

NOTE TO USERS

This reproduction is the best copy available.

UMI[®]



ELIDE: AN INTERACTIVE DEVELOPMENT ENVIRONMENT
FOR THE ERASMUS LANGUAGE

MITRA NAMI

A THESIS
IN
THE DEPARTMENT
OF
COMPUTER SCIENCE & SOFTWARE ENGINEERING

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE
CONCORDIA UNIVERSITY
MONTRÉAL, QUÉBEC, CANADA

MAY 2009

© MITRA NAMI, 2009



Library and Archives
Canada

Published Heritage
Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et
Archives Canada

Direction du
Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file *Votre référence*
ISBN: 978-0-494-63179-9
Our file *Notre référence*
ISBN: 978-0-494-63179-9

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protègent cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

■♦■
Canada

CONCORDIA UNIVERSITY
School of Graduate Studies

This is to certify that the thesis prepared

By: **Mitra Nami**

Entitled: **ELIDE: An Interactive Development Environment for the
Erasmus Language**

and submitted in partial fulfillment of the requirements for the degree of

Master of Computer Science

complies with the regulations of this University and meets the accepted standards with
respect to originality and quality.

Signed by the final examining committee:

_____ Chair
Dr. Eusebius Doedel

_____ Examiner
Dr. Olga Ormandjieva

_____ Examiner
Dr. Constantinos Constantinides

_____ Supervisor
Dr. Peter Grogono

Approved by

Chair of Department or Graduate Program Director

_____ 20 _____

Dr. Robin A.L. Drew, Dean
Faculty of Engineering and Computer Science

Abstract

ELIDE: An Interactive Development Environment for the Erasmus Language

Mitra Nami

The process-oriented programming language Erasmus is being developed by Peter Grogono at Concordia University, Canada and Brian Shearing at The Software Factory in England. Erasmus is based on communicating processes. The latest version of the compiler is operating in command-line mode. As the compiler evolved, we recognized that there was a lack of an editor or an integrated development environment (IDE) for this new language. Our objective is to construct a suitable IDE for the Erasmus language called ELIDE by understanding the features of Erasmus language such as cells, processes, ports, protocols, messages, and message passing, which are the main heart of this programming language. At the same time we wanted to enable ELIDE with the features that are available in IDEs of languages like Ruby and Erlang. In this respect, after detailed studies on current text editors, IDEs and their features and evolution of IDEs, we designed and implemented an integrated development environment for Erasmus language.

To speed up the implementation process, we decided to choose one of the existing platforms as our base and develop Erasmus-specific features on top of it. There were many platforms available. Some of these platforms were under investigation and test. Among them we finally chose NetBeans. This thesis describes the development of this new tool for Erasmus programmers. It must be noted that the design of the ELIDE was an iterative process though what we present is the final result. ELIDE is a strong environment for a complete programming support for Erasmus language with built-in compile/debug/run ability. The most important features included in ELIDE are syntax coloring, Code folding,

Code completion, Brace matching, Coding tips, Indentation and Annotations. ELIDE is capable of adding more features later in case there is a need. Furthermore ELIDE can be used for an easy integration of editing and visualising support for Erasmus language building block such as cells, processes, ports, protocols, messages, and message passing.

We also conducted a preliminary user survey of Erasmus and ELIDE involving a number of graduate students. The results were quite encouraging with respect to the group surveyed and current capabilities of Erasmus and newly designed ELIDE. This study confirmed that It was a must for the Erasmus language to have a customized IDE to empower Erasmus language capabilities as a process-oriented language teaching and research purposes.

Acknowledgments

First of all I would like to express and extend my sincere gratitude to my supervisor Dr. Peter Grogono for his support, encouragement and invaluable guidance and suggestions throughout my graduate study in Concordia University. His help has been very significant to my master studies. Also his careful research guidance has led to this research completion.

I am grateful to the members of my examining committee Dr. Constantinos Constantinides and Dr. Olga Ormandjieva who have generously contributed their time and expertise and provided me with very valuable comments. I should sincerely thank Dr. Joey Paquet for his advice and comments.

I would like to thank all of professors of Computer Science and Software Engineering Department at Concordia University whom I had a course with, for the knowledge they devoted to me in Software Engineering domain.

I am appreciative to Administrative Staff of Computer Science and Software Engineering Department at Concordia University, especially Graduate Programs Advisor Halina Monkiewicz for her advice.

Last but not least, I am also very grateful to my husband and children, who have sacrificed a lot and who have always been there with me, with their love, supports and understandings. I want also to thank my parents who have always inspired me in my educations and all steps of my life.

Contents

List of Figures	x
List of Tables	xii
1 Introduction	1
1.1 Contributions	1
1.2 Structure of the Thesis	2
2 Problem, Motivation and Proposal	4
2.1 Problem statement	4
2.2 Motivation	5
2.3 Proposal	5
3 Theoretical Background	7
3.1 Evolution of IDEs	7
3.2 Features of an IDE	9
3.2.1 User Interface	9
3.2.2 Single Document Interface	10
3.2.3 Multiple Document Interface	12
3.2.4 IDE-Style Interface	16
3.2.5 Dockable Child Windows	17
3.2.6 Collapsible Child Windows	17

3.2.7	Overlappable Windows	17
3.2.8	Tabbed Document Interface	18
3.2.9	Splitter Window	18
3.2.10	Modal and Non-modal Windows	19
3.2.11	Programming Capabilities	23
3.2.12	Protocol Support	28
3.3	Programming Environment Capabilities	28
3.4	Erasmus Language	29
4	Methodology	33
4.1	Developing an IDE	33
4.1.1	Creating an Editor	34
4.1.2	Registering a Type Extension	34
4.1.3	Building a Model Builder	34
4.1.4	Building Parsing Error Information	34
4.1.5	Managed Build System Integration	34
4.1.6	Toolkit Integration	35
4.2	Software Development Model	35
5	Design	36
5.1	Product Quality	37
5.1.1	Functional Requirements	39
5.1.2	Non-Functional Requirements	41
5.2	Use Case Diagram	44
5.3	State Chart Diagram	44
5.4	Class Diagram	46
5.5	Package Diagram	48

6	Implementation	50
6.1	Platform	50
6.2	User Interface	53
6.2.1	User Interface for Toolkit	53
6.2.2	User Interface for Editor	55
6.3	Software Development Model	58
7	Case Study	62
7.1	A Sample Client Server Program	62
7.2	Compilation and Execution	64
7.3	Verification	65
7.4	Validation	66
8	Evaluation	69
8.1	Preliminary Survey Results	70
8.2	Preliminary Survey Recommendations	71
9	Related Works	72
10	Conclusions and Future Works	74
	Bibliography	80
A	Implementation in Eclipse	88
B	Erasmus programs in ELIDE	93
B.1	Sample Program of Client Server 1: testodel	93
B.1.1	Source code	93
B.1.2	Execution	93
B.2	Sample Program of Client Server 2: testloop1	95
B.2.1	Source code	95

B.2.2	Execution	95
B.3	Program sample of Client Server 3: testmod1	97
B.3.1	Source code	97
B.3.2	Execution	97
B.4	Sample Program of Client Server 4: testextract	99
B.4.1	Source code	99
B.4.2	Execution	99
C	Current Text Editors and IDEs	101
D	Study Questionnaire	104
E	Grammar and Syntax of Erasmus Language	106

List of Figures

1	Evolution of IDEs (Source: [11,14])	8
2	Inkscape uses an SDI	11
3	Mac OS X uses SDI implementation	11
4	Adobe Photoshop uses MDI implementation	12
5	IDE-style Interface in RSS Bandit	16
6	Dockable and Undockable Windows in NetBeans	18
7	Splitter SDI Window	19
8	Splitter Static MDI Window	20
9	Splitter Dynamic MDI Window	20
10	Code Folding	24
11	Indentation-based Code Folding	25
12	Token-based Code Folding	25
13	Syntax Highlighting	26
14	A typical program in Erasmus language	31
15	Quality in the software lifecycle (source: [66])	37
16	Use Case Diagram	45
17	Top level State Chart Diagram	46
18	Class Diagram	47
19	Package Diagram	48
20	ELIDE Architecture using NetBeans	52

21	User interface in ELIDE	54
22	User Interface Design for tools	54
23	User Interface Design for Editor	56
24	Erasmus program with Code folding	57
25	Erasmus program with Code unfolding	58
26	Erasmus program with Coding tip, Annotations and Jump to certain point of program	59
27	Erasmus program with Coding tip and Comments	60
28	Erasmus program with no indentation	60
29	Erasmus program with indentation	61
30	Mapping file in XML format	63
31	Client-Server program in Erasmus	64
32	Graphical Configuration of the Client-Server program	65
33	ELIDE Architecture using Eclipse and Antler	89
34	Sample Erasmus program and grammar in ELIDE using Eclipse	91
35	Sample Program 1 Source- Testode1 in in ELIDE	94
36	Sample Program 1 Execution- Testode1 in ELIDE	94
37	Sample Program 2 Source- Testloop1 in ELIDE	95
38	Sample Program 2 Execution Testloop1 in ELIDE	96
39	Sample Program 3 Source Testmod1 in ELIDE	97
40	Sample Program 3 Execution Testmod1 in ELIDE	98
41	Program sample 4 Source- Testextract in ELIDE	99
42	Sample Program 4 Execution- Testextract in ELIDE	100

List of Tables

1	Some existing IDEs for C/C++	9
2	Some existing IDEs for Java	10
3	Comparison of some existing IDEs and Editors based on User Interface . .	15
4	Classification of capabilities of existing programming environment.	29
5	Result of Usability Survey -Population of users	71
6	Result of Usability Study - Level of knowledge of users	71
7	list of Current Text Editors and IDEs	103

Chapter 1

Introduction

The study of IDEs has become one of the most important fields with major applications in the software engineering, debugging, high performance compiler integration, programming tools and security. In this thesis we present the design and implementation of an IDE for Erasmus with tool support which we call it ELIDE. Erasmus project is a process oriented programming language which is being developed by Peter Grogono at Concordia University and Brian Shearing at Factory in England. Like object oriented programming languages, Erasmus project provides both a framework and a motivation for exploring the interactions among its building blocks. This process oriented programming language is mainly based on cells, processes and their interactions, and provides a full control over these interactions, which contrast with the object models in which an object doesn't provide a full control over the sequences in which method calls and events may hit an object.

1.1 Contributions

The main contributions of this thesis can be summarized as follows:

- Literature review of current text editors and IDEs.
- Requirement analysis and design of an IDE for a process oriented language.

- Parallel development of the Erasmus IDE on both Eclipse and NetBeans open source platforms in order to compare the two different approaches and finally choose the best one.
- Implementing a strong environment for a complete programming support for Erasmus language with built-in compile/debug/run ability as well as added features such as syntax coloring, Code folding, Code completion, Brace matching, Coding tips, Indentation and Annotations.
- Demonstrate a proof of concept through a case study.
- Usability study of the IDE, based on the findings of a user survey.

1.2 Structure of the Thesis

This thesis is organized into ten chapters. Chapter 1 introduces the problem, Contributions and the structure of this thesis.

Chapter 2 will describe problem statement and the motivation for the research presented in this thesis. We will then describe our proposal and its benefits.

In chapter 3, we introduce the background knowledge we need to know in order to comprehend this research better.

In chapter 4, we will review features of a suitable IDE for the Erasmus language by studying current IDEs for programming languages.

In chapter 5 we will define the general framework of the design of our IDE, and subsequently, the detailed design of our architecture from different aspects.

In Chapter 6, we will discuss the implementation of a suitable IDE for the Erasmus language based on reviewing current IDEs for programming languages.

Then in chapter 7 we will consider a case study of a program in detail.

In chapter 8 we will present the result of usability study.

Then in the chapter 9 we will introduce related works on IDEs by separating them into two categories of language-based IDEs and toolkit-based IDEs.

Finally we discuss conclusion and future works in chapter 10.

A list of Abbreviations is presented in the Glossary section.

Appendix A will describe an alternative of ELIDE in Eclipse IDE. Then appendix B will show more sample programs in NetBeans. Appendix C compares current text editors and IDEs with their capabilities. Questionnaire for evaluation is presented in Appendix 4 and finally the grammar and syntax of Erasmus Language is included in Appendix 5.

Chapter 2

Problem, Motivation and Proposal

In this chapter, first we discuss the problems and motivations behind this research which constitutes the scope of this dissertation. Then we present our proposal to achieve our goals.

2.1 Problem statement

Like any other programming language, Erasmus programmers not only need a Graphical user interface (GUI) for developing their programs, but also they need to be able to rely on a tool support, in order to be able to develop critical applications. These applications mostly are concurrent, distributed and fault-tolerant. At the moment such environment or tool support for Erasmus language does not exist and the lack of appropriate processes and tools to help these programmers to engineer complex systems is obvious. The research proposed here will address this current gap. A GUI-based, integrated environment with the necessary tools to support developing Erasmus language is very essential. On the other hand without any tools, developing applications is a very pressing problem because with enriched Erasmus's grammar, as well as its concurrent and distributed programming capabilities, programmers might easily lose their track of control. One of main issues is first to create an IDE and second to add a tailor designed toolkit on Erasmus IDE. Mean time

we have to take into consideration that this language still is under development and some of its features are yet to be implemented, therefore the final product should be capable of incremental development. The desired IDE should have an easy way to guide the user through his implementation while allowing modifications or feedbacks, finally it should generate outputs that can be easily deployed. With this in mind, we tried to find out what Environment, tools or features might be needed for such language. We specified its functional and nonfunctional requirements.

2.2 Motivation

The above mentioned problems motivated us to investigate comprehension of Erasmus language and its features and subsequently, review existing Interactive Development Environments and their features and finally reason the core features of an IDE with tool support. Our objective in this thesis is to design a suitable IDE for the Erasmus Language based on above study that reflects recent researches in the language support capabilities. This includes designing an IDE with compile/debug/run ability. Some of features included in ELIDE are syntax coloring, Code folding, Code completion, Brace matching, Coding tips, Indentation and Annotations. With ELIDE we will then be able to provide IT-based assistance for Erasmus programmers to implement Erasmus building blocks such as cells, processes, ports, message passing and protocols in a consistent environment since these building blocks are the main heart of this process oriented language.

2.3 Proposal

In order to resolve the problems mentioned earlier, we propose an Integrated Development Environment for Erasmus that builds upon NetBeans, enhancing it with syntax and grammar checking, compile/debug/run facilities and Document generating. ELIDE will support the current Compiler, grammar and syntax of Erasmus Language as well as it is

easy to replace it with later versions of compiler. We have taken into account that this new language is still evolving, therefore ELIDE will be designed in such a way that will be able to accommodate new features easily.

The expected contributions of this proposal are to provide an environment for Erasmus programmers to gain comprehension over Erasmus programs. This can be done by means of an integrated set of tools that this proposal will deliver to manipulate and process Erasmus programs. Implementation of this proposal will allow Erasmus developers to get up to speed with Erasmus programming. The developed IDE will be released in Open Source format, to promote its adoption and further enhancement by the Erasmus developer community.

Chapter 3

Theoretical Background

The need for IDEs became apparent when programmers started to type their programs in front of console or terminal. Early languages did not have one, since they were prepared using flowcharts, coding forms, and keypunches before being submitted to a compiler. BASIC was the first language to be created with an IDE. Its IDE was command-based, and therefore did not look much like the menu-driven, graphical IDEs of today. However it helped users in editing, file management, compilation, debugging and execution similar to recent IDEs. In this chapter we review a brief history about evolution of IDEs, next we review available features of current IDEs and then we take a look at the classification of capabilities of current programming environment and finally we explain some features of Erasmus language.

3.1 Evolution of IDEs

IDEs provide typically large numbers of features for authoring, modifying, compiling, deploying and debugging software. But the idea is that the IDE abstracts the configuration necessary to piece together command line utilities in a cohesive unit, which theoretically reduces the time to learn a language, and increases developer's productivity. It is also thought

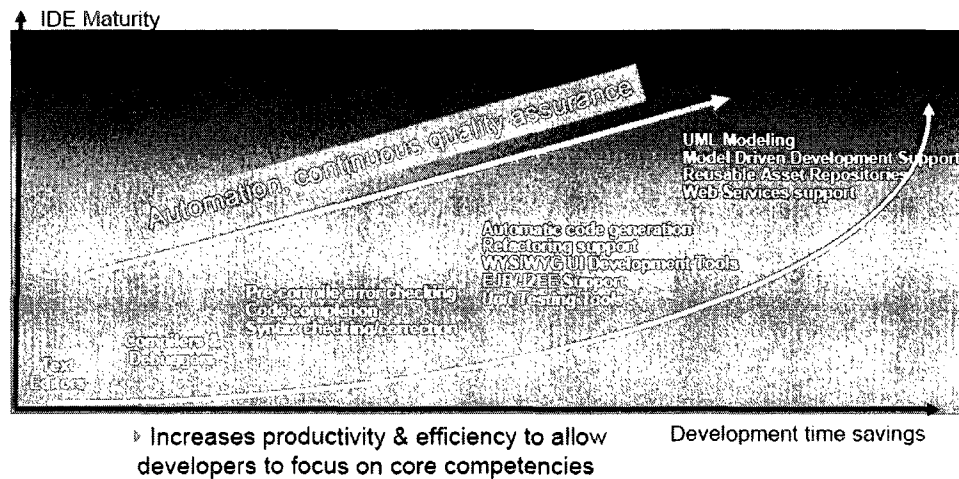


Figure 1: Evolution of IDEs (Source: [11,14])

that the tight integration of various development tasks can lead to further productivity increases (for example, code can be compiled while being written, providing instant feedback on syntax errors). Most modern IDEs are graphical but before the advent of windowing systems, IDEs were text-based, function keys or hotkeys were used to perform various tasks (Turbo Pascal is a common example).

The emergence and popularization of Open Source IDEs have played a critical role in IDE development for example we can refer to Eclipse and NetBeans. The combination of the Open Source philosophy with an open and extensible framework encourages the creation of a community of people to extend the capabilities of the IDE, allowing even exotic languages and applications to be supported by the environment.

Since there is growing interest in visual programming, IDEs are designed to allow users to create new applications by moving programming building blocks or code nodes to create flowcharts or structure diagrams which are then compiled or interpreted. These flowcharts often are UML. Figure 1 shows how IDEs evolved [14]. At first they were text editors, later more functionality were added to them. Then compilers and debuggers were added and later pre-compile error checking, code completion and syntax checking/recognition were among

Name	Developer	Latest Stable Release	OS	Cost
C++	Borland	2006	Windows	1090USD/2490USD
Dev-C++	Bloodshed Software	February 2005	Windows Windows	Free
Eclipse CDT	Eclipse Foundation	September 2006	Cross platform	free
NetBeans C/C++ pack	Sun Microsystems	October 2006	Cross platform	free
Turbo C++ 2006	Borland	2006	Windows	Free/399USD
Visual C++	Microsoft	November 2005	Windows	Free/299USD/ 799USD/10939USD
wxDev-C++ wxDev-C++	Guru Kathiresan	November 2006	Windows	Free

Table 1: Some existing IDEs for C/C++

new features. Later automatic code generation, refactoring support, UI development tools and unit testing tools were added. Recently UML modeling, model driven development support, web services support are added. As it can be seen the trend is toward more automation and continuous quality assurance and the results are increases in productivity and efficiency and development time saving [11].

Table 1 and 2 show some existing IDEs for C/C++ and Java are compared in terms of time of release and cost.

3.2 Features of an IDE

IDEs are usually designed with a number of features. In the following pages we review some of these important features.

