolfram Mathematica 6*. [Online; accessed June 11, 2008] <http://www.wolfram.com/products/mathematica/index.html>.
- [10] Wolfram Research, Inc. *Wolfram Mathematica Documentation Center, Linear Programming*. [Online; accessed June 11, 2008] <http://reference.wolfram.com/mathematica/tutorial/ConstrainedOptimizationLinearProgramming.html>.

References

- [11] Jorge J. Moré and Stephen J. Wright. *Optimization Software Guide*. SIAM, pages 65-148, 1993.
- [12] NEOS Server team. Argonne National Laboratory (operated by The University of Chicago), U.S.A. Department of Energy. *Optimization Software*. [Online; accessed June 12, 2008] <http://www-new.mcs.anl.gov/otc/Guide/SoftwareGuide/>.
- [13] The Eclipse Foundation. *Eclipse Wiki*. [Online; accessed June 14, 2008] <http://wiki.eclipse.org/>.
- [14] The Eclipse Foundation. *Language IDE*. [Online; accessed June 16, 2008] <http://www.eclipse.org/home/categories/languages.php>.
- [15] Sun Microsystems, Inc. and CollabNet, Inc. *NetBeans IDE 6.1 Features*. [Online; accessed June 16, 2008] <http://www.netbeans.org/features/>.
- [16] JetBrains s.r.o. *IntelliJ IDEA 7.0 Features*. [Online; accessed June 16, 2008] <http://www.jetbrains.com/idea/features/index.html>.
- [17] Microsoft Corporation. *Visual Studio 2008 Overview*. [Online; accessed June 16, 2008] <http://msdn.microsoft.com/en-us/vstudio/products/bb931331.aspx>.
- [18] Microsoft Corporation. *Application Development in Visual Studio 2008*. [Online; accessed June 16, 2008] <http://msdn.microsoft.com/en-us/library/h8w79z10.aspx>.
- [19] Microsoft Corporation. *Visual Studio Shell*. [Online; accessed June 16, 2008] Available on <http://msdn.microsoft.com/en-us/vsx/bb933751.aspx>.
- [20] Arthur Greef, et al. *Inside Microsoft Dynamics AX 4.0*. Microsoft Press, 2006.
- [21] Chris Sells and Ian Griffiths. *Programming WPF*. O'Reilly, 2007.
- [22] Adam Nathan. *Windows Presentation Foundation unleashed*. Sams, 2007.
- [23] Microsoft Corporation. *Windows Forms in the .NET Framework*. [Online; accessed June 16, 2008] [http://msdn.microsoft.com/en-us/library/96bef039\(VS.80\).aspx](http://msdn.microsoft.com/en-us/library/96bef039(VS.80).aspx).
- [24] Microsoft Corporation. *Smart Client - Composite UI Application Block*. [Online; accessed June 16, 2008] <http://msdn.microsoft.com/en-us/library/aa480450.aspx>.
- [25] Microsoft Corporation. *Patterns & Practices Smart Client Guidance Community Site*. [Online; accessed June 16, 2008] <http://www.codeplex.com/smartclient>.
- [26] Microsoft Corporation. *Windows Presentation Foundation*. [Online; accessed June 16, 2008] Available on <http://msdn.microsoft.com/en-us/library/ms754130.aspx>.
- [27] Microsoft Corporation. *The LINQ Project*. [Online; accessed June 16, 2008] <http://msdn.microsoft.com/en-us/netframework/aa904594.aspx>.
- [28] Microsoft Corporation. *Microsoft Active Accessibility*. [Online; accessed June 16, 2008] <http://msdn.microsoft.com/en-us/library/ms697707.aspx>.