3.2.1 User Interface

IDEs usually are built of many windows each of which relate to one feature. The facilities for managing these windows are very important. There are several representations which are as follows [56,57]:

Name	Developer	Latest Stable Release	OS	Cost
Eclipse	Eclipse Foundation	September 2006	Cross platform	Free
Jbuilder	Borland Software	2006	Cross platform	Free/499USD/3500USD
JCreator	Xinox Software	October 2006	Windows	free/69USD
JDeveloper	Oracle	January 2006	Cross platform	free
NetBeans	Sun Microsystems	October 2006	Cross platform	Free
Sun Java Studio Enterprise	Sun Microsystem	October 2006	Cross platform	Free
WebSphere Development Studio	IBM	January 2006	Windows	1000USD/4500USD

Table 2: Some existing IDEs for Java

3.2.2 Single Document Interface

In graphical user interfaces, a single document interface or SDI is a method of organizing graphical user interface applications into individual windows that the operating system's window manager handles separately. SDI window does not have a "background" or "parent" window containing its menu or toolbar; instead, each window contains its own menu or toolbar [64]. That is the reason that applications which allow the editing of more than one document at a time, e.g. word processors, may therefore give the user the impression that more than one instance of an application is open.

Often, each window is displayed as an individual entry in the operating system's task bar or manager. Some task managers summarize windows of the same application. For example, Mac OS X uses a feature called Expose which allows the user to temporarily see all windows belonging to a particular application. Figure 2 shows a sample Single Document Interface of Inkscape and figure 3 shows Single Document Interface of Max OS Expose.

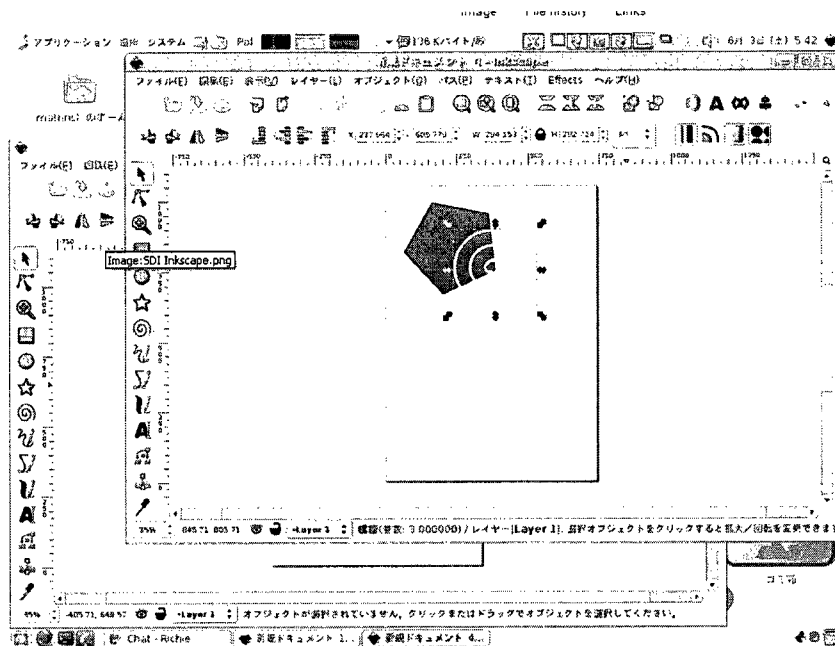


Figure 2: Inkscape uses an SDI

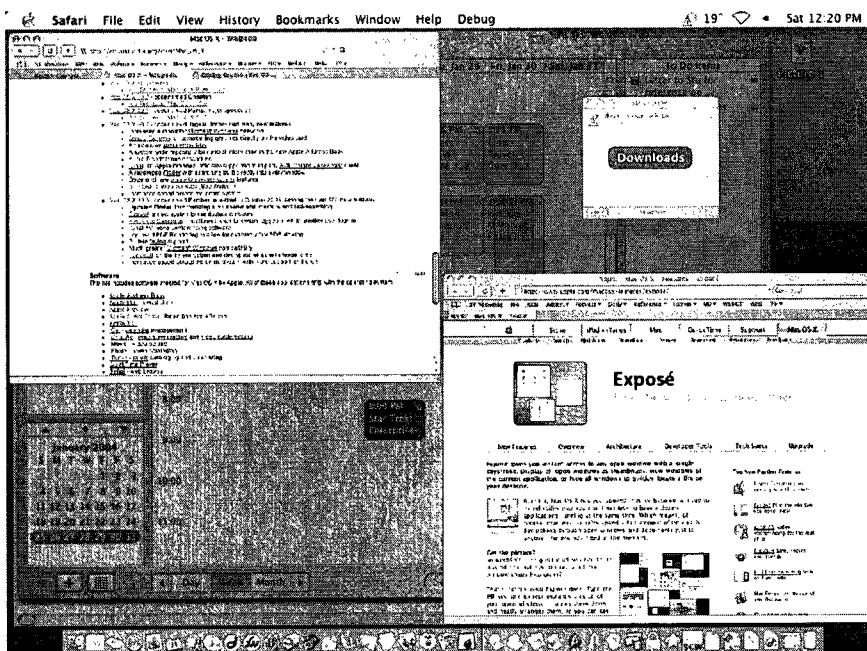


Figure 3: Mac OS X uses SDI implementation

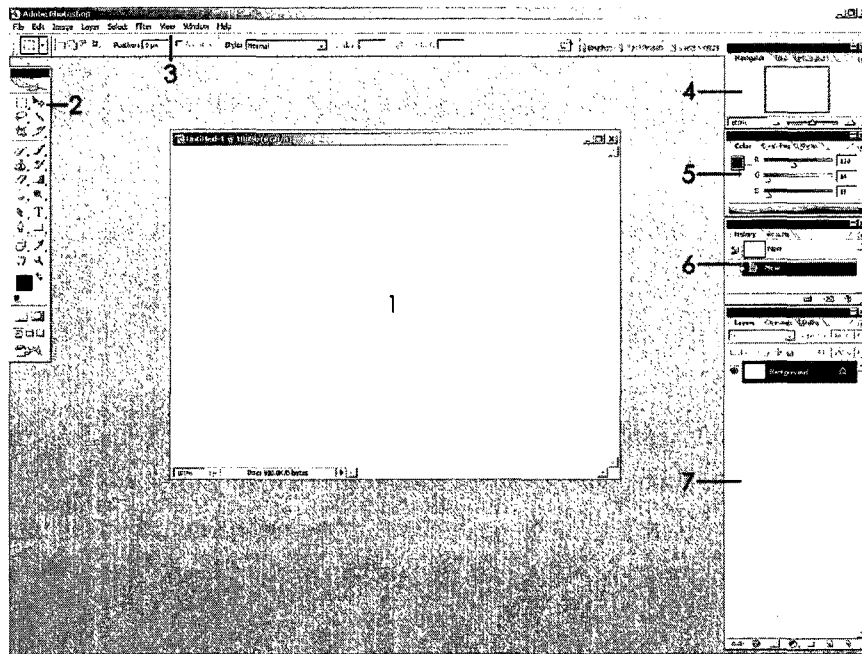


Figure 4: Adobe Photoshop uses MDI implementation

3.2.3 Multiple Document Interface

Graphical computer applications with a Multiple Document Interface (MDI) are those whose windows reside usually under a single parent window (exception is modal windows)[64]. In the usability community, there has been much debate over which interface type is preferable. Generally SDI is seen as more useful in cases where users work with more than one application. Companies have used both interfaces with mixed responses. For example, Microsoft has changed its office applications from SDI to MDI mode and then back to SDI, although the degree of implementation varies from one component to another [64]. Figure 4 shows Adobe Photoshop as a sample of Multiple Document Interface. Each numbered section identifies as:

- Document Workspace: This is the active image, where all work is done. The window may be resized, and the image can be zoomed in/out.
- Toolbar: Primary PS functions. The toolbar will be covered in more detail.

- Tool / Action Options: Shows options available for the currently active tool. Some tools share the same options.
- Navigator: Shows how much of the image is displayed, and what part of it.
- Colors: Manual color selection tool. Use sliders or type in values directly.
- History: Tracks what actions you have taken to provide a list of undo steps.
- Layers: The primary concept behind Adobe Photoshop is the layers system.

Advantages of MDI are as follows [56,57,64]:

- With MDI, a single menu bar and/or toolbar is shared between all child windows, reducing clutter and increasing efficient use of screen space.
- An application's child windows can be hidden/shown/minimized/maximized as a whole.
- Features such as "Tile" and "Cascade" can be implemented for the child windows.
- Possibly faster and more memory efficient, since the application is shared, and only the document changes. The speed of switching between the internal windows is usually faster than having the OS switch between external windows.
- Some applications have keyboard shortcuts to quickly jump to the functionality you need (faster navigating), and this doesn't need the OS or window manager support, since it happens inside the application.

Disadvantages of MDI are as follows [56,57,64]:

- There is the lack of information about the currently opened windows in MDI while in a SDI application, the currently opened windows is displayed. In order to view a list of windows open in MDI applications, the user typically has to select a specific menu ("window list" or something similar).

- MDI Can be tricky to implement on desktops using multiple monitors as the parent window may need to span both monitors.
- Virtual desktops cannot be spanned by children of the MDI. However, in some cases, this is solvable by initiating another parent window; this is the case in Opera, for example, which allows tabs/child windows to be dragged outside of the parent window to start their own parent window (on Windows).
- MDI can make it more difficult to work with several applications at once, by restricting the ways in which windows from multiple applications can be arranged together.
- Without an MDI frame window, floating toolbars from one application can clutter the workspace of other applications, potentially confusing users with the jumble of interfaces.
- The shared menu changes, which may cause confusion to some users.
- MDI child windows behave differently from those in single document interface applications, requiring users to learn two subtly different windowing concepts. Similarly, the MDI parent window behaves like the desktop in many respects, but has enough differences to confuse some users.
- Many window managers have built-in support for manipulating groups of separate windows, which is typically more flexible than MDI in that windows can be grouped and ungrouped arbitrarily. A typical policy is to group automatically windows that belong to the same application. This method can provide a solution without using MDI.

In recent years, applications have increasingly added “task-bars” and “tabs” to show the currently opened windows in an MDI application, which is used to see current windows. This type of interface is called “tabbed document interface” (TDI) which is explained in

Name	Comment
Internet Explorer 6	SDI
Visual Studio 6 development environment	MDI
Visual Studio .NET	MDI or TDI with "Window"
Firefox Mozilla	TDI by default,can be SDI instead
Opera	MDI combined with TDI
GIMP	Floating windows(limited MDI with plugin)
Adobe Photoshop	MDI in Windows XP version
Adobe Acrobat	Purely MDI until version 7.0.
MS Excel 2003	SDI
MS Word 2003	MDI
UltraEdit	Combination of MDI and TDI
Notepad++	TDI
PSPad	TDI
TextMate	TDI
Corel Wordperfect	MDI
Macromedia Studio under Windows	TDI and MDI
NetBeans	IDE-style
Eclipse	IDE-style
Visual Studio 6	IDE-style
RSS Bandit	IDE-style
JEdit	IDE-style
MATLAB	IDE-style

Table 3: Comparison of some existing IDEs and Editors based on User Interface

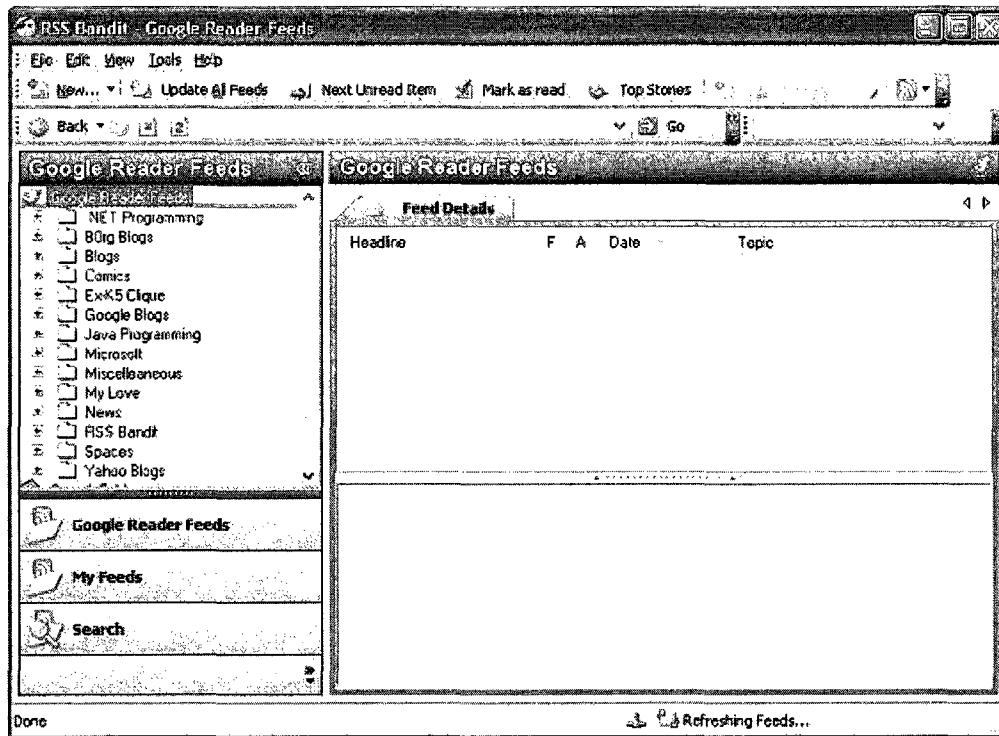


Figure 5: IDE-style Interface in RSS Bandit

the next topic. When tabs are used to manage windows, individual ones usually can not be resized. Comparison of some existing IDEs and editors based on their User Interface is shown in table 3.

3.2.4 IDE-Style Interface

IDE-style interface is the User Interface that has been used in recently-build IDEs. Graphical computer applications with an IDE-style interface are those whose child windows reside under a single parent window (usually with the exception of modal windows)[58]. An IDE-style interface is consisting of:

- Overlappable Windows,
- Tabbed Document Interface, and
- Window Splitting.

IDE-style window is distinguishable from a MDI interface, because all child windows in an IDE-style interface are enhanced with added functionality not ordinarily available in MDI applications. Because of this, IDE-style applications can be considered as a functional superset and descendant of MDI applications. Figure 5 shows an IDE-style interface.

Some of child-window functionality are as follows:

3.2.5 Dockable Child Windows

When panels are displayed as part of the main window, it is called Dockable. It is possible to undock each panel, so that it is displayed in a separate window. Once undocked, it is possible to dock the panel so it appears back in the main window again. It can be useful to undock a window when using [64]:

- a small screen;
- large tables, queries or forms; and/or
- More than one monitor.

Sample of dockable and undockable windows is shown in Figure 6.

3.2.6 Collapsable Child Windows

A common convention for child windows in IDE-style applications is the ability to collapse child windows, either when inactive, or when specified by the user. Child windows that are collapsed will conform to one of the four outer boundaries of the parent window, with some kind of label or indicator that allows them to be expanded again [64].

3.2.7 Overlappable Windows

It allows each opened document gets its own fully movable window inside the editor environment

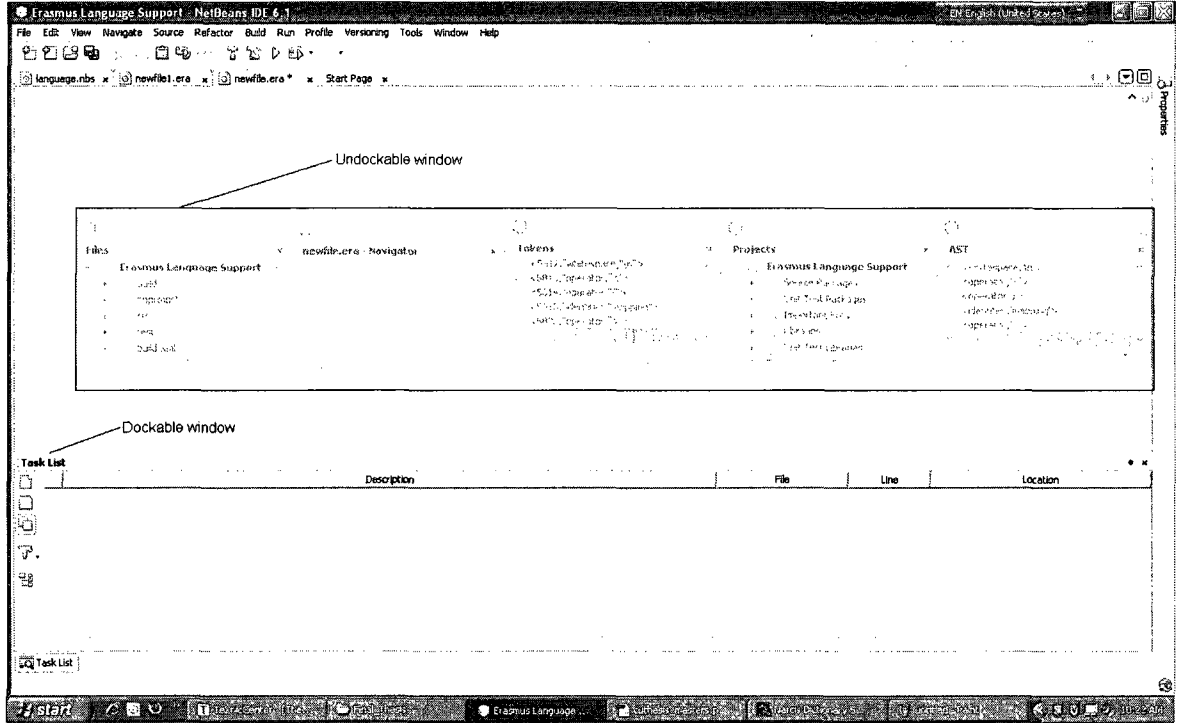


Figure 6: Dockable and Undockable Windows in NetBeans

3.2.8 Tabbed Document Interface

It allows multiple documents to be contained within a single window, using tabs to navigate between them. In contrast to MDI applications, which ordinarily allow a single tabbed interface for the parent window, applications with an IDE-style interface allow tabs for organizing one or more subpanes of the parent window. Examples are: Eclipse, Visual Studio 6, Visual Studio .NET, RSS Bandit, JEdit.

3.2.9 Splitter Window

A splitter window appears as a special type of frame window that holds several views in panes. The application can split the window on creation, or the user can split the window by choosing a menu command or by dragging a splitter box on the window's scroll bar [64]. After the window has been split, the user can move the splitter bars with the mouse to adjust the relative sizes of the panes. Splitter windows can be used in both SDI and

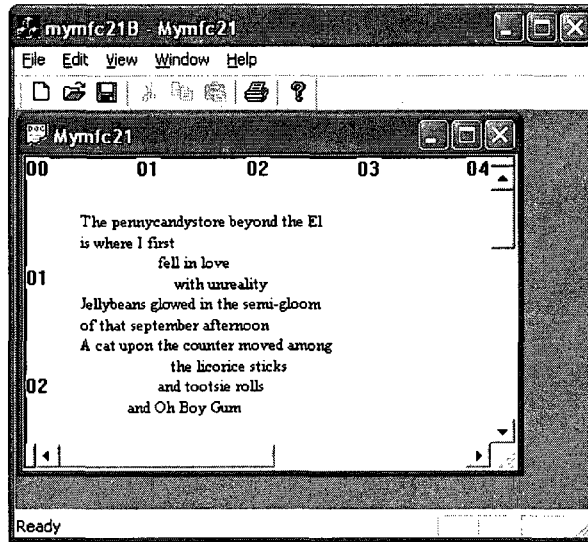


Figure 7: Splitter SDI Window

MDI applications. Examples of splitter windows for SDI and Static MDI and dynamic MDI application are shown in figures 7, 8 and 9. As is shown Splitter Window can be used to resize sub-panes of the parent window.

3.2.10 Modal and Non-modal Windows

A non-modal window does not restrict the user's interaction with other open windows on the desktop in any way. Using non-modal windows gives the user maximum flexibility to perform tasks within your application in any order and by whichever means they choose [61]. A modal window, while it is open, prevents the user from interacting with other windows in the same application (application modal), or in all applications, including the desktop itself (system modal)[61].

Advantages of the tabbed document interface are [56,57]:

- It holds many different documents logically under one window, instead of holding a large number of small child windows.
- Sets of related documents can be grouped within each of several windows.

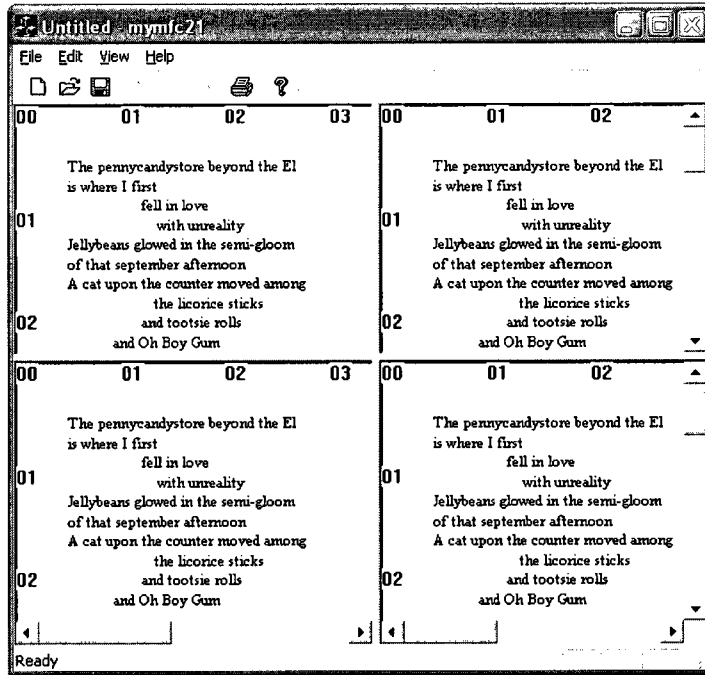


Figure 8: Splitter Static MDI Window

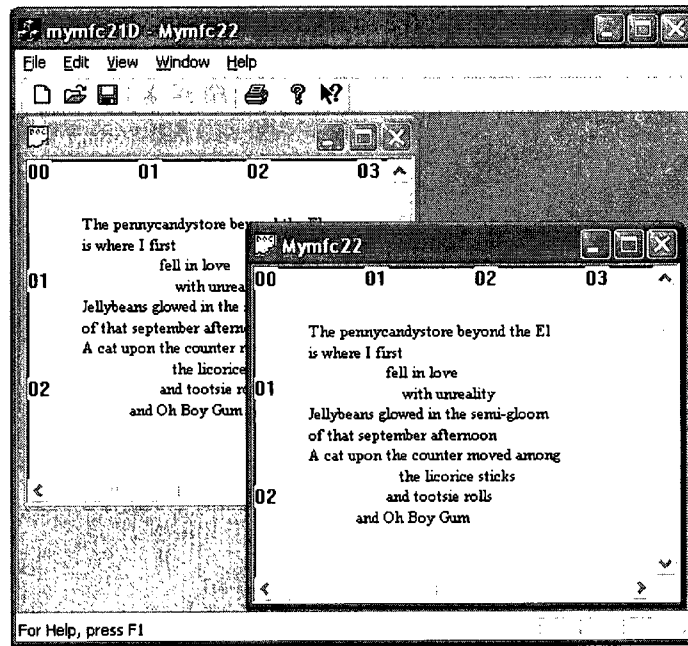


Figure 9: Splitter Dynamic MDI Window

- Using tabs instead of new windows to display content creates a smaller memory footprint and therefore reduces the strain on the operating system.
- Tabbed web browsers often allow users to save their browsing session and return to it later.

Although the tabbed document interface does allow for multiple views in one window, there are problems with this interface.

- One such problem is dealing with many tabs at once. When a window is tabbed to a certain number that exceeds the available resolution of the monitor, the tabs clutter up (this is the same problem as with SDI but moved to another place in the user interface).
- Multi-row tabs are a second issue that will appear in menu dialogs in some programs. Dealing with multiple rows of tabs in one window has two disadvantages that it creates excess window clutter and it makes some complications.
- Some people can have difficulty in finding a specific tab in a 3 or 4 level tabular interface. Part of the issue with this difficulty lies in the lack of any sorting scheme. Tabs can be strewn about without any sense of order, thus looking for a tab provides no meaningful understanding of a position to a tab relative to other tabs. Thus, although tabbed windows are adequate in environments where there is a minimal necessity for tabs (around ten tabs or less), this scheme does not scale, and alternate methods may be required to address this issue.

There are methods for addressing scalability of tabs:

- use more than one monitor.
- reduce the width of individual tabs, so that more can fit within the available one
- introduce scrolling to enable tabs to occupy a non-visible region of the screen

- introduce sections to spread tabs out to multiple areas

Large numbers of tabbed windows scale better with the tabs along the left or right edges of the window, instead of the top or bottom edges. That is because tab labels are usually much wider than they are tall. One can place tabs along the right window edge, and laid windows out in a vertical column, so each tab will be initially visible, and the user can use them to raise and lower the windows, drag them around in the column, or pull them out to anywhere on the screen.

Also tabbed window interfaces can give the user the freedom to position the tabs along any edge, so all four edges are available to organize different groups of tabs somehow that the user or application wants to see. Once one had tabbed views that he/she could stick onto the stack (represented as a “spike”), and then he/she can move the tabs to any edge. This will enable the users to position the tabs anywhere along any edge, and the tabs pops up pie menus with window management functions, to uncover and bury windows, etc.

TDI can be confusing for those who got used to SDI, MDI as windows can be hidden behind other windows. Some MDI applications lack a taskbar or menu to allow quick access to all windows, so in some cases a window can only be found by closing all others. On the other hand, since in TDI applications most tabs are visible and directly accessible, it is much harder for windows to get “lost”.

Also TDI windows must always be maximized inside their parent window, and as a result two tabs cannot be visible at the same time. This makes comparing of documents or easy copy-and-pasting between two documents more difficult. Full MDI interfaces allow for tiling or cascading of child windows, and do not suffer from these limitations. One example of an application that allows either TDI or MDI browsing is Opera. Using TDI by default, this application also supports full MDI and can also run as a SDI application.

In order to mitigate these problems, some IDEs, such as recent versions of XEmacs and Microsoft’s Visual Studio, provide a hybrid interface which allows splitting the parent window into multiple MDI-like “panes”, each with their own separate TDI tab set. The

Ion window manager does the same for the entire desktop. This provides many of the advantages of both MDI and TDI, although it can still be difficult for users to get used to. The Konqueror browser (available for the K Desktop Environment on UNIX and UNIX work-alike, such as Linux) also supports multiple TDI splits within the main window. Table 7 in appendix C shows some of text editors according to above classification. Aquamacs Emacs and GNU Emacs use a tabbed document interface using the tabbar plug-in while TextPad comes very close to SDI when configured to “allow multiple instances to run” and Vim version 7 supports a tabbed document interface. Table 7 in appendix C shows some of text editors according to editing feature.

3.2.11 Programming Capabilities

Table 7 in appendix C lists text editors according programming features. Brief explanations of some of the useful features are as follows [57,58,60,61]:

- Code Folding is one of important features of some of IDEs that allows the user to selectively hide and display sections of a currently-edited file as a part of routine edit operations [23,24]. This allows the user to manage large regions of potentially complicated text within one window, while still viewing only those subsections of the text which are specifically relevant during a particular editing session [23,24]. This feature is very useful among developers who manage a high volume and variety of source code files. Nonetheless, text editors that include this feature often provide enough flexibility to allow uses, for purposes other than source code management. In order to support code folding, the IDE must provide a mechanism for identifying folding points within a text file. Folding points are specified with one or more of the following conventions [23,24]:
 - Token-based folding points are specified using special delimiters that serve no other purpose in the text than to identify the boundaries of folding points.
 - Indentation-based folding points are specified by the position and sequence of

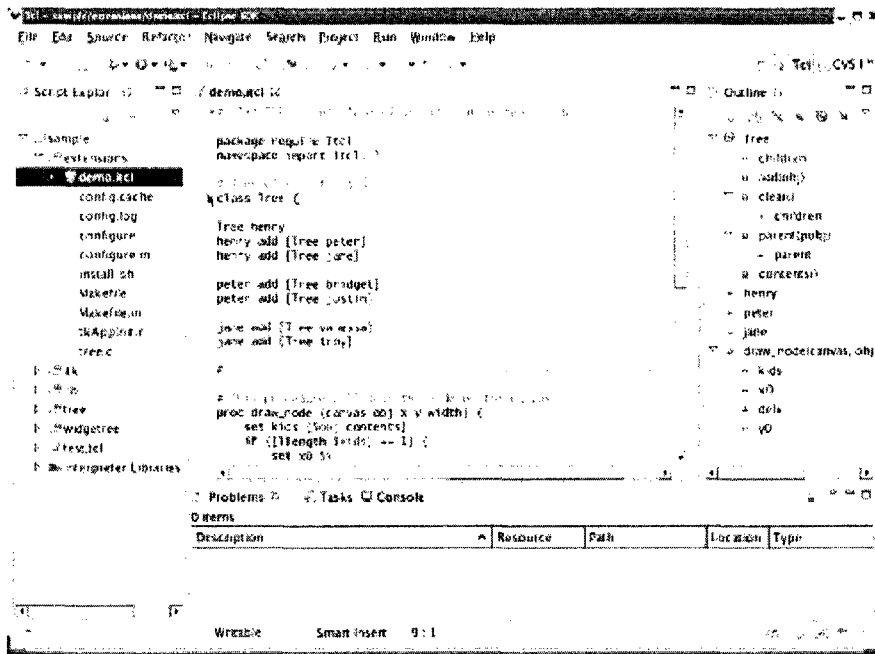


Figure 10: Code Folding

non-printing whitespace (such as tabs and spaces) within the text. This convention is particularly suitable to syntaxes and text files that require indentation as a rule by themselves.

- Syntax-dependent folding points are those that rely on the specific source code or programming language of the currently-edited file in order to specify where specific folding regions should begin and end.

Each of these conventions has its own distinct advantages and difficulties, and it is essentially up to the developer. When loaded into a folding editor, the outline structure will be shown (just the headings):

Usually clicking on the marks makes the appropriate body text appear as is shown in figure 10, 11 and 12. There are some IDEs and text editors that have these features for example recent versions of the open-source text editor Vim, Emacs, and the Java IDE Eclipse offer highly-configurable support for code folding. EditPad Pro supports

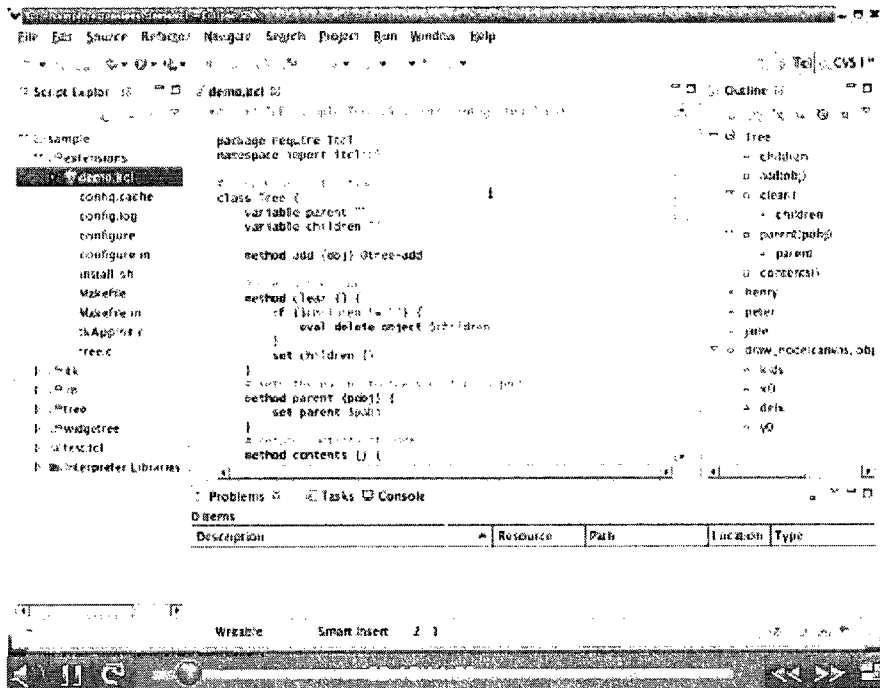


Figure 11: Indentation-based Code Folding

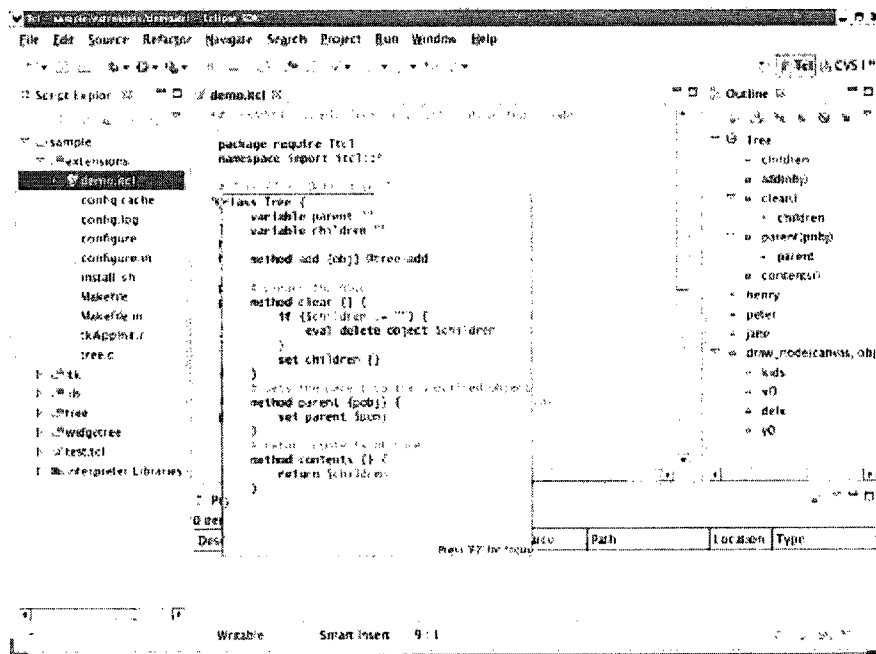


Figure 12: Token-based Code Folding


```

1 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
2 <html>
3 <head>
4 <title>Example</title>
5 <link href="screen.css" rel="sty
6 </head>
7 <body>
8 <h1>
9 <a href="/">Header</a>
10 </h1>
11 <ul id="nav">
12 <li>
13 <a href="one/">One</a>
14 </li>
15 <li>
16 <a href="two/">Two</a>
17 </li>

```

Figure 13: Syntax Highlighting

code folding through fully editable file navigation schemes. Sun’s NetBeans IDE, Microsoft’s Visual Studio.NET, the Code::Blocks IDE, the UltraEdit text editor, the Zeus IDE , the Mac OS X editors TextMate and BBedit, the KDE text editor Kate, Macromedia Dreamweaver, The cross-platform SciTE editor based on the Scintilla editing component, the GNOME IDE Anjuta and the general source code editor Notepad++ offer code folding as well [22,61].

Many text editors provide folding capability. Some of them are EditPad Pro, Emacs, EmEditor, Folding Text Editor, GridinSoft Notepad, jEdit, Kate, Kwrite, Notepad++, RJ Text Editor, SciTE, STET, TextMate, Vim, Visual Studio, XEDIT (however its folding was more a filtering) and Zeus for Windows IDE.

- Syntax highlighting is a feature of some text editors that displays source code text in different colors and fonts according to the category of terms. Figure 13 shows syntax highlighting.
- Multiple undo/redo is effective when there are multiple level of history otherwise when there is only one level of edit history remembered, by successively issuing the undo command, the last change will only “toggle”. Modern or more complex editors usually provide a multiple level history such that issuing the undo command repeatedly, will revert the document to successively older edits. A separate redo command will cycle the edits “forward” toward the most recent changes. The number of changes remembered depends upon the editor and is often configurable by the user.

- Rectangular block selection is useful for concentrating on a section of program. When a large section of code containing many compound statements nested to many levels of indentations is selected, it might be very helpful. In this case, there is a risk of losing track of block boundaries since by the time the programmer scrolls to the bottom of a huge set of nested statements, he may have lost track of which control statements go where.
- Bracket matching is a syntax highlighting feature that highlights matching sets of braces. The purpose is to help the programmer navigate through the code and also spot any improper matching, which would cause the program to not compile or malfunction.
- Auto indentation is used to format program source code in order to improve its readability.
- Auto completion involves the program predicting a word or phrase that the user wants to type in without the user actually typing it in completely. This feature is effective when it is easy to predict the word being typed based on those already typed, such as when there are a limited number of possible or commonly used words (as is the case with email programs, web browsers, or command line interpreters, or when editing text written in a highly-structured, easy-to-predict language, such as in source code editors). Auto completion speeds up human-computer interactions in environments to which it is well suited.
- Text folding is a similar feature used in folding editors and outliners. Text folding is generally distinguishable from code folding in that the latter tends to be used with the specific syntax of markup languages or programming languages, whereas the former can be used with ordinary text. Text folding is a solution for not losing track of braces which lets the developer hide or reveal blocks of code by their indentation level or by their compound statement structure.

- Compiler integration is used to allow running compilers/linkers/debuggers from within editor, capturing the compiler output and stepping through errors, automatically moving cursor to corresponding location in the source file.

3.2.12 Protocol Support

Protocol support has been important for some languages and IDEs offering web services. Table 7 in appendix C lists text editors according to protocol support or remote file editing over Internet Protocols. Brief explanations of some of these protocols are as follows:

- FTP: File transfer protocol, is a file transfer protocol for exchanging and manipulating files over any TCP-based computer network.
- HTTP: Hypertext Transfer Protocol is a communications protocol for the transfer of information on the Internet. It is used for retrieving inter-linked text documents
- SSH: Secure Shell, is a network protocol that allows data to be exchanged using a secure channel between two networked devices.
- WebDAV: Web-based Distributed Authoring and Versioning, allows users to collaboratively edit and manage files on remote World Wide Web servers.

3.3 Programming Environment Capabilities

Our study end up with three categories of existing programming support environments such as program editors, parser compilers, and IDEs as shown in table 3. A complete list of Editors with their characteristics is shown in Appendix C.

With program editors like Emacs, vi and JEdit, there is a need for independent line editor such as gdb, cvs, diff and find. There is no ability to have GUI and/or syntax analyzer. There are some language support such as syntax coloring and indentation and code folding. Also there is ability of defining the lexical analyzer.

Type	Name	Pros	Cons
Program editor	Emacs vi JEdit		needs independent command line tool
Parser Compiler	Antlr Javacc		
IDE	Eclipse	Creates a frame work of common components	
IDE	NetBeans	Creates a frame works	

Table 4: Classification of capabilities of existing programming environment.