References

- [29] Microsoft Corporation. *UI Automation*. [Online; accessed June 16, 2008] <http://msdn.microsoft.com/en-us/library/ms747327.aspx>.
- [30] Microsoft Corporation. *UI Automation and Microsoft Active Accessibility*. [Online; accessed June 16, 2008] <http://msdn.microsoft.com/en-us/library/ms788733.aspx>.
- [31] Charles Rankin. *The Software Testing Automation Framework*. IBM Systems Journal, Vol. 41, No. 1, pages 126-139, 2002.
- [32] James McCaffrey. *.NET Test Automation Recipes: A Problem-Solution Approach*. Apress, 2006.
- [33] Encyclopædia Britannica, Inc. *Operations Research*. [Online; accessed June 18, 2008] <http://www.britannica.com/eb/article-9106307/operations-research>.
- [34] Kenneth Laudon and Jane Laudon. *Management Information Systems: Managing the Digital Firm*, Seventh Edition. Prentice Hall, 2001.
- [35] Vipul Jain and Ignacio E. Grossmann. *Algorithms for Hybrid MILP/CP Models for a Class of Optimization Problems*. INFORMS Journal on Computing, Vol. 13, No. 4, pages 266-268, 2001.
- [36] Microsoft Corporation. *Microsoft Releases Microsoft Dynamics AX 2009*. [Online; accessed July 2, 2008] <http://www.microsoft.com/Presspass/press/2008/jun08/06-02DynamicsAX2009GAPR.msp>.
- [37] Ecma International. *What is Ecma International*. [Online; accessed July 2, 1008] <http://www.ecma-international.org/memento/index.html>.
- [38] Ecma International. *Office Open XML File Formats*. Ecma International, pages 11-15.
- [39] Microsoft Corporation. *Open XML Developer*. [Online; accessed July 2, 2008] <http://openxmldeveloper.org/>.
- [40] Microsoft Corporation. *Primary Interop Assemblies (PIAs) and Interop Assemblies (IAs)*. [Online; accessed July 2, 2008] <http://msdn.microsoft.com/en-us/library/aa169585.aspx>.
- [41] André Rodrigues. *Operations Research Modeling Language for an ERP System*. Faculty of Engineering of the University of Porto, 2008.
- [42] Nebulon Pty. Ltd. *Feature Driven Development*. [Online; accessed July 3, 2008] <http://www.featuredrivendevelopment.com/>.
- [43] Scott Ambler. *Feature Driven Development and Agile Modeling*. [Online; accessed July 3, 2008] <http://www.agilemodeling.com/essays/fdd.htm>.

References

- [44] Erich Gamma, et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, 1995.
- [45] Microsoft Corporation. *Microsoft Math*. [Online; accessed July 5, 2008] <http://www.microsoft.com/math/>.
- [46] Microsoft Corporation. *Microsoft Math SDK Manual*. Microsoft, 2006.
- [47] Jakob Nielsen. *Quantitative Studies: How Many Users to Test?* [Online; accessed July 5, 2008] http://www.useit.com/alertbox/quantitative_testing.html, June 26, 2006.
- [48] Paul Andrew. *The .NET Framework 3.5 Commonly Used Types and Namespaces poster*. [Online; accessed July 4, 2008] <http://blogs.msdn.com/pandrew/archive/2007/11/02/announcing-the-net-framework-3-5-commonly-used-types-and-namespaces-poster.aspx>, November 2, 2007.
- [49] Jakob Nielsen. *Usability Testing With 5 Users*. [Online; accessed July 5, 2008] <http://www.useit.com/alertbox/20000319.html>, March 19, 2000.
- [50] Liz Danzico. *How Not To Get Noticed*. [Online; accessed July 5, 2008] <http://bobulate.com/2007/07/22/how-not-to-get-noticed/>, July 22, 2007.
- [51] Ron Tolido. *The Only Good Technology is No Technology*. [Online; accessed July 5, 2008] http://www.capgemini.com/ctoblog/2008/07/the_only_good_technology_is_no.php, July 5, 2008.
- [52] Paul Andrew. *The .NET Framework 3.5 Commonly Used Types and Namespaces poster*. [Online; accessed July 5, 2008] <http://blogs.msdn.com/pandrew/archive/2007/11/02/announcing-the-net-framework-3-5-commonly-used-types-and-namespaces-poster.aspx>, November 2, 2007.
- [53] Microsoft Corporation. *Introduction to ActiveX Controls*. [Online; accessed July 7, 2008] [http://msdn.microsoft.com/en-us/library/aa751972\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/aa751972(VS.85).aspx).
- [54] Microsoft Corporation. *COM: Component Object Model Technologies*. [Online; accessed July 7, 2008] <http://www.microsoft.com/com/default.mspix>.
- [55] The World Wide Web Consortium. *What is MathML?* [Online; accessed July 7, 2008] <http://www.w3.org/Math/>.
- [56] Trygve Reenskaug. *MVC*. [Online; accessed July 7, 2008] Available on <http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html>.
- [57] Microsoft Corporation. *RecordSet Object (ADO)*. [Online; accessed July 7, 2008] <http://msdn.microsoft.com/en-us/library/ms681510.aspx>.
- [58] The World Wide Web Consortium. *Extensible Markup Language (XML)*. [Online; accessed July 7, 2008] <http://www.w3.org/XML/>.