Parser compilers such as Antlr and Javacc do not contain support for IDEs. They do not have the ability to be used as interpreter. They are not incremental and they provide weak support for error recovery. The positive point about them is that they are useful to define standards for describing a new programming language.

IDEs such as Eclipse creates a framework of common components for a new language to be added to them. This framework consists of a customizable editor, source control, a debugger framework and finally a debugger frame work. It is still a tedious task to create a full featured IDE by using Eclipse. Although Eclipse provides a rich support for many programming languages but its performance is influenced by number of listeners. Also adding a new language support is not simple and fast.

IDEs such as NetBeans allow defining a new language and their API is powerful but still involves 100 classes. In order to have Build/Run/Debug, we have to integrate the compiler into NetBeans platform. Also it does not provide any assistance for refactoring and semantic-based code completion or find usage.

3.4 Erasmus Language

As software development practices evolved, many interesting and crucial problems have been solved. These successes however created new challenges in computation. To resolve these new issues, various languages and techniques have been proposed, implemented and utilized. One of them was transition from procedural to object-oriented programming which

helped in implementation of new powerful software applications. The drawback was the complexity involved as a result of a change in their functionality or environment. As an example we can mention the difficulty involved in refactoring or software enhancement. Later with advancement in manufacturing microprocessors with multiple cores, there is now a need to write software to exploit hardware parallelism [34,35]. But concurrent programming is not easy because there is no suitable abstraction for expressing and controlling concurrency [36,37,38]. The process-oriented programming introduces a new approach to concurrent programming. The benefits of process-oriented programming are as follows [44]:

- Processes are more general to use.
- Processes are loosely coupled and have more autonomy as compared to objects since they only have effect on one another only on rare and brief communication.
- Processes can unlike object control over the way in which its methods are called because a process owns its thread of control and can choose when to communicate or not to communicate.
- In case when a Process operate only on its private data, it will be completely independent.

Hoare is the first for proposing communication sequential processes as a method of structuring programs [39,40,41,42]. The next process-oriented languages are Joyce, Erlang and Occam- π . Erasmus is the latest language based on communicating processes and concurrency in Erasmus is based on communicating processes. To get familiar with Erasmus, It is important to mention some definitions in Erasmus language. The basic building blocks of an Erasmus program are cells, closures and protocols. Closure is an autonomous process with its own state and instructions. Each closure may have parameters and communicate over synchronous channels satisfying some well-defined protocols [6]. A port serves as an interface for a closure to communicate with another closure. A cell is a collection of one or

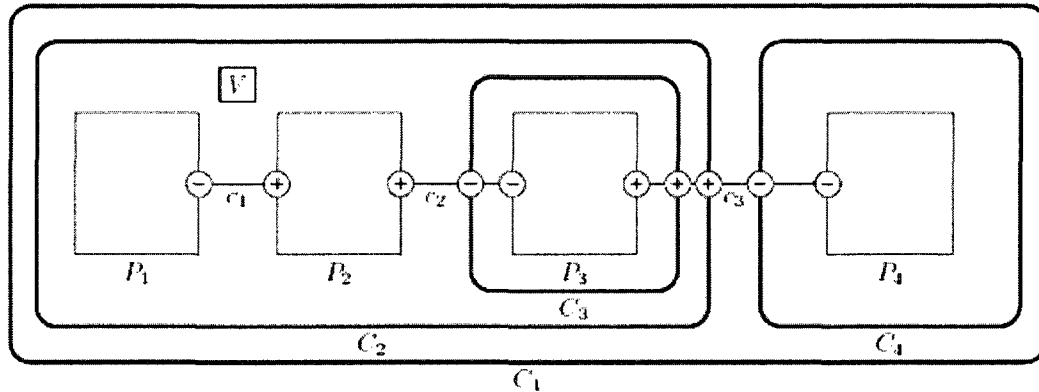


Figure 14: A typical program in Erasmus language

more closures. Protocols define constraints on messages that can be transferred with any port that is associated with the protocol. Cells and processes can be created dynamically [6]. Erasmus cells, processes and ports can be transmitted from one process to another and may be sent over a network. A program is a sequence of definitions followed by the instantiation of a cell. A visual presentation of a typical Erasmus program might look like as shown in figure 14.

In this example the outer cell, C_1 , corresponds to the “main program”, it consists of two nested cells, C_2 and C_4 . Processes P_1 and P_2 share a single thread of control and both have access to variable V . Cell C_2 has a third process, P_3 but, since P_3 is nested inside cell C_3 , it has its own thread and cannot access V . The processes communicate using the channels c_1 , c_2 , and c_3 . With respect to channel c_1 , process P_1 is a client (indicated by “-”) and P_2 is a server (indicated by “+”). Process P_3 acts as a client with respect to c_2 and acts as a server with respect to c_3 [5]. The specialty of Erasmus lies in the distributed programming. Transferring parts of programs, whole programs, parameters, cells, closures over synchronous channels satisfying some well-defined protocols would not need any recompiling, redesigning, but instead only its ports needed to be connected in its new environment.

To summarize, we can mention the main differences between Erasmus and object-oriented languages are as follows [44]:

- Coupling: In object oriented program, when an object is moved to another class, it will drag an unpredictable number of other objects along with it. But in Erasmus, its ports needed to be connected in its new environment.
- Protocols: By using protocols in Erasmus, we can specify asymmetries for example a client might need only a subset of services offered by server.
- Small Components: In object oriented programming small method are encouraged but not small classes, but Erasmus processes encourage small and reusable components.
- Localized Threads: In concurrent object-oriented programming when an object is active in multiple threads, it will be difficult to handle. But in Erasmus, first of all a thread is restricted to a single cell, therefore it avoids race conditions. Secondly control flow never crosses a cell boundary; therefore it has more autonomy and full control.
- Abstraction level: Erasmus programs can be easily transformed to object oriented language but not vice versa [6].
- Message passing: Erasmus processes exchange data by passing messages to one another. It loosens coupling, tightens encapsulation, provides flexible interfaces, and provides the potential for increase security [6].

The complete Grammar and syntax of Erasmus language is included in Appendix E.

Chapter 4

Methodology

Convenience of Integrated Development Environments (IDEs) is virtually unquestioned, but they are very large projects and typically require several man-years to create them. Until very recently, IDEs were products of large corporation such as IBM, SUN, Microsoft, etc. In this chapter we present the methodology used in developing IDEs.

4.1 Developing an IDE

For creating an IDE for a new language, the following steps are needed [47,31]:

- Creating an Editor,
- Registering a type extension,
- Building a Model Builder,
- Building Parsing Error Information,
- Managed Build System Integration (MBS), and
- Toolkit Integration.

4.1.1 Creating an Editor

The first step in integrating a new language is building an editor for the new language in which you can type your program [47].

4.1.2 Registering a Type Extension

The second step is to register a filename extension. So that it can be identified what file corresponds to the source code of supported language [47].

4.1.3 Building a Model Builder

The third step consists of construction of a parser and structural information for the new language. Model Builder will construct an Abstract Syntax Tree (AST) and can be built on a tree view [47].

4.1.4 Building Parsing Error Information

The fourth step consists of building parsing error information which shows the console view and red markers appearing in the editor. Basic information are filename, line number, error description and use them to populate problem view. Output of error messages of compiler will appear in the problem view with corresponding red markers appearing in the editor. So the parsing error information is created by scanning the output of the compilation and extracting the filename, line number, and error description, and use this information to populate the problem view [47].

4.1.5 Managed Build System Integration

Managed Build System (MBS) is not a necessary step but it is useful when users prefer not to maintain their own make files; they would rather have a “makefile” be automatically generated and automatically updated as source files are added to and removed from their

project. Therefore MBS is responsible for maintaining “makefiles” in these projects, tracking dependencies and updating “makefile” as project changes [47]. At this time We didn’t create MBS for Erasmus since we didn’t want to handle make file automatically.

4.1.6 Toolkit Integration

Some IDEs have build-in toolkits to assist their users. Developers benefit from using them in their development process. These tools, depending on the language needs, are different [56].

Some IDEs have simplified the process of developing new IDEs. Eclipse has simplified the creation of IDEs by creating a framework of common components i.e. a customizable editor, source control, a debugger framework and project support even though creating a full feature IDE remain a hard task. On the other hand NetBeans simplifies the process by providing more freedom and more user-friendly environment.

4.2 Software Development Model

We have used waterfall lifecycle model of software development. We started with requirements analysis and produced System Requirements Document (SRS). Based on SRS, we designed the system architecture and refined it into detailed designs which were the foundation for the implementation phase; however, we frequently had to go back to the earlier phases to address the changes that resulted from feedback in the later phase.

In the next chapter we will review the design of ELIDE in more details.

Chapter 5

Design

In previous chapter we defined our methodology. In this chapter, we describe some of the key features and requirements that we have included in design of ELIDE. These features provide a basis for our design that will be explained in next sections. Our vision for ELIDE is to provide a teaching and research platform for Erasmus based programs. As mentioned earlier, we have three ultimate goals for developing ELIDE, (1) to provide a lightweight integrated environment for Erasmus developers, (2) to provide a platform to future Erasmus compiler developments and (3) to manage quality properly at each stage of the software lifecycle consisting internal quality, external quality and quality in use. To achieve the first goal, ELIDE must have integrated high level tools to support the iterative nature of program development including editing, checking, compiling and debugging. We integrated into ELIDE these programming tools. To achieve the second goal, ELIDE must provide a flexible and extendable framework so that other tools can be easily integrated in the future. To achieve the third goal, it is necessary to define these perspectives, the specification and evaluation of quality according to ISO/IEC 9126-1 [66]. In the following sections we define our product quality according to standards. Next we represent the formulated ELIDE features and requirements such as main use case diagram, state chart diagram, lass diagram and package diagram and functional and non-functional requirements. And finally, as part of software requirement specification, we provide a clear and precise description of the

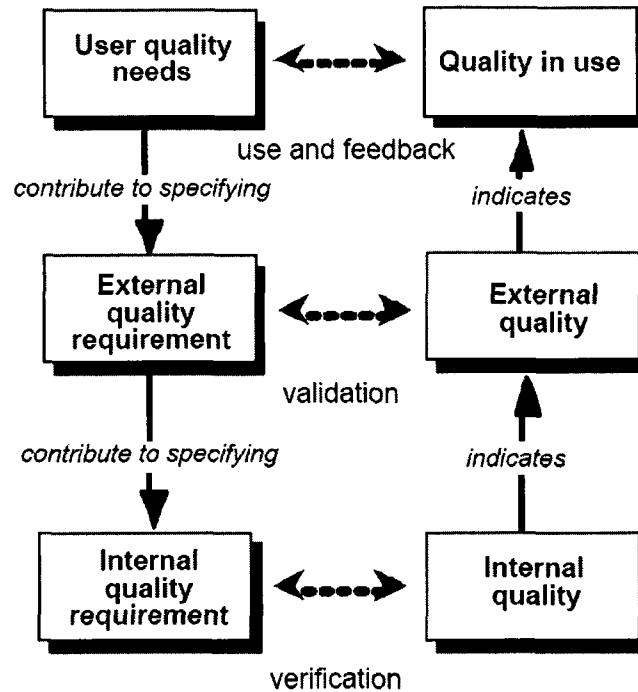


Figure 15: Quality in the software lifecycle (source: [66])

functional and non-functional Requirements according to ISO/IEC9126-1.

5.1 Product Quality

ISO/IEC 9126-1 defines goal of quality in the software lifecycle is to achieve the necessary and sufficient quality to meet the real needs of users [66]. Figure 15 shows the relation between User quality needs, External Quality Requirements, Internal Quality Requirements, Internal quality, External Quality and Quality in Use to achieve quality in the software lifecycle.

- “User quality needs can be specified as quality requirements by quality in use metrics, by external metrics, and sometimes by internal metrics. These requirements specified by metrics should be used as criteria when a product is validated and normally requires an iterative approach with continual feedback from a user perspective” [66].

- “External Quality Requirements specify the required level of quality from the external view. They include requirements derived from user quality needs, including quality in use requirements. External quality requirements are used as the target for validation at various stages of development. External quality requirements for all the quality characteristics defined in this part of ISO/IEC 9126 should be stated in the quality requirements specification using external metrics, should be transformed into internal quality requirements, and should be used as criteria in evaluation of product” [66].
- “Internal Quality Requirements specify the level of required quality from the internal view of the product including static and dynamic models, other documents and source code. Internal quality requirements can be used as targets for validation at various stages of development” [66].
- “Internal quality is the totality of characteristics of the software product from an internal view. Internal quality is measured and evaluated against the internal quality requirements. Details of software product quality can be improved during code implementation, reviewing and testing” [66].
- “External Quality is the totality of characteristics of the software product from an external view. It is the quality when the software is executed, which is typically measured and evaluated while testing in a simulated environment with simulated data using external metrics. During testing, most faults should be discovered and eliminated. However, some faults may still remain after testing. As it is difficult to correct the software architecture or other fundamental design aspects of the software, the fundamental design usually remains unchanged throughout testing” [66].
- “Quality in Use is the users view of the quality of the software product when it is used in a specific environment and a specific context of use. It measures the extent to which users can achieve their goals in a particular environment, rather than measuring the properties of the software itself” [66].

5.1.1 Functional Requirements

In this section we define the Functional Requirements for actor programmer. Functional requirements define the internal workings of the ELIDE in other words what ELIDE must be able to do and they are included in the use case Diagram. These Functional requirements are set by Erasmus language designer and programmers. They are categorized in 5 groups of Editing facilities, Erasmus Syntax and Grammar facilities, Build facilities, Run facilities and Documents facilities. The Editing facilities are gifted by platform used but the other four facilities must be designed and implemented. We used the notion (FR.*) for identifying the functional requirement number (*).

- Editing facilities
 - FR.1 Create New Project.
 - FR.2 Open Existing Project.
 - FR.3 Save Project.
 - FR.4 Close Project.
 - FR.5 Add New File to Project.
 - FR.6 Add Existing File to Project.
 - FR.7 Remove File From Project.
 - FR.8 Open File
 - FR.9 Close File
 - FR.10 Enter Text
 - FR.11 Edit Text
 - FR.12 Paste Text
 - FR.13 Undo Edit
 - FR.14 Redo Edit

- FR.15 Copy Text
- FR.16 Select All
- FR.17 Find/Replace
- Erasmus Syntax and Grammar facilities
 - FR.18 Check Erasmus Syntax and Grammar
- Build facilities
 - FR.19 Build
- Run facilities
 - FR.20 Run
- Document handling facilities
 - FR.21 View Graphical Representation
 - FR.22 View XML file
 - FR.23 View Tree view
 - FR.24 View Error
 - FR.25 View Coding tip
 - FR.26 Perform Code folding
 - FR.27 Perform Code unfolding
 - FR.28 Choose Code completion
 - FR.29 Open Help
 - FR.30 Close Help
 - FR.31 Indent
 - FR.32 View Annotation
 - FR.33 Jump to certain point of program

5.1.2 Non-Functional Requirements

In this section we define the Non-Functional Requirements for actor programmer. Non-functional requirements specify something about the system itself, and how well it performs its functions. In other words they specify criteria that can be used to judge the operation of ELIDE. These functional requirements are listed in the order according to ISO/IEC 9126-4 [69]. The user need weights are set either by Erasmus language designer and programmers or have been adopted by comparing to other IDEs. The weights are expressed in the High/Medium/Low manner or using the ordinal type scale in the range 1-9. Therefore we can conclude low is between 1 and 3, medium is between 4 and 6 and high is between 7 and 9. We used the notion (NFR.*) for identifying the non functional requirement number (*).

- NFR.1 Functionality: ISO/IEC 9126-1 defines functionality as “the capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions.” [66]. ELIDE has to encompass the entire process of syntax checking, syntax coloring, Code folding, Code completion, Brace matching, Coding tips, Indentation, Annotations, parsing, compiling and executing of Erasmus programs. ELIDE should support the Erasmus language and its compiler. The user needs weight was set as high.
- NFR.2 Reliability: ISO/IEC 9126-1 defines reliability as “the capability of the software product to maintain a specified level of performance when used under specified conditions.” [66]. The Architecture and Language Definition Files ensure that ELIDE always uses the Erasmus compiler to interpret and execute the Erasmus instructions in the intended manner. The user needs weight was set as medium.
- NFR.3 Usability: ISO/IEC 9126-1 defines usability as “The capability of the software product to be understood, learned, used and attractive to the user, when used under specified conditions.” [66]. ELIDE will be familiar to anyone with experience in developing program with IDEs. There is also a help system to improve usability for

less experienced users. The user needs weight was set as medium.

- NFR.4 Understandability: ISO/IEC 9126-1 defines understandability as “The capability of the software product to enable the user to understand whether the software is suitable, and how it can be used for particular tasks and conditions of use.” [66]. The user needs weight was set as medium.
- NFR.5 Learnability: SO/IEC 9126-1 defines learnability as “The capability of the software product to enable the user to learn its application.” [66]. The interface is similar to the standard NetBeans IDE interfaces or what can be seen in other IDEs. The user needs weight was set as low.
- NFR.6 Operability: SO/IEC 9126-1 defines operability as “The capability of the software product to enable the user to operate and control it.” [66].The user needs weight was set as high.
- NFR.7 Efficiency: ISO/IEC 9126-1 defines efficiency as “The capability of the software product to provide appropriate performance, relative to the amount of resources used, under stated conditions.” [66]. The User Needs weight was set as high.
- NFR.8 Maintainability: ISO/IEC 9126-1 defines maintainability as “The capability of the software product to be modified. Modifications may include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications.” [66]. The user needs weight was set as medium.
- NFR.9 Portability: ISO/IEC 9126-1 defines portability as “The capability of the software product to be transferred from one environment to another.” [66]. ELIDE is a Windows application programmed in Java and can be run on any computer that runs Windows and NetBeans environment. The user needs weight was set as high.
- NFR.10 Adaptability: ISO/IEC 9126-1 defines adaptability as “The capability of the software product to be adapted for different specified environments without applying

actions or means other than those provided for this purpose for the software considered.” [66]. ELIDE should facilitate adaptation to faults and other runtime changes. This would ensure that the IDE deals with runtime aspects as well. The user needs weight was set as high.

- NFR.11 Installability: ISO/IEC 9126-1 defines level of installability as “The capability of the software product to be installed in a specified environment.” [66]. The user needs weight was set as low.
- NFR.12 Performance: ISO/IEC 9126-1 defines level of performance as “Level of performance is the degree to which the needs are satisfied, represented by a specific set of values for the quality characteristics.” [66]. The management and editing features of ELIDE will be quick and responsive. Debugging relies on emulation, where the performance is heavily based on the speed of the emulating machine. The user needs weight was set as high.
- NFR.13 Scalability: ISO/IEC 9126-1 defines scalability as “ The internal capacity (e.g. screen fields, tables, transaction volumes, report formats, etc.)” [66]’. ELIDE should cater to different versions or CVS. The user needs weight was set as medium.
- NFR.14 Legality: ELIDE is intended to be released as a free product for non-commercial use. The user needs weight was set as low.
- NFR.15 Security: ISO/IEC 9126-1 defines security as “The capability of the software product to protect information and data so that unauthorized persons or systems cannot read or modify them and authorized persons or systems are not denied access to them” [66]. Given that ELIDE deals with development and even deployment of new functionality, there is also a need for a clear access control mechanism. This would ensure that only authorized users are allowed to perform such operations. The user needs weight was set as high.

In next section we will present the use case diagram for ELIDE to show its behavior as it responds to the requests that originate from outside of ELIDE.