References .

[59] The World Wide Web Consortium. *The Extensible Stylesheet Language Family (XSL)*. [Online; accessed July 7, 2008] <http://www.w3.org/Style/XSL/>.

[60] Microsoft Corporation. *X++ Programming Guide*. [Online; accessed July 7, 2008] <http://msdn.microsoft.com/en-us/library/aa867122.aspx>.

Appendix A

Declarative Mathematical Languages: Studying an Operations Research Problem (Production Planning)

Despite being outside the planning and development scope of the ORML IDE project, the definition of the ORML language had some participation of the author of this project. The most tangible artifact of that participation is a study that was made of a classic operations research problem (production planning) whose objective was to propose a possible declarative syntax for the various constructs of the language.

The problem is defined like this in [11]:

«We consider a specific scheduling problem that falls into the class of problems considered in this paper and that is similar to the one considered by Hooker et al. (1999). This scheduling problem involves finding a least-cost schedule to process a set of orders I using a set of dissimilar parallel machines M . Processing of an order $i \in I$ can only begin after the release date r_i and must be completed at the latest by the due date d_i . Order i can be processed on any of the machines. The processing cost and the processing time of order $i \in I$ on machine $m \in M$ are C_{im} and p_{im} , respectively. »

The referred paper presents a solution to the problem, which is also transcribed here:

«The scheduling problem described above can be modeled as an MILP.

The main decisions involved in this scheduling problem are assignment of orders on machines, sequence of orders on each machine, and start time for

Declarative Mathematical Languages: Studying an Operations Research Problem
(Production Planning)

all the orders. The binary variable x_{im} is an assignment variable, and it is equal to one when order i is assigned to machine m . Binary variable $y_{ii'}$ is the sequencing variable, and it is equal to one when both i and i' are assigned to the same machine and order i' is processed after order i . The continuous variable ts_i denotes the start time of order i . Using these variables the MILP model for this problem can be written as:

$$\min \sum_{i \in I} \sum_{m \in M} C_{im} x_{im} \quad (15)$$

$$\text{s.t. } ts_i \geq r_i \quad \forall i \in I \quad (16)$$

$$ts_i \leq d_i - \sum_{m \in M} p_{im} x_{im} \quad \forall i \in I \quad (17)$$

$$\sum_{m \in M} x_{im} = 1 \quad \forall i \in I \quad (18)$$

$$\sum_{i \in I} x_{im} p_{im} \leq \max_i \{d_i\} - \min_i \{r_i\} \quad \forall m \in M \quad (19)$$

$$y_{ii'} + y_{i'i} \geq x_{im} + x_{i'm} - 1 \\ \forall i, i' \in I, i' > i, m \in M \quad (20)$$