5.2 Use Case Diagram

The top level use case diagram for ELIDE is shown in figure 16, the main use cases are as follows:

- Edit: edits one or more Erasmus Source code files.
- Check Erasmus Syntax: checks an Erasmus source code file for syntax and type errors.
- Compile (build): Compiles source code file with Erasmus compiler.
- Run: Executes a compiled Erasmus program.
- Document handling: shows and handles with AST, XML, Graphical Representation and Tokens files of each compiled Erasmus program.

The primary actor is the user of ELIDE who is mainly an Erasmus programmer. The secondary actors are Erasmus Compiler, NetBeans Platform and NetBeans Schliemann. In the next section, we will describe the functional behavior of ELIDE and we will present it in the State Chart Diagram.

5.3 State Chart Diagram

Figure 17 shows the top-level state chart diagram for ELIDE. It describes how the system interacts with the user by showing the events that initiate transitions from one system state to another. Essentially, the system allows the user to use all the functionalities of ELIDE (i.e. editing, checking, compiling, document handling or run). In the next section the Class Diagram is presented.

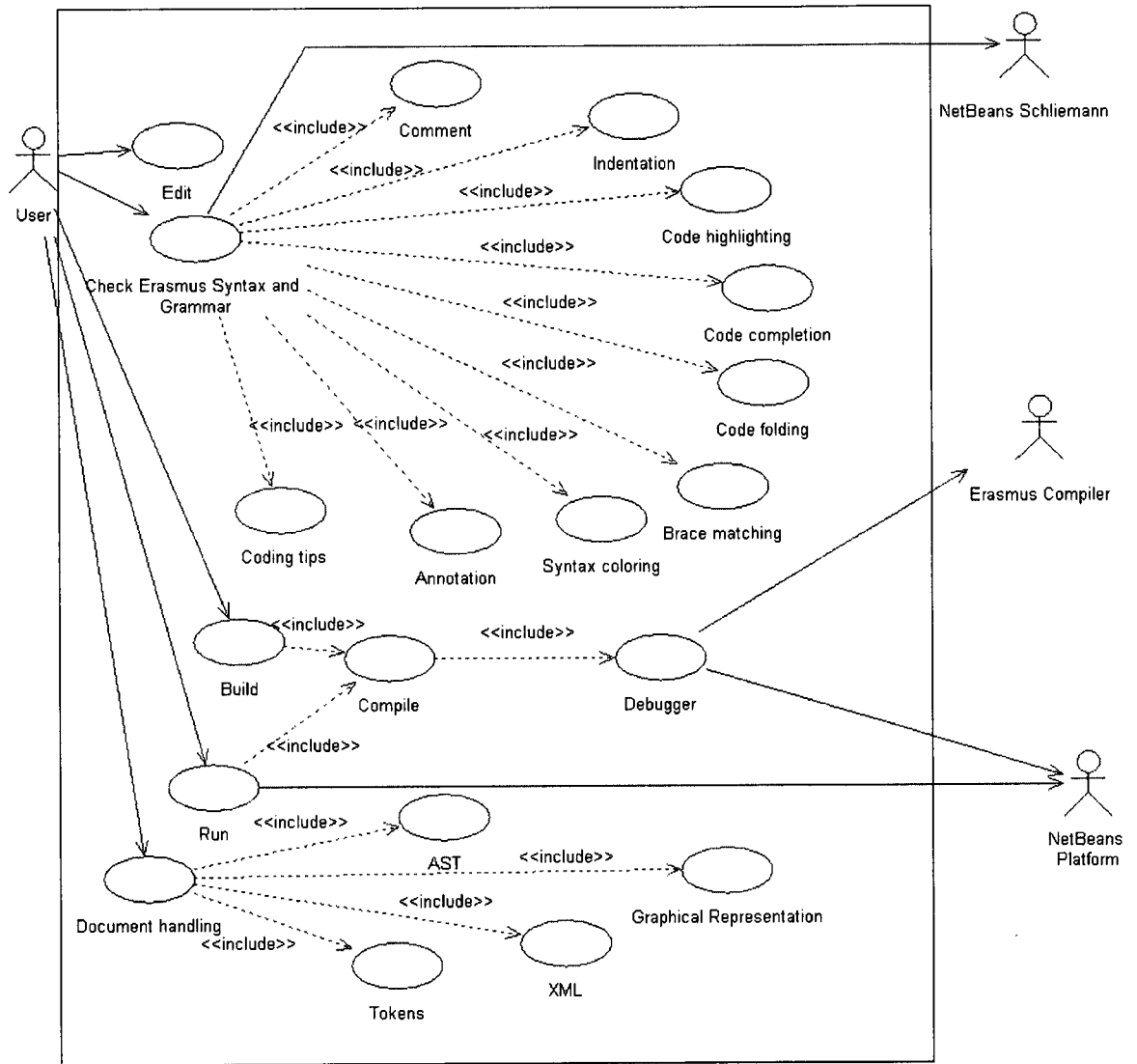


Figure 16: Use Case Diagram

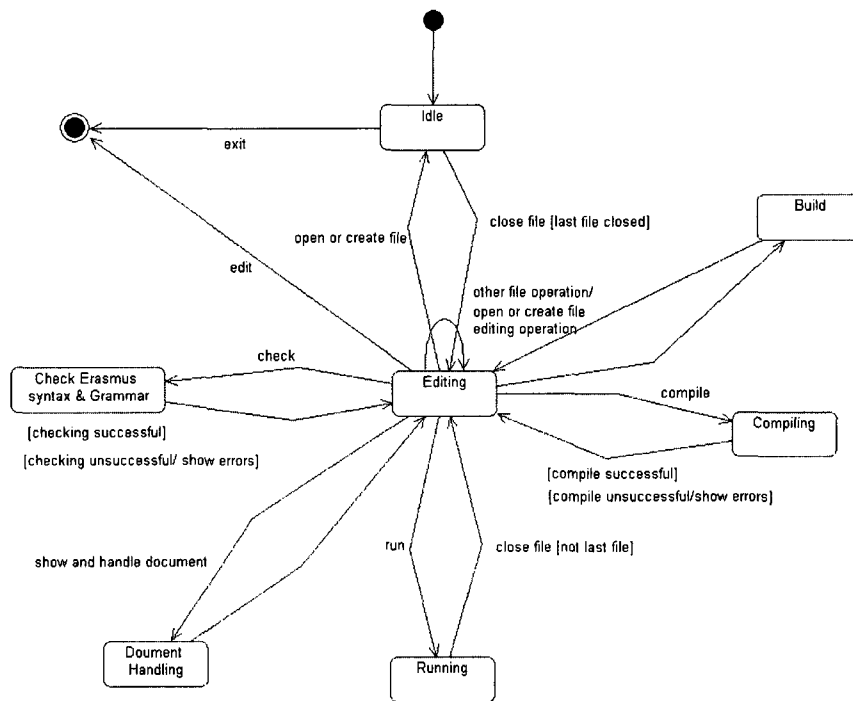


Figure 17: Top level State Chart Diagram

5.4 Class Diagram

The class diagram identifies the relationships among all major entities within ELIDE, and identifies their important methods and attributes. The class diagram provides a structural view of ELIDE that can be complemented with dynamic views in use case diagram. Also in the class diagram we try to describe the scope of ELIDE. In figure 18, the objects that directly interact with the users is shown. Every feature is made available after a project is opened or created. A project is a collection of files and settings that correspond with the program being developed. A reference is a macro, i.e. a set of commands, or a file. A file is an entity that can accept input from users and is used in the compilation and execution of a program. Files contain different types of input, which correspond to features of the Erasmus language such as labels, commands, and functions, as well as entities to support debugging, such as breakpoints and information on the values of variables. The debugger class uses the reference class to know which files exist in the current project, and it uses the

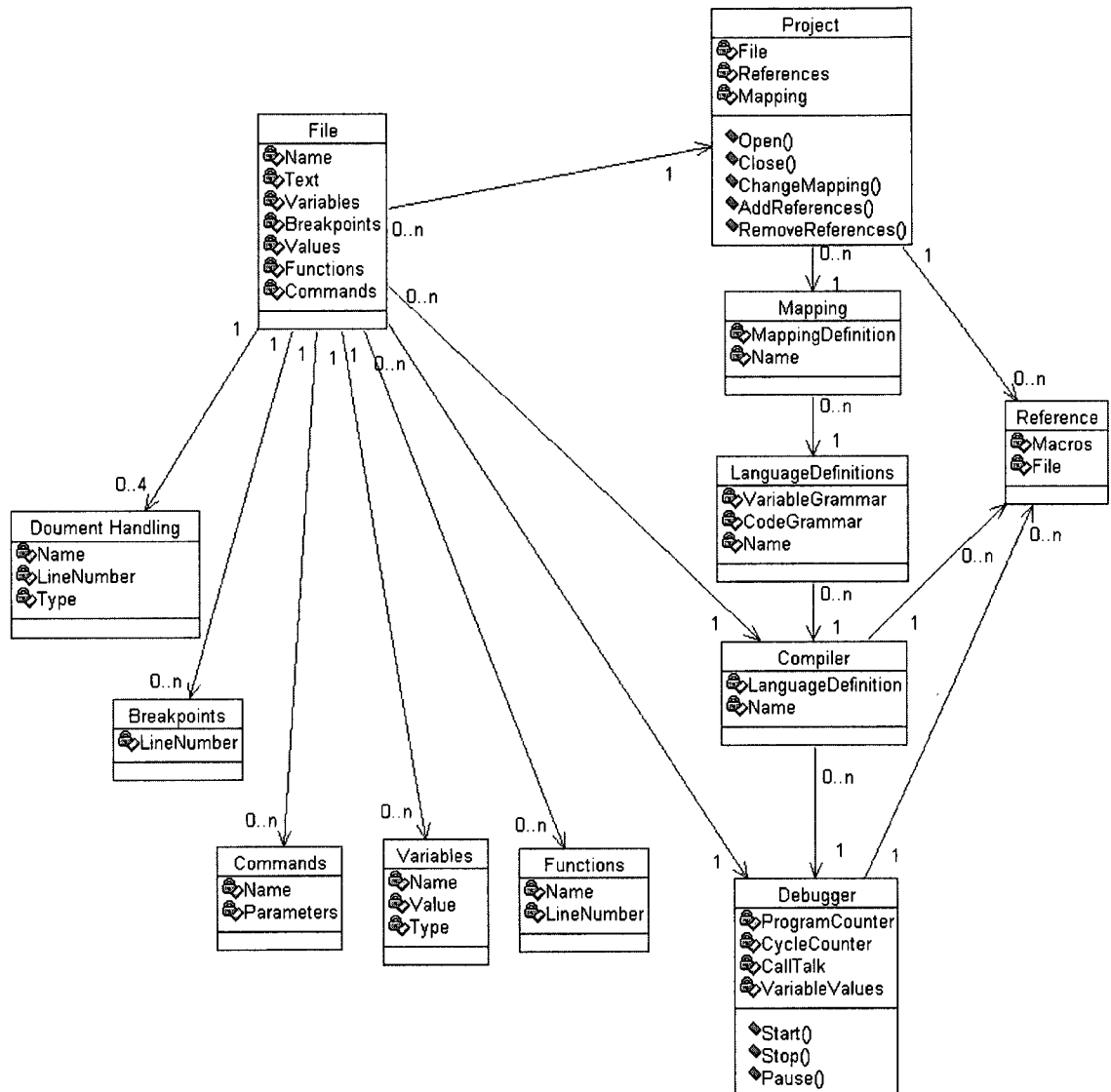


Figure 18: Class Diagram

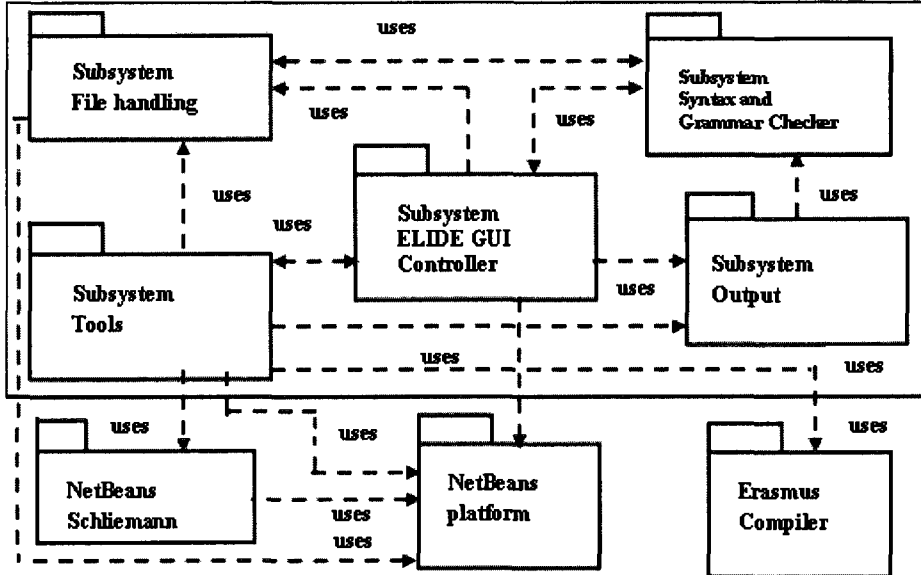


Figure 19: Package Diagram

file class to parse the code and creates and runs the program in debug mode. The mapping class defines the target processor for the project. The language definition class defines what each instruction for that specific Erasmus language does and is used to tell the debugger how to handle each command. In next section, we represent the package diagram.

5.5 Package Diagram

The ELIDE package diagram is shown in figure 19. The ELIDE consists of five major subsystems.

- File handling subsystem: browses files and directories and keeps track of previously opened files in ELIDE.
- Syntax and Grammar Checker subsystem: provides Erasmus source code editor with syntax coloring, Code folding, Code completion, Brace matching, Coding tips, Indentation and Annotations. The editor is also used to locate and to view source code lines corresponding to the error messages.

- Tools subsystem: is the interface that is used to invoke compiler.
- Output subsystem: displays both console based outputs e.g. error messages produced by compiler and GUI-based outputs.
- ELIDE GUI Controller subsystem: manages GUI views of abstract syntax tree (AST) and XML files.

The most important design goal for ELIDE was to make the design flexible and extensible so that new ELIDE tools can be easily integrated in the future. We tried to achieve this goal by identifying and analyzing variable parts of the system and separating them through well defined interfaces. In the next chapter we explain the Implementation.

Chapter 6

Implementation

In this chapter we will explain the way we implemented ELIDE and we also mention lessons we learned by fulfilling this project.

6.1 Platform

Before starting to code Erasmus IDE, we noticed features such as project creation, file creation, editing, build/error detection, analyze, online help, search and version control are available in some of current IDEs. In order to use the most of available technology, we did a research on available platforms: Eclipse, NetBeans and other IDEs like DLTK and Aptana. Eclipse and NetBeans were our best choices because not only they would provide the above mentioned basic features, but also they are extendable. The rest of features can be added to them as plug-ins. The plug-in development environment helps us to develop and add those extra as plug-in later. We can also use Eclipse and/or NetBeans as integration tool to integrate those components and to replace those components with Eclipse and/or NetBeans native components later. If we use the Eclipse and/or NetBeans suggested API for making Erasmus plug-in, then its upgrade will be easy. Therefore we will keep up with new innovations.

We first developed ELIDE in Eclipse platform as is mentioned in appendix A. We created

a plug-in to have a powerful editor. Then we used Antlr plug-in in order to be able to have syntax checking. Unfortunately the Antlr plug-in is designed such that it can parse one sentence at a time and we could not use it inside the editor to parse the whole file in one shot. Since these two plug-ins acted separately we could not benefit from Antlr inside the editor, also for compiler integration, the process of creating a user interface was quite time consuming. At this stage we tried to find alternatives and NetBeans was chosen as our second choice.

NetBeans is open source software, which enables developers to write their own plug-in modules to add new toolset features. NetBeans is an IDE framework using a common set of APIs to connect code editors, compilers, debuggers, and other NetBeans-compatible tools. Although optimized for Java, NetBeans can work with APIs that hook into other languages. This Java-based framework is made up of two components: the NetBeans platform, a runtime library that provides the basic IDE elements such as an application's data presentation, configuration, and user interface; and the IDE itself, which provides controls, such as editing and version control, for the platform's functionality [24]. In particular, NetBeans uses the Abstract Window Toolkit, a widget set that Sun built to work with Java [24]. AWT is a set of base-level APIs that functions between the Java code and the graphical subsystem and lets developers create and manage how application elements (including GUI windows, toolbars, and buttons) will look and work [24]. AWT provides only basic graphics. For more advanced graphics, including colors and the way an application's appearance will change in reaction to user input, we need the Swing toolkit, a UI component library that functions on top of AWT [24].

As is mentioned in the Netbeans website, it is built like LEGO type as shown in the figure 20. The Modules ELIDE GUI Controller, ELIDE tools, ELIDE Syntax, Grammar Checker and ELIDE output were what we wanted to built on top of other LEGO pieces. In this way we can take advantage of years of experience of other build-in modules of NetBeans. We have implemented ELIDE GUI Controller, ELIDE tools, ELIDE Syntax,

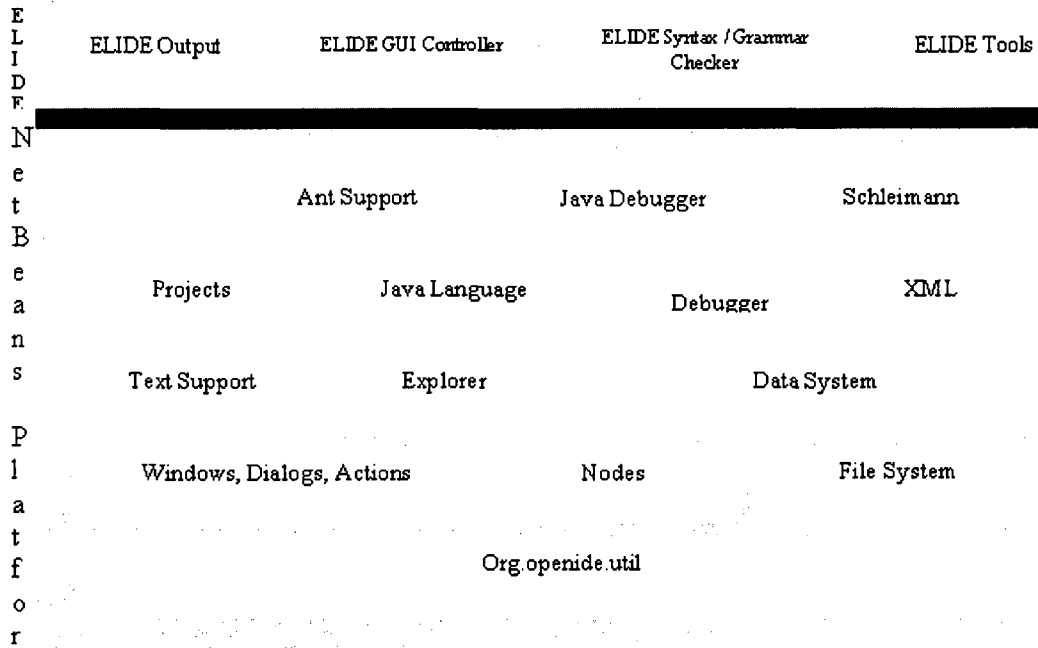


Figure 20: ELIDE Architecture using NetBeans

Grammar Checker and ELIDE output components of figure 19 and used File system of Netbeans mentioned earlier in chapter 5. We used Schliemann framework of NetBeans and on top of it we build the complete Editor with syntax coloring, Code folding, Code completion, Brace matching, Coding tips, Indentation and Annotations. Then we added our Compiler integration interface with extra tools of XML file related to mapping and graphical configuration presentation with built-in compile/debug/run ability. Using NetBeans has several benefits and drawbacks. It will be easy for user to switch between existing languages supported by NetBeans and new Erasmus language. There is no need to produce the Erasmus compiler in Java. The ELIDE will be able to use existing compiler which is working properly or future versions of the compiler. Moreover by using NetBeans, we avoided reinventing wheels and we bind the ELIDE to what we already have in NetBeans and we just concentrated to new functionalities. As mentioned earlier, the only drawback is that the ELIDE would be host application dependent.

6.2 User Interface

We have designed two User Interfaces, one is used for tool support Compile/Run/Graphical configuration and the other User Interface is used for Editor.

6.2.1 User Interface for Toolkit

We started the coding stage by creating a user interface with enough buttons for function keys and with menus for compile, build, test, and run any Erasmus file with current compiler. The way the Erasmus IDE works, is to type an Erasmus program in a file with “era” extension, then this file is compiled with the current version of Erasmus compiler. After syntax and semantic checking is completed, four different files will be generated. The first file is the executable file, the second is an equivalent version of program in C++, the third file is the log file which contains the AST file and Erasmus version of program and finally the fourth is a file with extension “evm” which is the machine code. If there is any compiling or execution error, then appropriate error messages will be shown on Erasmus IDE console view screen otherwise the result of execution will be displayed. The AST in log file will be used for ELIDE Diagram Support. Figure 21 demonstrates the User Interface used for editing and executing of any Erasmus program. On the left is the user interface of ELIDE, on the right is same thing but in command-line mode. The right screen (output in command-line) is not part of IDE. It is just used for comparison of IDE execution and Command-line execution of the same program. The user interface screen consists of 5 views. These view are project view, navigator view, source code, output and toolkit user interface. The next step was to design the User interface. The combination of MDI and tabbed window with IDE-style was the best choice. The next step in problem design was to decide how this new IDE application should help user to navigate between the windows. In Eclipse platform, we could use Eclipse IDE as base platform to add Erasmus language as a new language. But in Net Beans we had to create our own user interface. The user interface for toolkit looks like as in figure 22.

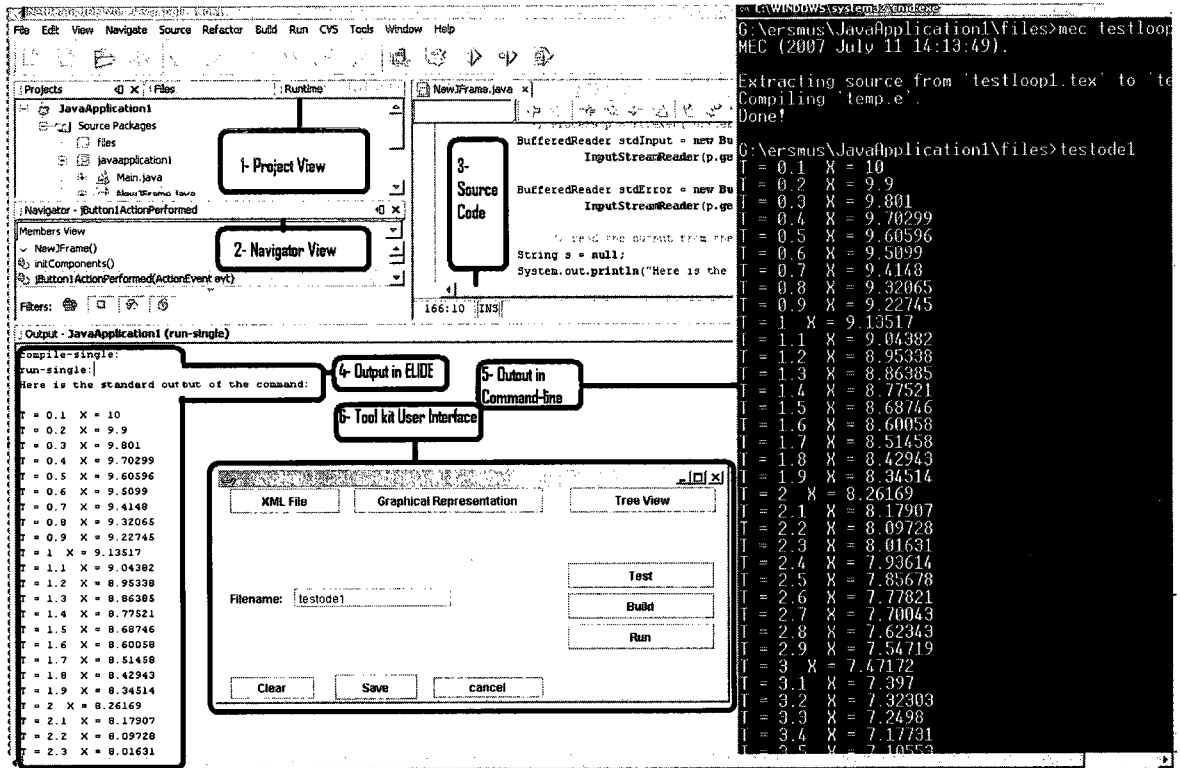


Figure 21: User interface in ELIDE

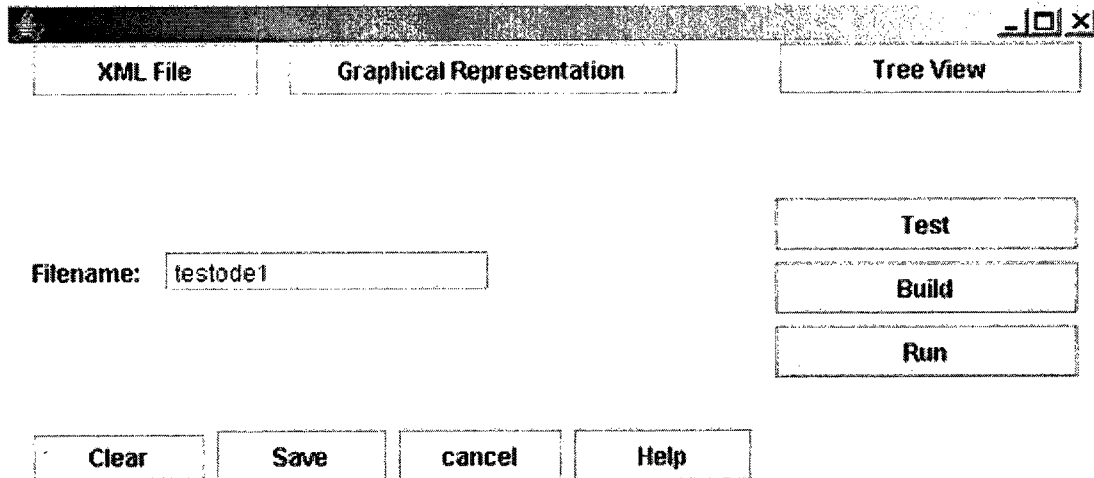


Figure 22: User Interface Design for tools

XML Button will show the mapping file which is used in compilation and is in XML format. The Graphical Representation Button will show the graphical view of Erasmus programs or projects in format of figure 14.

Tree View Button will show the tree view of the Erasmus program files and projects, so that we can open or save them.

Test Button will help in testing the Erasmus program files and defines breakpoints which will be used for debugging. In Erasmus test is an integral part of the code and like assertion and comments, makes them consistent with the codes [33]. Build button will only compile the Erasmus program files and projects.

The Run button will compile and execute the Erasmus program files and projects.

The Help Button will provide interactive help for Erasmus language syntax and other related topics.

The Cancel Button will exit from ELIDE.

The Save Button will save the exe files.

The Clear Button will erase the file name.

6.2.2 User Interface for Editor

Our first version of ELIDE produced in August 2008 had been implemented by using NetBeans platform and it supported only compiler integration, View and change the Erasmus programs and tool interface subsystem. Later in December 2008, the second version of ELIDE has been implemented by using Schliemann in NetBeans. In the second version of ELIDE, we added the following feature based on Erasmus syntax and grammar:

- Syntax coloring: to distinguish tokens and possibly non-terminals of the Erasmus language by a color in editor.
- Code folding: to wrap and unwrap pieces of code (e.g. block of statement in a loop) in editor.

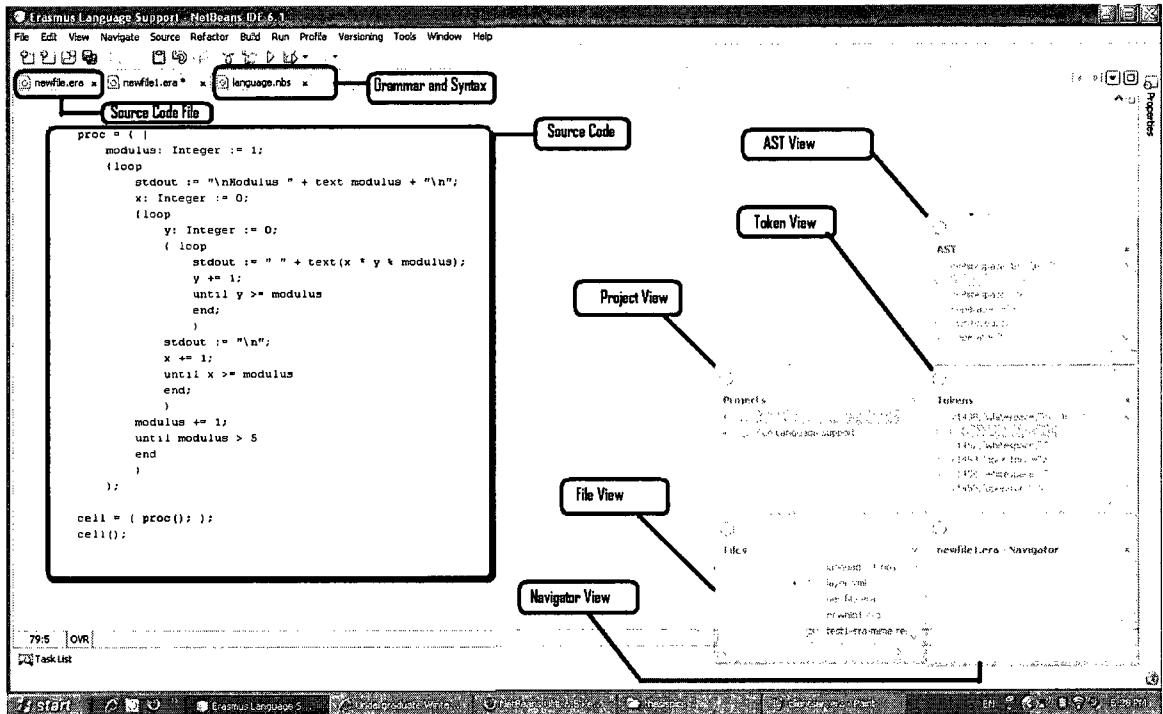


Figure 23: User Interface Design for Editor

- Navigation: to browse logical elements of the Erasmus language in Navigator window.
- Code completion: to help user to complete a piece of code based on the typed prefix.
- Brace matching: for a bracket located under the cursor, to highlight the pairwise bracket.
- Coding tips: to display coding tips on elements of the Erasmus language.
- Indentation: to properly indent documents based on the Erasmus language structure
- Annotations: to annotate specific lines of documents (e.g. error lines)

The implementation of these features is based on the output of lexical and syntactic analysis.

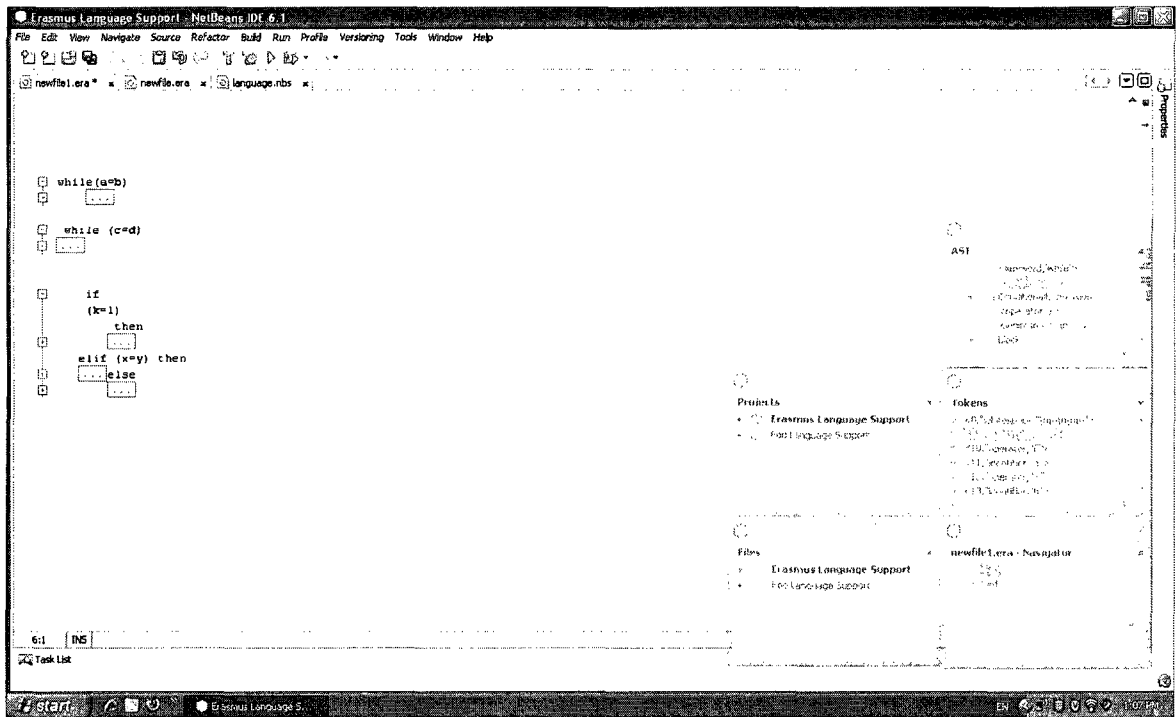


Figure 24: Erasmus program with Code folding

A sample screen shot of ELIDE editor User Interface is shown in figure 23. The user interface screen consists of 6 views. These view are project view, navigator view, source code, AST view, token view and file view. In order to show different Erasmus program files opened by user from File view, the Tabbed Document Interface is used. In this view the user has opened 3 files of “newfile.era”, “newfile1.era” and grammar file of Erasmus language. One of our challenges was to integrate grammar into ELIDE. Having done that, later we added one feature at a time. For example Code folding and Code unfolding was added to ELIDE and is shown in figure 24 and 25.

Later we added the Coding tips, Annotations and Jump to certain point of program which is shown in figure 26. Also Coding tips and Comment are shown in figure 27. Also screenshots of before and after indentation source code of Erasmus program are shown in figures 28 and 29.

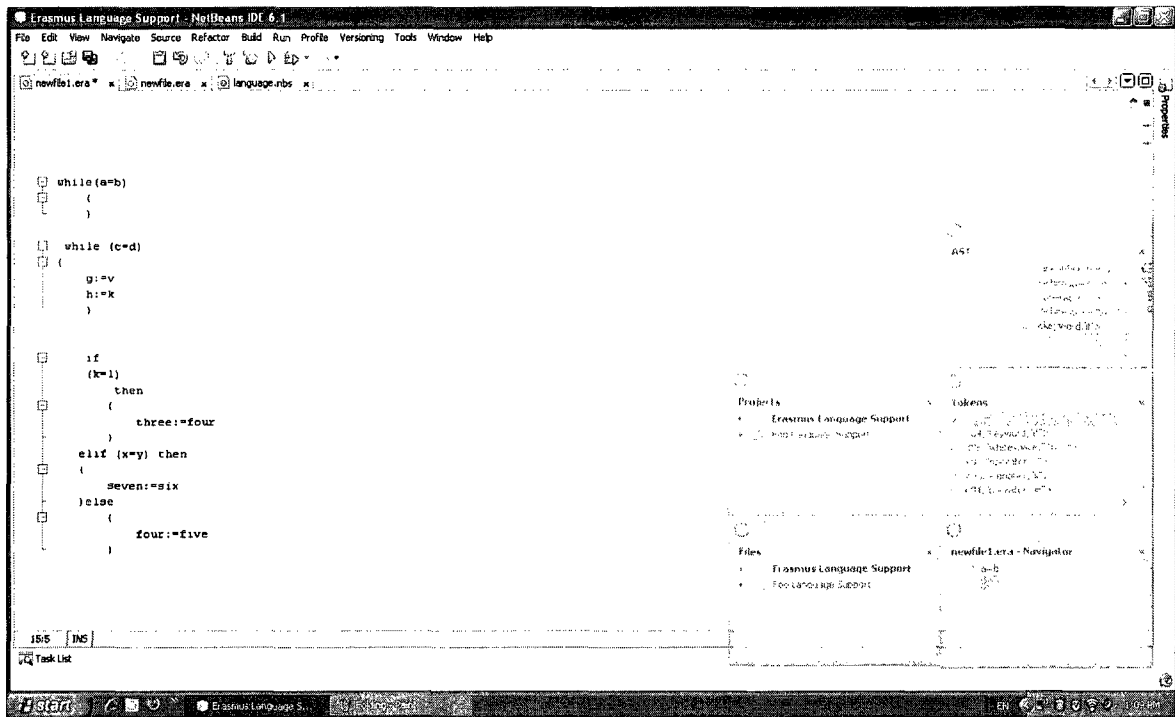


Figure 25: Erasmus program with Code unfolding

6.3 Software Development Model

As it was mentioned earlier, we followed waterfall lifecycle model of software development. We started with requirement analysis and produced System Requirements Document (SRS). Based on SRS, we designed the system architecture and refined it into detailed designs which were the foundation for the implementation phase; however, we frequently had to go back to the earlier phases to address the changes that resulted from feedback in the later phase.