$$ts_{i'} \geq ts_i + \sum_{m \in M} p_{im} x_{im} - U(1 - y_{ii'}) \\ \forall i, i' \in I, i' \neq i \quad (21)$$

$$y_{ii'} + y_{i'i} \leq 1 \quad \forall i, i' \in I, i' > i \quad (22)$$

$$y_{ii'} + y_{i'i} + x_{im} + x_{i'm'} \leq 2 \\ \forall i, i' \in I, i' > i, m, m' \in M, m \neq m' \quad (23)$$

$$ts_i \geq 0 \quad (24)$$

$$x_{im} \in \{0, 1\} \quad \forall i \in I, m \in M \quad (25)$$

$$y_{ii'} \in \{0, 1\} \quad \forall i, i' \in I, i' \neq i \quad (26)$$

«Objective (15) of this problem is to minimize the processing cost of all the orders. Constraints (16) and (17) ensure that processing of an order i starts after the release date and is completed before the due date. An order needs exactly one machine for processing and this is enforced using assignment constraint (18). Inequality (19) is a valid cut, and it tightens the LP relaxation of the problem. It is based on the fact that the total processing time of all the orders that are assigned on the same machine should be less than the difference of the latest due date and the earliest release date. Constraint (20) is a logical relationship between assignment and sequencing variables. The underlying logic behind this constraint is that if order i and i' are assigned to machine m then they must be processed one after the other. Constraint (21) is the sequencing constraint in which $U = \sum_{i \in I} \max_{m \in M} \{p_{im}\}$. It ensures that if the sequencing variable $y_{ii'}$ is one, then

Declarative Mathematical Languages: Studying an Operations Research Problem (Production Planning)

order i' is processed after order i . Constraints (22) and (23) are logical cuts. The former is based on the logic that either i is processed before i' or vice versa. The latter ensures that sequencing variables are zero if i and i' are assigned to different machines. Adding both of these constraints reduces the computational time needed to solve the MILP very significantly. Other logic cuts were also tried, but they were not as effective as (22) and (23). »

Syntax Proposal:

```
Constants { // assume they are filled up
  Integer M : [1..3]; // const range, represents the set of machines available
  Integer I : [1..4]; // const range, represents the set of jobs
  Integer T : [1..31]; // const range, represents the work days available
  Integer Costs[I.Count][M.Count]; // cost of processing order i in machine m
  Integer ProcessingTime[I.Count][M.Count] in T; // time taken processing order i in machine m
  Integer ReleaseTime[I.Count] int T; // ReleaseTime[i] is the time at which order i was issued
  Integer DueTime[I.Count] int T; // DueTime[i] is the time at which order i must be ready
}

Variables {
  Integer Schedule[I.Count][M.Count] in [0..1];
  // each position Schedule[i][m] is true if order i is processed by machine m, false otherwise

  Integer Before[I.Count][I.Count] in [0..1];
  // each position Before[i][j] is true if order i comes before order j and is processed on the same machine

  Integer StartTime[i] in T; // time at which order i will start to be produced
  Integer U; // just to be used as a constant later...
}

Minimise {
  Sum(i in I, sum(m in M, Costs[i][m] * Schedule[i][m])); // (15)
}

Constraints {
  having(i in I): StartTime[i] >= ReleaseTime[i]; // (16)
  having(i in I): StartTime[i] <= DueTime[i] - sum(m in M, ProcessingTime[i][m] * Schedule[i][m]); // (17)
  having(i in I): sum(m in M, Schedule[i][m]) == 1; // (18)
  having(m in M): sum(i in I, Schedule[i][m]) <= max(i in I, DueTime[i]) - min(i in I, ReleaseTime[i]); // (19)
  having(i in I, j in I, m in M, j > i): Before[i][j] + Before[j][i] >= Schedule[i][m] + Schedule[j][m] - 1; // (20)

  U = sum(i in I, max(m in M, ProcessingTime[i][m]));
  // this is more of a constant... just to be used in the next constraint

  having(i in I, j in I, i != j):
    StartTime[j] >= StartTime[i] + sum(m in M, ProcessingTime[i][m]) - U * (1 - Before[i][j]); // (21)
  having(i in I, j in I, j > i): Before[i][j] + Before[j][i] <= i; // (22)
  having(i in I, j in I, m in M, n in M, j > i, m != n):
    Before[i][j] + Before[j][i] + Schedule[i][m] + Schedule[j][n] <= 2; // (23)
  having(i in I): StartTime[i] >= 0; // (24)
  // (25) and (26) are already expressed in the variables declaration: the domains of the variables
}
```