We first developed ELIDE in Eclipse platform details of which is mentioned in appendix A. Then we developed a plug-in as an editor and then we had to add Antlr plug-in for parsing. Unfortunately it provided us with the opportunity to parse only one statement at a time which was not compatible with most of our FRs and some of NFRs. Also for compiler integration the process of building the user interface was quite time consuming. At this stage we tried to find alternatives and NetBeans was chosen as our second choice. We used Schliemann framework of NetBeans and on top of it we build the complete editor with

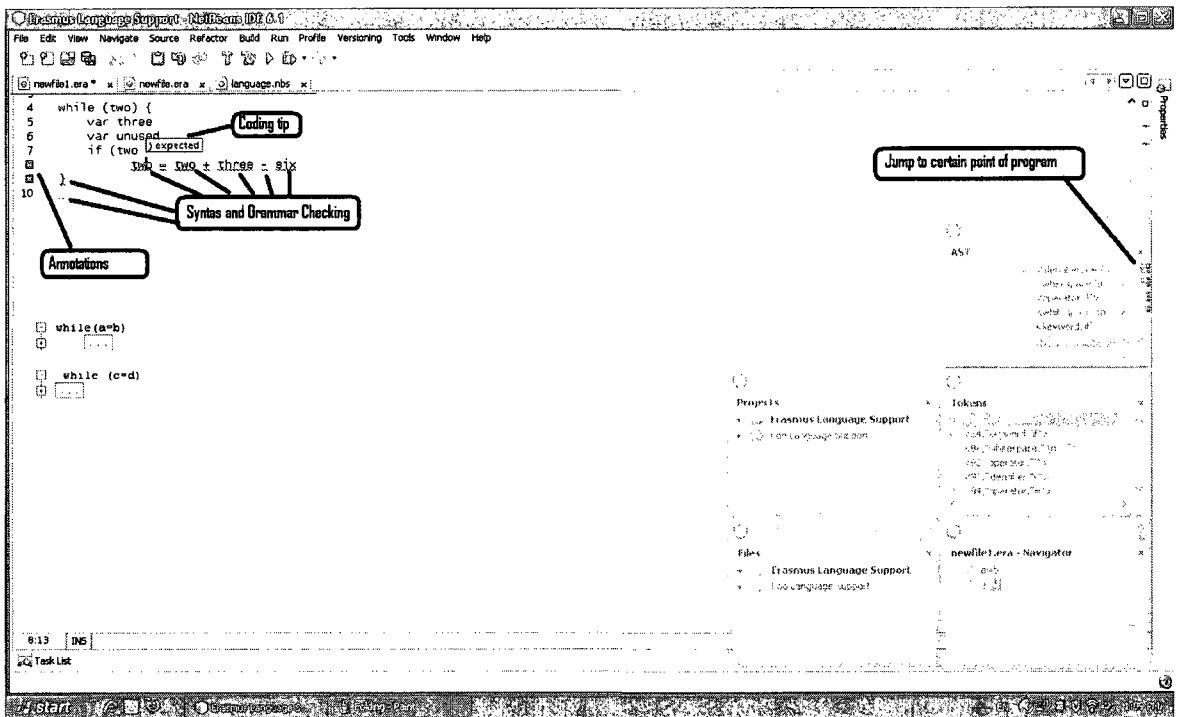


Figure 26: Erasmus program with Coding tip, Annotations and Jump to certain point of program

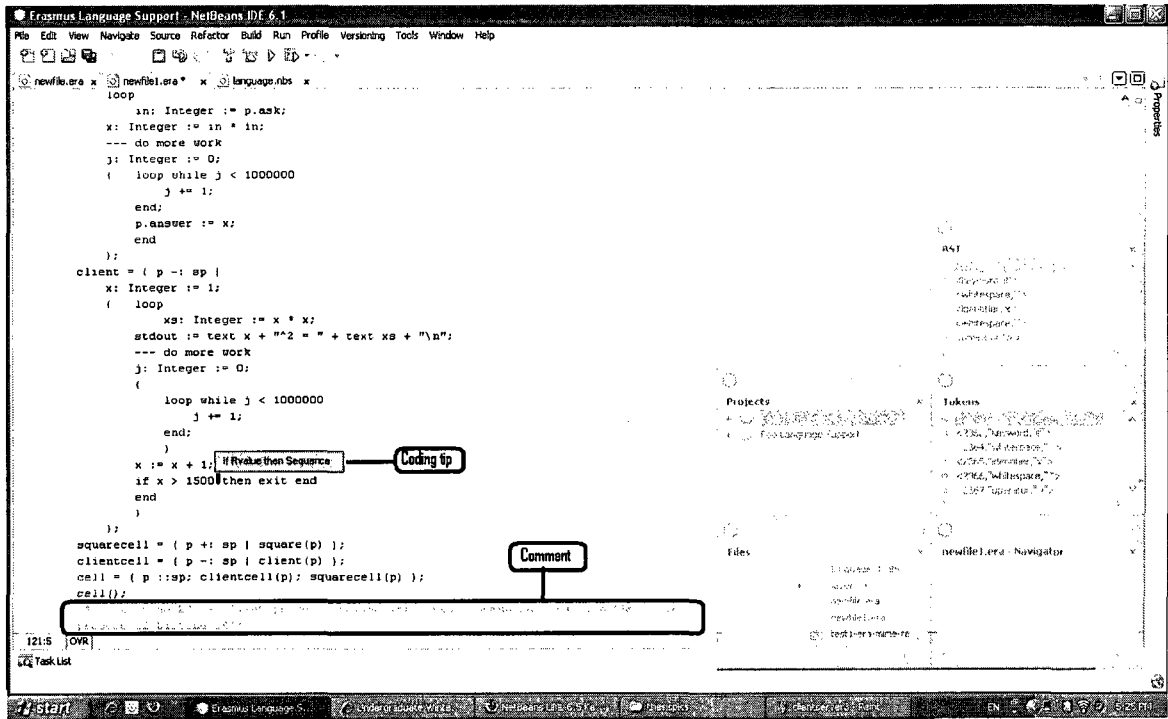


Figure 27: Erasmus program with Coding tip and Comments

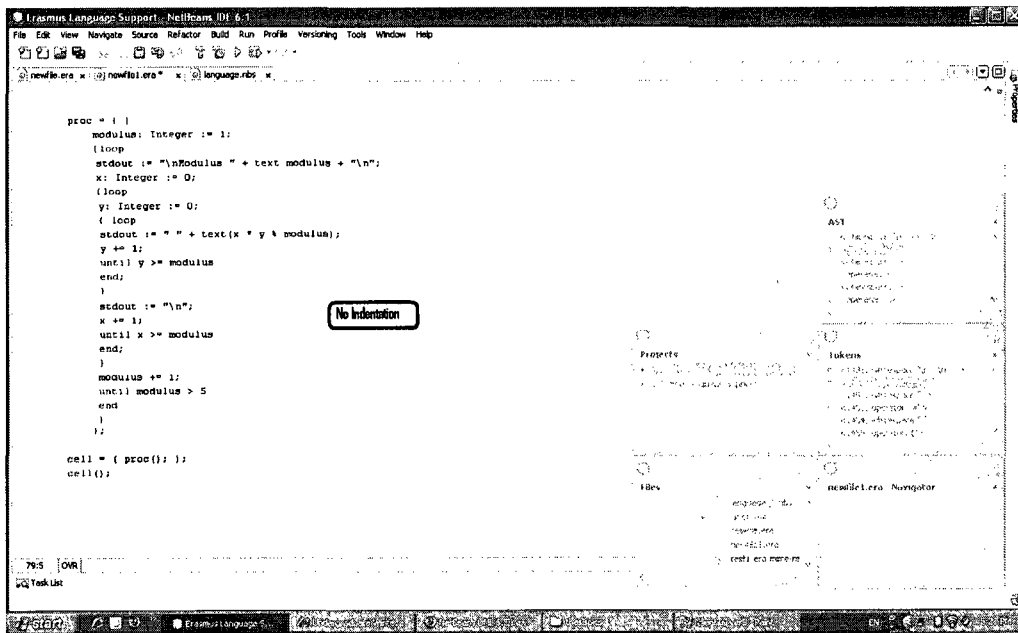


Figure 28: Erasmus program with no indentation

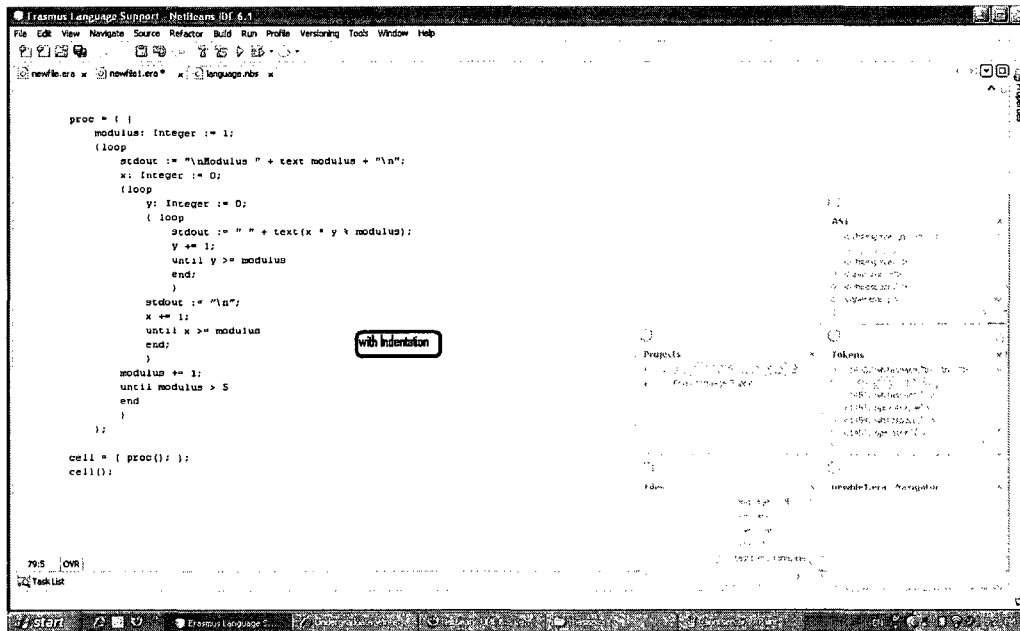


Figure 29: Erasmus program with indentation

Syntax coloring, Code folding, Code completion, Brace matching, Coding tips, Indentation and Annotations. Then we added our Compiler integration on toolkit user interface with extra tools of mapping file used by compiler and graphical configuration presentation with built-in compile/debug/run ability.

Chapter 7

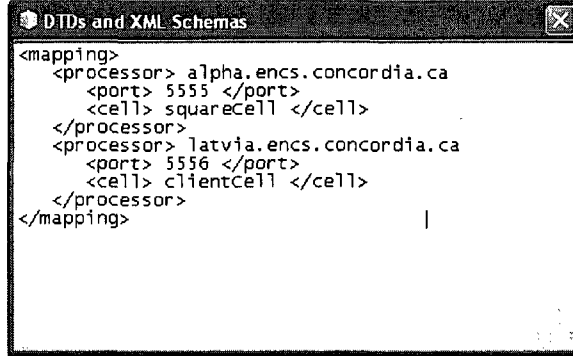
Case Study

With respect to the cycle of define, design, code, test, deploy and manage cycle, we wrote some Erasmus programs and run them in both command-line and then we run them in both platforms of ELIDE in Eclipse and NetBeans and these sample programs were compiled and executed successfully. In this chapter we illustrate one of these programs as a case study to show ELIDE's capability for a multiprocessing or distributed system programming.

7.1 A Sample Client Server Program

As it was mentioned earlier, Erasmus cells can be transmitted from one process to another and may be sent over a network. Compilation of this program requires a separate configuration file that specifies details about mapping of cells onto the participating processors. The mapping is saved in a file with XML format. The XML file contains properties about processes in a pair of `<mapping>` and `</mapping>` tags. The Mapping consists of :

- a pair of `<processor>` and `</processor>` tags called record. It contains the next two parts:
- a pair of `<port>` and `</port>` tags consists of port number of its communication agent called broker.



```
<mapping>
  <processor> alpha.encs.concordia.ca
    <port> 5555 </port>
    <cell> squareCell </cell>
  </processor>
  <processor> latvia.encs.concordia.ca
    <port> 5556 </port>
    <cell> clientCell </cell>
  </processor>
</mapping>
```

Figure 30: Mapping file in XML format

- a list of `<cell>` and `</cell>` tags.

The mentioned program and its mapping are shown in figure 31 and figure 30 respectively. The configuration file shown above maps `squareCell` onto the processor identified by “`alpha.encs.concordia`” and `clientCell` onto “`latvia.encs.concordia`”. If program figure 31 is compiled when no configuration file, it operates for stand alone systems. The `squareCell` and `clientCell` may be mapped onto different processes as shown in the configuration figure 30.

In this program two cells `clientCell` and `squareCell` are connected to a channel called `chan` through their local ports. The graphical configuration of this program is shown in the figure 32.

The ports use the protocol defined by `sqport`. The port that provides a service is a server and has a prefix “+” before name of protocol associated with the port. The port that needs a service is client port and is declared with “-”. In Erasmus when a port has no sign it creates a channel. Port `p` of `square` is server (it provides a square service) while `client` has a client port (it needs a square service). When the `mainCell` is instantiated, `clientCell` and `squareCell` are also instantiated. This causes a concurrent execution of processors in the two cells. The cell `clientCell` sends a number (i.e. 10) to `Square` cell via its local port called “port”. The `squareCell` sends its reply (i.e. string containing 100, the square of 10) to the

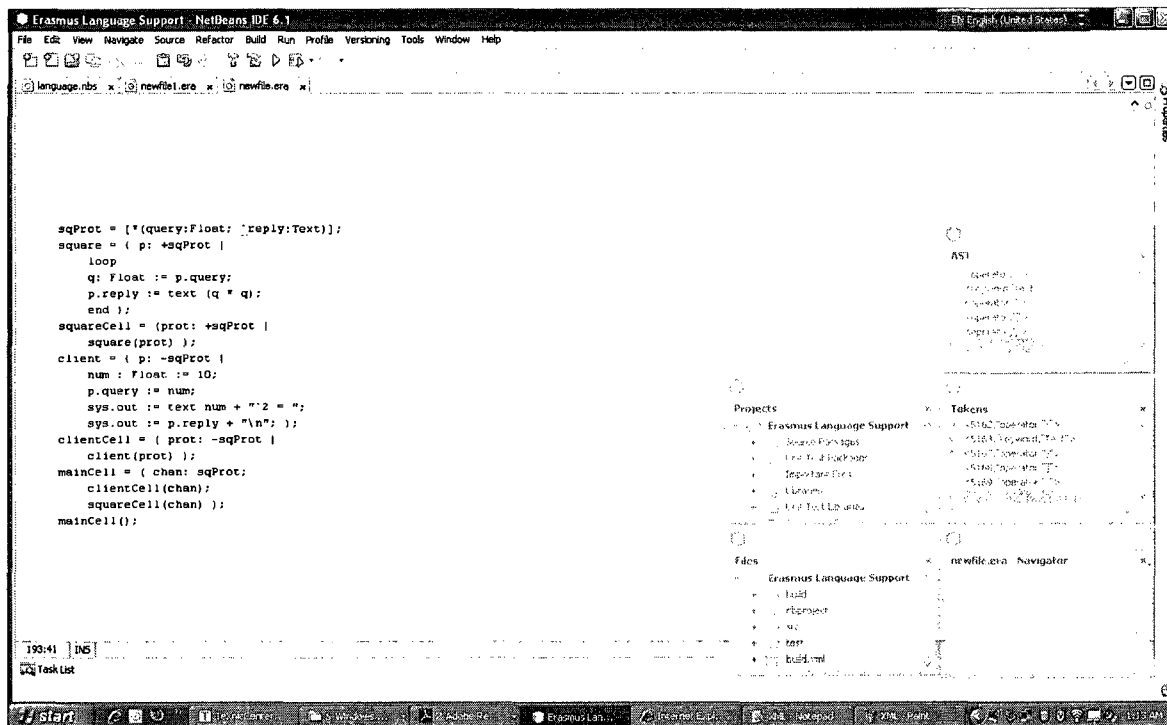


Figure 31: Client-Server program in Erasmus

client.

7.2 Compilation and Execution

By compiling the above program and mapping file, the compiler reads the XML content of mapping file extracts the data and organizes the contents into a table. The compiler generates a unique identification for each cell. This is eventually written to a file called `host.txt` in an order determined by cell id; each line of the file contains a record about a cell. If there is no entry for a cell in the configuration file, local host and portnumber "0" is written for the cell. This indicates that any available processor may execute the cell. Although the program solves a trivial function, it nevertheless shows how Erasmus facilitate, distribution of programs to different architectures. In practice this approach may help in less maintenance efforts when the software environment changes.

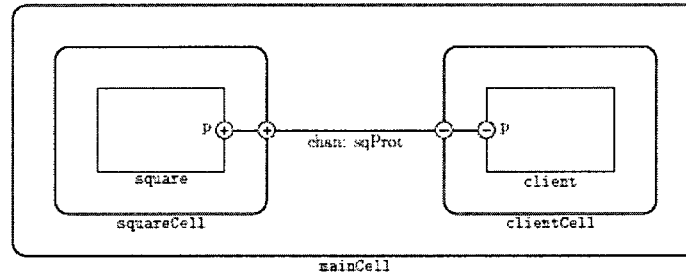


Figure 32: Graphical Configuration of the Client-Server program

7.3 Verification

We wanted to ensure that ELIDE fully satisfies all the expected requirements. We did dynamic verification, by performing all possible tests and experimentation on ELIDE. These tests verified the internal attributes. The types of tests performed are as follows:

- Unit test (a single function or class or group of classes)
- Module test (a single module)
- Integration test (more than one module)
- System test (the entire system)

First series of tests were performed in Eclipse, and then we imported all program test files of Erasmus together with the grammar file into Eclipse. The grammar file contains all grammar rules of Erasmus language. Then we used ELIDE to parse the grammar of those Erasmus program. By typing any statements of Erasmus language, it will be checked against the current grammar in ELIDE which does the syntax checking. If there is no error, then we can run the compiler to run that program otherwise the ELIDE displays error message. All test file were run successfully and the same results as in command-line environment outputs. The screenshots of these samples are added to Appendix A.

The second series of tests were performed in Netbeans by using the Erasmus compiler. The results of unit testing, module testing, integration testing and system testing confirmed

that ELIDE has satisfied all functional requirements. Functional requirements FR.1 up to FR.17 were related to editing facilities which were satisfied by NetBeans platform. Functional requirements FR.18 up to FR.20 were related to Erasmus compiler and were satisfied by several system tests. Functional requirements FR.21 up to FR.30 were related to toolkit and were satisfied by Module test, integration tests and system tests by both developers and language designers. Also we performed an extra set of tests by importing all of existing program files of Erasmus into ELIDE. After we compiled those programs, we executed them and we get the same result as command-line execution. The procedure to perform those tests was to save Erasmus program in text files with “era” extension. Then the file name is entered in the User Interface of ELIDE in the box called file name, then we press the Build button. This will cause the Erasmus compiler to compile it. If there is no compiling error, the output of compiler is shown otherwise the error message will appear on the ELIDE. If there is no compiling error, then we can press the Run button to see the output of the program. If there is execution error, then the appropriate error message is shown. All test file were run successfully and the same results as in command-line version were output. Screenshots of Some of these tests are added to the Appendix B. The Erasmus development team was satisfied with test results. As these tests measured purely in terms of the product itself, separate from its behavior or in other words they measured internal attributes of product.

The programs tested in this thesis may also vary with regard to their maturity level. For example, some programs may have gone through many iterations of changes whereas others may be relatively immature.

7.4 Validation

Validation of ELIDE provides a high degree of assurance that it accomplishes its intended non-functional requirements and can only be measured with respect to how the product relates to its environment. This process needs acceptance of end users (existing Erasmus

programmers and other product stakeholders (i.e. designers of Erasmus language). Also we can show the following non-functional requirements are satisfied:

- NFR.1 is satisfied since ELIDE can successfully perform Syntax checking, Syntax coloring, Code folding, Code completion, Brace matching, Coding tips, Indentation, Annotations, parsing, compiling and executing of Erasmus programs. Also ELIDE supports the Erasmus language and its compiler.
- NFR.2 is satisfied since the NetBeans and ELIDE have capacity for screen fields, tables, transaction volumes, report formats, etc.
- NFR.3 is satisfied since NetBeans and ELIDE have capability for different specified environments without applying actions or means other than those provided for this purpose.
- NFR.4 is satisfied since ELIDE is understood, learned, used and is attractive to the user, when used under specified conditions. More details will be discussed in next chapter.
- NFR.5 is satisfied since ELIDE enables the user to understand whether the software is suitable, and how it can be used for particular tasks and conditions of use. More details will be discussed in next chapter.
- NFR.6 is satisfied since ELIDE enables the user to learn its application. More details will be discussed in next chapter.
- NFR.7 is satisfied since ELIDE enables the user to operate and control it.
- NFR.8 is satisfied since ELIDE maintains a specified level of performance when used under specified conditions.
- NFR.9 is satisfied since ELIDE satisfies and represents the needs by a specific set of values for the quality characteristics. The management and editing features of ELIDE will be quick and responsive.