Appendix B

List of Features of the ORML IDE

Feature group: Basic functionality

IDE-F001: Present a list of the models saved in the database

This list should be more of a tree, presenting not only the list of models but also their parameters and instances and allowing for direct interaction like opening a model from the list.

Priority: High.

Relates to use cases IDE-UC01.

IDE-F002: Open an ORML code file, loading it in the code editor

Priority: High.

Relates to use case IDE-UC01.

IDE-F003: Compile a model

Uses the ORML Interpreter.

Priority: High.

Relates to use case IDE-UC05.

IDE-F004: Capture the result of a compilation so that it is available to the user

Priority: High.

Relates to use case IDE-UC05 and feature IDE-F003: Compile a model.

IDE-F005: Provide navigation from compilation errors/warnings to code

Priority: Medium.

Relates to use case IDE-UC05 and feature IDE-F004: Capture the result of a compilation so that it is available to the user.

IDE-F006: Run a model

The model must be compiled if any changes to it were made since the last compilation or if it is a model that the user wants to run directly from file. In the latter case, if there are any compilation errors, the file is opened for editing and the errors are reported to the user. If not, the solution to the model is displayed. If the model has any arguments, they shall be asked from the user before actually running the model.

Uses the ORML Interpreter.

Priority: High.

Relates to use cases IDE-UC05, IDE-UC06 and features IDE-F003, IDE-F007, IDE-F010.

IDE-F007: Display the solution to a model

The solution view must provide an option to save the results to the database.

Priority: High.

Relates to use cases IDE-UC06, IDE-UC07.

IDE-F009: Run a model instance

Priority: High.

Relates to use case IDE-UC07 and features IDE-F006, IDE-F007, IDE-F010.

IDE-F010: Get user input on argument values to run a model or an instance

A dialog should be displayed in which the user inputs the values of the arguments. The values should be validated according to the data type the model requires for each parameter.

Priority: High.

Relates to use cases IDE-UC06, IDE-UC09, IDE-UC11 and features IDE-F006.

Feature group: Model Editing

IDE-F021: Provide a text editing area

To allow the user to edit ORML code.

Priority: High.

Relates to use case UDE-UC02.

IDE-F022: Provide syntax highlighting in the text editor

Uses the ORML Interpreter.

Priority: Medium.

Relates to feature IDE-F021.

IDE-F022: Provide code assistance / “intellisense” in the text editor

Possibly uses the ORML Interpreter.

Priority: Low.

Relates to feature IDE-F021.

IDE-F023: Provide a tree view of the data sources available in the database

Data sources are tables, views and queries.

Uses the ORML Common Data Layer.

Priority: Medium.

Relates to feature IDE-F021.

IDE-F024: Add correct code to the editor from a user selection in the data sources tree view

Priority: Medium.

Relates to feature IDE-F023.

IDE-F025: Allow the user to preview the contents of a data source selected in the data sources tree view

Uses the ORML Common Data Layer.

Priority: Low.

Relates to feature IDE-F023.

IDE-F026: Provide a tree view of the model sections and their child elements

Possibly uses the ORML Interpreter.

Priority: Medium.

Relates to feature IDE-F021.

IDE-F027: Focus the portion of code that corresponds to the element selected by the user on the model sections tree view

Possibly uses the compiler/interpreter services.

Priority: Low.

Relates to feature IDE-F026.

IDE-F028: Provide a visual editor for the model's sections and expressions

This editor should also allow the user to enter comments between the model's sections and expressions

Priority: Medium.

Relates to use case IDE-UC02.

Feature Group: Models and Instances Management

IDE-F041: Provide a list or tree view of the models that reside in the database

Priority: Medium.

Uses the Model Management Library.

IDE-F042: Open a model selected by the user on the list view of models saved in the database

Uses the Model Management Library.

Priority: Medium.

Relates to use case IDE-UC01 and features IDE-F002, IDE-F041.

IDE-F043: Run a model or an instance selected by the user on the list view of models saved in the database

Uses the ORML Interpreter.

Priority: Medium.

Relates to use cases IDE-UC06, IDE-UC07 and feature IDE-F006.

IDE-F044: Save a model to the database

If:

- a model with the same name already exists in the database,
- that model has instances,
- the parameter signatures of the current and existing model are different,
- and the user has confirmed the model should be overwritten,

the existing model and all its instances will be deleted prior to saving; otherwise, the model shall not be saved and the existing model and instances shall be left untouched.

Uses the Model Management Library.

Priority: High.

Relates to use case IDE-UC09.

IDE-F045: Save a model instance to the database

Uses the Model Management Library.

Priority: High.

Relates to use case IDE-UC04.

IDE-F046: Save a model to the filesystem

Priority: High.

Relates to use case IDE-UC09.

Feature Group: Support Features

IDE-F061: Convert the code or code+documentation representation of a model to a human-friendly format

Uses the Interpreter Exporting library.

Priority: Medium.

Relates to use case IDE-UC10 and feature IDE-F028

IDE-F062: Convert an ORML code file to a code+documentation XML file

Priority: Low.

Relates to use cases IDE-UC01, IDE-UC02.

IDE-F063: Open a code+documentation file and load it in the documented code model editor

Priority: Low.

Relates to use cases IDE-UC01.

IDE-F064: Convert an ORML code file to a code+documentation XML file

Priority: Low.

Relates to use cases IDE-UC01, IDE-UC02 and feature IDE-F028.

IDE-F065: Print a model's text

Formatting should be kept and pagination (with header and/or footer) should be taken into account.

Priority: Medium.

Relates to use case IDE-UC11.

Appendix C

Prototyping Results: WPF controls in Dynamics AX

The goal of this prototype was to create a simple custom control in WPF and assess the feasibility/difficulty of integrating it inside a Dynamics AX form.

Phases:

1. Reading about WPF;
2. Building a composed control that looks and behaves minimally like a code editor;
3. Research on how AX would be able to import this control;
4. Doing the necessary work to import the control into an AX form.

Results:

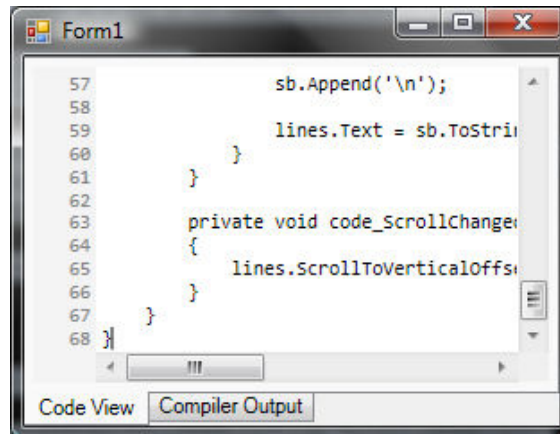
WPF is not quite a straightforward step from WinForms, but reading about its particularities and experimenting with XAML+C# can quickly yield a good understanding of the capabilities provided by WPF.

For example, scrolling two text boxes vertically side-by-side, something that in WinForms would take some Win32 interoperability (Interop) in order to catch OS-level messages (because it's possible to scroll using the scrollbar, the mouse wheel and even while moving up and down during editing), in WPF is a matter of wiring one event and putting one line of code inside.

```
private void code_ScrollChanged(object sender, ScrollChangedEventArgs e) {  
    lines.ScrollToVerticalOffset(code.VerticalOffset);  
}
```

Prototyping Results: WPF controls in Dynamics AX

The simple control looks like this:



After asking around the office (and consulting “Inside Microsoft Dynamics AX 4.0” a bit) it was clear that the way to integrate external controls into AX would be exporting those controls as ActiveX controls so that AX would be able to instantiate them.

WPF controls aren't ActiveX / COM enabled out of the box, so a kind of adapter control was used to bridge the gap: `System.Windows.Forms.Integration.ElementHost`¹. This WinForms control that ships with WPF specifically targets the issue of using a WPF control inside a WinForms application. Since all WinForms controls can easily be exported for ActiveX / COM interoperability, this control is the ideal candidate to use as a WPF container exported for COM Interop.

It was also possible to export the WPF control directly to COM Interop, but it would require the implementation of some methods to handle Win32 messages², a task that was already dealt with by the `ElementHost` control.

¹ Seen on <http://msdn2.microsoft.com/en-us/library/ms742215.aspx>.

² Seen on <http://msdn2.microsoft.com/en-us/library/ms744829.aspx>.

Appendix D

Prototyping Results: WPF User Interface Automation and Testing

This prototype was done based on the previous one. The objective was to assess the difficulty of preparing an application for automation testing with the new UI Automation library released with .NET Framework 3.0.

Conclusions drawn from making this prototype:

- In order to ease the automation aspects of finding certain controls and their state, special annotations should be included in the XAML of a WPF application so that it's controls are easily uniquely identifiable;
- When testing controls instead of isolated applications, special care should be taken in order to keep the automation code in a separate thread from the UI—the automation may not work otherwise (From <http://msdn2.microsoft.com/en-us/library/ms788709.aspx>: Because of the way Microsoft UI Automation uses Windows messages, conflicts can occur when a client application attempts to interact with its own UI on the UI thread. These conflicts can lead to very slow performance or even cause the application to stop responding.);
- The control patterns can not only be used to abstract the control types but can also be a very good starting point for user interface fuzzy testing.

The automation code is fairly simple, especially in this case that the window of the application being tested can be instantiated directly in the test. The following pieces of code exemplify a (not very efficient, but functional) way to access the controls and use properties and control patterns to interact with them.

Prototyping Results: WPF User Interface Automation and Testing

```
public class App
{
    private static MainWindow appWindow;
    private static Action finalizeWindowOpenedEventHandler;
    //...
}
```

```
var ui = new Thread(
    () =>
    {
        new Application();
        appWindow = new MainWindow();
        Application.Current.Run(appWindow);
    });
ui.SetApartmentState(ApartmentState.STA);
ui.Start();
```

```
// get an automation object to interact with the main window
IntPtr hMainWindow =
    (IntPtr)appWindow.Dispatcher.Invoke(DispatcherPriority.Send, new GetHandleDelegate(GetHandle),
    appWindow);
var window = AutomationElement.FromHandle(hMainWindow);

// get references to the two app tabs
var condition = new PropertyCondition(AutomationElement.AutomationIdProperty, "OutputTab");
var outputTab = window.FindFirst(TreeScope.Descendants, condition);
condition = new PropertyCondition(AutomationElement.AutomationIdProperty, "CodeTab");
var codeTab = window.FindFirst(TreeScope.Descendants, condition);

// do something
for (int i = 0; i < 3; i++)
{
    outputTab.SetFocus();
    Thread.Sleep(250);
    codeTab.SetFocus();
    Thread.Sleep(250);
}

// switch othe tabs that has the button we want to press
outputTab.SetFocus();
Thread.Sleep(500); // this is just for viewing purposes, not necessary when really testing

// we're going to open a modal dialog and then wait for it to close
bool dialogClosed = false;
// set up the window opened event handler, to check when the dialog is opened after we press the button
AutomationEventHandler windowOpenedHandler = WindowOpenedAutomationEventHandler;
AutomationClass.AddAutomationEventHandler(WindowPattern.WindowOpenedEvent, window, TreeScope.Subtree,
    windowOpenedHandler);
// when this event has finished its processing, here is what should be called
/* this is just a shortcut for: unregisterWindowOpenedEventHandler = delegate() {
AutomationClass.RemoveAutomationEventHandler(WindowPattern.WindowOpenedEvent, window,
    windowOpenedHandler); dialogClosed = true; }; */
finalizeWindowOpenedEventHandler =
    () =>
    {
        /*
        Stopwatch watch = new Stopwatch();
        watch.Start();
        AutomationClass.RemoveAutomationEventHandler(WindowPattern.WindowOpenedEvent, window,
        windowOpenedHandler);
        watch.Stop();
        Debug.WriteLine("Automation.RemoveAutomationEventHandler took " + watch.Elapsed + ", is it
        really necessary??");
        */
        AutomationClass.RemoveAllEventHandlers(); // much MUCH quicker than
        RemoveAutomationEventHandler!
        dialogClosed = true;
    };
```

(continues in the next page)

Prototyping Results: WPF User Interface Automation and Testing

```
// find the button that opens our dialog and press it
condition = new PropertyCondition(AutomationElement.AutomationIdProperty, "AXLogin");
var axLogin = outputTab.FindFirst(TreeScope.Descendants, condition);
if (axLogin != null)
{
    axLogin.SetFocus();
    Thread.Sleep(200);
    var pattern = axLogin.GetCurrentPattern(InvokePattern.Pattern) as InvokePattern;
    if (pattern != null)
    {
        pattern.Invoke();
    } // else fail
} // else fail
```

```
private static void WindowOpenedAutomationEventHandler(object src, AutomationEventArgs e)
{
    var messageWindow = (AutomationElement) src;
    // find the close button on the dialog window and press it
    var condition = new PropertyCondition(AutomationElement.AutomationIdProperty, "CloseButton");
    var messageBoxCloseButton = messageWindow.FindFirst(TreeScope.Descendants, condition);
    if (messageBoxCloseButton != null)
    {
        var pattern = messageBoxCloseButton.GetCurrentPattern(InvokePattern.Pattern) as InvokePattern;
        if (pattern != null)
        {
            Thread.Sleep(2000); // wait a bit for the user to "read" it :)
            pattern.Invoke();
        } // else fail
    } // else fail
    finalizeWindowOpenedEventHandler();
}

private static IntPtr GetHandle(Window window)
{
    return new WindowInteropHelper(window).Handle;
}
```

Appendix E

AOT-Like Models Tree Structure Definition

Tree structure

- Models [Menu#1]
 - Example Model [Menu#2]
 - Parameters
 - X : Integer
 - Y : Real
 - Z : String
 - Instances
 - Example Instance [Menu#3]
 - ◆ Arguments
 - X = 1
 - Y = 0.33
 - Z = “ESB-007”
 - Example Instance 2
 - ...
 - Example Model 2
 - ...

Context menus and command accelerators

Menu #1 – Models root

AOT-Like Models Tree Structure Definition

- New model [CTRL+N] – open a tab with a new model
- Refresh [F5] – reload the models list from the server

Menu #2 – Model node

- Edit – open the selected model in a new tab
- Rename [F2]
- Delete [DEL] – if there are instances of this model, the user is asked if it's ok to delete them too
- New instance [CTRL+SHIFT+N] (to differentiate from CTRL+N – new model) – pop up a dialog to edit the values of the arguments to the instance and the instance's name

Menu #3 – Instance node

- Run [SHIFT+F5] (to differentiate from global command F5 – run currently open model)
- Edit arguments – pop up a dialog to edit the values of the arguments to the instance and the instance's name
- Rename [F2]

Edit model – open a new tab with the model in which this instance is based