- NFR.10 is satisfied since ELIDE is transferred from one environment to another. Moreover ELIDE is a Windows application programmed in Java and can be run on any computer that runs Windows and NetBeans environment.
- NFR.11 is not completely satisfied since at the moment ELIDE can only be used in the lab and later we will be able to download it over the internet.
- NFR.12 not completely satisfied since, although ELIDE is a free product, at the time of writing this thesis, it can not be released as a free product for non-commercial use.
- NFR.13 is satisfied since ELIDE protects information and data so that unauthorized persons or systems can not read or modify them and authorized persons or systems are not denied access to them.
- NFR.14 is satisfied since ELIDE provides appropriate performance, relative to the amount of resources used, under stated conditions.
- NFR.15 is satisfied since ELIDE can be modified. It is designed such that modifications like corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications as well as new features and tools can be added.

To measure its external attributes, we also performed usability study. As was mentioned earlier, the detail of this survey is discussed in more detail in the next chapter.

Chapter 8

Evaluation

With respect to the cycle of define, design, code, test, deploy and manage cycle, we deployed ELIDE in our lab. Usability of ELIDE was another very critical aspect of this software system. Since we knew even when all expected functions are realized correctly, the product's success still is not guaranteed. Stan and Stahl point out that all systems, independent of what hard or software and how clever they are built, involve humans [53,54]. This means that there is a need for user interface that eases the use of the system in question. Non-usable user interfaces might cause users to turn away from product [53,54]. As a quality attribute, usability gives the possibility to differentiate between products. Stahl and Sommerville define similar attributes for the usability of a software system, which at the same time give a hint for potential measurements [54,55,62]:

- Learning time or learnability: The Learning time or learnability refers to the time that a user needs to reach a level of ability to use the system productively.
- Task Performance Time: The Task Performance Time defines the time the user needs to perform a specific task with the software system.
- Error Rates: The Error Rates show how often a user generates errors. A high error rate indicates a difficult to use software.

- **Robustness:** The Robustness indicates how tolerant a system behaves, in case the user makes an error.
- **Error Recovery Time or Recoverability:** The Error Recovery Time or Recoverability defines how good a system is able to deal with errors made by users.
- **Adaptability** The Adaptability is a measure of whether it is possible to adapt the system to different ways of working.

In most literature we studied (Stan, Stahl, Sommerville) the authors state that only the true users can determine whether a system is usable or not. ELIDE has been demonstrated to 10 graduate students. ELIDE was used to write small Erasmus program and run them in ELIDE. We conducted a usability study, some of the findings of which are discussed in the next section. Sample Questioner is shown in Appendix D.

8.1 Preliminary Survey Results

As was mentioned earlier, we conducted a preliminary usability survey of ELIDE involving ten Graduate students. Our approach was to ask the participants about their familiarity with Eclipse, NetBeans and Erasmus. Then we asked for feedback about the tool and its applicability to programming. Nine out of ten people that we surveyed were familiar with at least one of them and thought that it is useful. Out of these nine, two were experts in Eclipse and NetBeans, three moderately familiar, and two novices. The other two didn't know anything about them. However, none of the participants were aware of any such tool for Erasmus language. The results of usability survey are shown in table 5 and table 6.

Some of the findings which extracted from the survey are as follows. All people felt that the tool is more user friendly. Nine out of ten people thought that the output of the tool is useful. Finally, all of the participants thought that developers would find this tool useful.

Title	Population of users
familiarity with Eclipse and/or NetBeans and/or Erasmus	90%
non-familiarity with Eclipse and NetBeans and Erasmus	10%
familiarity with ELIDE	0%

Table 5: Result of Usability Survey -Population of users

Title	Expertly of users
Expert in Eclipse and/or NetBeans and with knowledge about Erasmus	20%
Moderately familiar with Eclipse and/or NetBeans and with knowledge about Erasmus	30%
Novices to Eclipse and/or NetBeans and with knowledge about Erasmus	20%
No knowledge about Erasmus	20%
No knowledge about any of above	10%

Table 6: Result of Usability Study - Level of knowledge of users

In summary, the results were quite encouraging with respect to the group surveyed and current capabilities of ELIDE and Erasmus. We consider them as initial feedback which will be the base for future surveys.

8.2 Preliminary Survey Recommendations

We also got some useful suggestions regarding the enhancement that can be made to the tool. Some of these are addition of browsing and searching in help and support for visual (drag and drop) and automatic replacement of the output of the tool in a file. We plan to work on some of these in future.

Chapter 9

Related Works

Several research papers have proposed approaches to implement IDEs in the literature. These IDEs are created for different purposes. Some of them specifically relates to design of IDE for languages, though others are more tool based. These new experiences mostly are introduced by private institution and their design and methodology is hidden from researcher. But there are a few research papers about adding application software as a plug-ins in Eclipse for different purpose compared to ours. Of course there are some differences between the types of plug-in they used and the domain in which they are used. Nevertheless they provided a good starting point.

The first similar work is Mylar. It is a tool that focuses on the elements visible in IDE views on the context of a programmer's task [1][2]. Mylar is designed as a plug-in which will function as a tool for Eclipse platform. The similarity was the way they created a plug-in in Eclipse and the way they evaluated their final product. The difference is the fact that Mylar is not a language. It is rather a tool for Eclipse.

The other related work is Synthy which is an IDE for end-to-end composition of Web services [7]. This research came to our attention because it was related to IDE Design and implementation, though the concept was completely different. The similarity was the way they created plug-in in Eclipse. The difference is the fact that Synthy is not a language and that it is rather a tool for web services.

The next related work is the design of IDE for PSF (Process Specification Formalism) [3]. The similarity is the way they created plug-in in Eclipse and that it is a language. The difference is the fact that it is rather a verification tool for process Specification Formalism and the language is more mathematical.

The next interesting work is called JML4 which is a tool support proposal for an Integrated Verification Environment (IVE) for Java Modeling Language [45,46]. This IDE is built upon Eclipse's support for Java. It is enhanced with run time assertion checking (RAC), Extended Static Checking (ESC) and full static program verification (FSPV).

The next work was about extending Eclipse CDT to support a gcc-based top language (Eightbol). With the experiences gained, the same extensions have used to implement Photran version 3.6 a full featured IDE for FORTRAN language in Eclipse C/C++ development tool [47].

Finally the Experience Report of Building an Eclipse-based IDE for Haskell was another related work on developing an IDE for a Haskell language on Eclipse platform. In this paper they have provided good comments about the lessons learned from the project [56].

Chapter 10

Conclusions and Future Works

In this thesis, we presented the design and development of an Integrated Development Environment (IDE) for Erasmus language called ELIDE. The design of the IDE was based on the functional and non-functional requirements of ELIDE based on views of developers of Erasmus language. What we achieved was a complete IDE as we planned and what is left for future is refactoring of codes, adding bookmarks, semantic checking of comments and creating a visual modeling language for Erasmus. The visual modeling language can also be used for other process-oriented languages. We can achieve forward and reverse engineering by simply drawing the graphical representation of a program to get source code or by coding and getting the Graphical representation of program. Also the breakpoint for debugging needs more elaboration. ELIDE is designed in such a way that it is able to be extended and any feature gaps can be filled by later plug-ins in both Eclipse and Netbeans versions. The emphasis was to elaborate around the concept of a perspective that makes it easier to get the job done, because the appropriate tools were close at hand. It will be more effective tool if we use an easy navigation within source code by adding bookmarks. Also we can add strong support for semantic checking of in-line documentation with the associated source to help the users of Erasmus language in debugging and maintenance and documentation. We had some difficulties in using plug-ins with Eclipse because each plug-in is written by separate group for different versions of Eclipse and NetBeans. Therefore there were some inconsistencies at

the beginning. As our knowledge about Eclipse, NetBeans and plug-ins became deeper and deeper, it became more interesting. At first we developed a version for the only used plug-in (Antlr) in our design. But this plug-in was designed for a line of program at a time and it was not very useful when coding large programs. Therefore we searched for other platforms for providing us more capabilities. We restart the whole process in NetBeans and end up with ELIDE. ELIDE is a strong environment for a complete programming support for Erasmus language with built-in compile/debug/run ability. The most important features included in ELIDE are Syntax coloring, Code folding, Code completion, Brace matching, Coding tips, Indentation and Annotations. We managed to converted a batch oriented command-line based interface to an interactive one and increased the productivity and efficiency of Erasmus programmers. Our aim is to introduce our final product ELIDE and to save it at Concordia's University Server so that it can easily be downloaded and installed on any computers. Our product would be open source. This would encourage more application to be implemented in ELIDE.

Glossary

Adobe Photoshop: Adobe Photoshop or simply Photoshop, is a graphics editor developed and published by Adobe Systems.

Adobe Acrobat: Adobe Acrobat was the first software to support Adobe Systems' Portable Document Format. It is a family of software, some commercial and some free of charge.

Aptana Studio: is an open source integrated development environment (IDE) for building Ajax web applications.

AST: Abstract Syntax Tree is a special kind of tree that can have an arbitrary number of subtrees(children) which are ASTs themselves. When we walk on the tree one can manipulate the order in which nodes are visited with all the expressiveness of the manipulation language. The AST structure is defined in operator precedence Development stage.

Beta form: In computer programming, development stage terminology expresses how the Software engineering of a piece of software has progressed and how much further development it may require.

CDDL : Common Development and Distribution License (CDDL) is a free software license, produced by Sun Microsystems, based on the Mozilla Public License (MPL), version 1.1.

DLTK: Dynamic Library Tool Kit is an IDE that has some features like the Project Wizard, Code Editor, Code Navigation, Code Assist and Launching and Debugging. This IDE has been used for TCL and Ruby languages.

Fork: In software engineering, a project fork happens when a developer takes a copy of source code from one software package and starts to independently develop a new package.

FTP: File transfer protocol, is a file transfer protocol for exchanging and manipulating files over any TCP-based computer network.

GIMP: The GNU Image Manipulation Program or just GIMP is a free software Raster graphics editor.

GPL: The GNU General Public License (GNU GPL or simply GPL) refers to free software license. A software license which grants recipients rights to modify and redistribute the software which would otherwise be prohibited by copyright law. A free software license grants, to the recipients, freedoms in the form of permissions to modify or distribute copyrighted work.

HTTP: Hypertext Transfer Protocol is a communications protocol for the transfer of information on the Internet. It is used for retrieving inter-linked text documents

IDE-style interface: IDE-style interface are those whose child windows reside under

a single parent window(usually with the exception of modal windows).

Macromedia Studio: Macromedia Studio is a suite of different programs designed for web content creation and was designed and distributed by Macromedia.

Microsoft Excel: Microsoft Excel is a spreadsheet program written and distributed by Microsoft for computers using the Microsoft Windows operating system and for Apple Macintosh computers.

Microsoft Windows: Microsoft Windows is a family of operating systems by Microsoft. They can run on several types of platforms such as server , embedded devices and, most typically, on personal computers.

Microsoft Word: Microsoft Word, or Microsoft Office Word, is Microsoft's flagship word processor computer software.

MDI: Multiple Document Interface (MDI) are those whose windows reside under a single parent window (usually with the exception of modal windows), as opposed to all windows being separate from each other (single document interface).

Modal window: In User Interface design, a modal window is a child window, which has to be closed before the user can return to the operating the parent application.

Multiple instances: Multiple instances of the program can be opened simultaneously for editing multiple files. It applies both for SDI and MDI programs. Also applies for program that has an user interface that looks like multiple instances of the same program (such as some versions of Microsoft Word).

Notepad++: Notepad++ is a free source code editor which supports several programming languages running under the Microsoft Windows environment.

Overlappable windows: each opened document gets its own fully movable window inside the editor environment.

PSPad: PSPad editor is a freeware text editor and source editor for the Windows platform. First released in 2001, this software is produced by the single developer Jan Fiala.

Single document window splitting: window can be split to simultaneously view different areas of a file.

SSH: Secure Shell, is a network protocol that allows data to be exchanged using a secure channel between two networked devices.

Tabbed document interface: In Graphical User Interface, Tabbed document interface is the one that allows multiple panes of information to be contained within a single master window, using tabs to navigate between them.

TextMate: TextMate is a general-purpose text editor for Mac OS X, which tries to combine the power and flexibility of UNIX text editors such as Vim and Emacs with the simplicity and elegance of a Macintosh program.

TCL: Tool Command Language is a scripting language created by John Ousterhout. It is most commonly used for rapid prototyping,

UltraEdit: UltraEdit is a text editor for Microsoft Windows created by IDM Computer Solutions. The editor contains tools for programmers, including macros, syntax highlighting, code folding, file type conversions, project management, regular expressions.

Virtual desktop: Virtual desktop is a term used, usually within the WIMP paradigm, to describe any of several possible ways in which a computer's metaphorical desktop environment is modified, through the use of software.

Window splitting: splitting application window to show multiple documents (non-overlapping windows).

WebDAV: Web-based Distributed Authoring and Versioning, (WebDAV), is a set of extensions to the Hypertext Transfer Protocol (HTTP) which allows users to collaboratively edit and manage files on remote World Wide Web servers.

Bibliography

- [1] M. Kersten, *Focusing Knowledge work with task content (Mylar IDE plug-in)*, PhD Thesis, University of British Columbia, (January 1998).
- [2] M. Kersten and G.C. Murphy, *Mylar: a degree-of-interest model for IDEs* Department of Computer Science, University of British Columbia, Vancouver, BC, Canada (1998).
- [3] B. Diertens, *Software (Re-) Engineering with PSF III: an IDE for PSE*, Programming Research Group, Faculty of Science, University of Amsterdam.
- [4] P. Grogono and B. Shearing, *Concurrent Software Engineering: Preparing for Paradigm Shift*, Canadian Conference on Computer Science and Software Engineering (C3S2E'08), Montreal, Pages 99-108 (May 2008).
- [5] P. Grogono and B. Shearing, *Modular ConCurrency: a New Approach to Manageable Software*, 3rd International Conference on Software and Data Technologies (ICSOFT 2008), Portugal, (July 2008).
- [6] N. Lameed And P. Grogono, *Separating program semantic from development*, 3rd International Conference on Software and Data Technologies (ICSOFT 2008), Portugal, (July 2008).
- [7] G. Chaffe et al., *An Integrated Development Environment for Web Service Composition*, IEEE International Conference on Web Services (CWS 2007) (2007).
- [8] P. Grogono, *Sample Erasmus programs* (2006)(2007)(2008).

- [9] D. Gallardo, *Migrating to Eclipse: A developer's Guide to Evaluating Eclipse vs. NetBeans*, International Business Machine Corporation, (September 1992).
- [10] D. Gallardo et al., *A Guide for Java Developers: Eclipse in Action*, Manning, (2003).
- [11] K. Williams, *Seminar on the Versatility and power of the IDE - Maximising Developers productivity*, Webcast, (March 22, 2006 at 11AM EST).
- [12] E. Gamma et al., *Eclipse modeling framework*, Addison-Wesley, (2004).
- [13] G. Goth, *Beware the March of This IDE: Eclipse is Overshadowing other Tool Technologies*, IEEE SOFTWARE, IEEE Computer Society, (July/August 2005).
- [14] G. Coulouris, Jean. Dollimore and T. Kindberg, *Distributed Systems Concepts and design*, Addison Wesley, 4th Edition (September 2005)
- [15] Y. Zhang, G. Huang, N. Zhang and H. Mei, *Editable Replay of IDE-Based Repetitive Tasks*, 32nd Annual IEEE International Computer Software and Applications Conference, pp 473-480, (2008).
- [16] P.B. Goth, *The Joyce Language Report*, Syracuse University, Syracuse, New York.
- [17] *Eclipse C/C++ Development Tooling-CDT*, [Http://www.eclipse.org/cdt/](http://www.eclipse.org/cdt/), (Last seen August 2008).
- [18] *AJDT:AspectJ Development tools*, [Http://www.eclipse.org/ajdt/](http://www.eclipse.org/ajdt/), (Last seen September 2008).
- [19] *Eclipse Java Development tools (JDT) Subproject*, [Http://www.eclipse.org/jdt/](http://www.eclipse.org/jdt/), (Last seen August 2008).
- [20] P. Grogono, *Issues in the Design of an Object Oriented Programming Language*, (January 1991).
- [21] J. Martin, *Rapid Application Development Macmillan Coll Div, ISBN 0-02-376775-8*, International Business Machine Corporation, (September 1992)

- [22] J. Martin and W. Scacchi, *Collaboration, Leadership, Xontrol, and Conflict Negotiation and NetBeans.Org Open Source Software Development Community*, Institute for Software Research, University of California Irvine, Irvine, CA USA 92697-3425(September 1992).
- [23] [Http://www.eclipse.org/](http://www.eclipse.org/), (Last seen December 2008).
- [24] [Http://www.netbeans.org/](http://www.netbeans.org/), (Last seen January 2009).
- [25] [Http://www.antlereclipse.sourceforge.net/](http://www.antlereclipse.sourceforge.net/), (Last seen January 2009).
- [26] [Http://www.antler.org/](http://www.antler.org/), (Last seen January 2009).
- [27] [Http://www.erlang.org/](http://www.erlang.org/), (Last seen January 2009).
- [28] [Http://www.canonware.com/onyx/](http://www.canonware.com/onyx/), (Last seen January 2009).
- [29] [Http://www.wikipedia.org/](http://www.wikipedia.org/), (Last seen January 2009).
- [30] S. Holzer *Eclipse*, O'Reilly, (2004).
- [31] J. Arnowitz, M. Arent, N. Berger *Effective prototyping fo software maker*, Elsevier, (2007).
- [32] B. Schneiderman, *Leonardo's Laptop: Human Needs and the New Computing Technologies*, MIT Press, Cambridge, MA, (October 2002).
- [33] K. Olukotun and L. Hammond, *The future of Microprocessors*, ACM Queue, 3(7) pp 26-34, (2005).
- [34] P. Grogono and B. Shearing, *A Modular Language for Concurrent Programming*, Technical Report, (September 2006).
- [35] H. Sutter, *The free lunch is over - A fundamental turn toward Concurrency in software*, Dr. Dobb's Journal 30(3), <http://www.gotw.ca/publications/concurrency-ddj.htm>, (2005).

- [36] E.A. Lee, *The problem with threads*, IEEE Computer, 39(5) pp33-42, (2006).
- [37] H. Sutter, *The trouble with locks*, Dr. Dobb's Journal, (2005).
- [38] T. Harris and K. Fraser, *Language Support for Lightweight Transactions*, ACM SIG-PLAN Notices, 38(11) pp388-402, (2003).
- [39] C.A.R. Hoare, *Communication Sequential Processes*, Communication of the ACM 21(8) pp 666-667, (1978).
- [40] P. Brich Hansen, *Joyce - A Programming Language for Distributed Systems*, Software Practice and Experience 17(1) pp 29-50, (1987).
- [41] J. Armstrong, R. Viriding, C. Wikstrom and M. Williams, *Concurrent Programming in ERLANG*, Printice Hall, Second Edition, (2003).
- [42] F. Barnes and P. Welch, *Prioritized Dynamic Communicating and Mobile Processes*, IEEE Proceedings Software, 150(2) pp 121-136 (2003).
- [43] F. Barnes and P. Welch, *Communicating Mobile Processes*, Communicating Process Architectures, pp 201-218 IOS Press (2004).
- [44] [Http://www.IBM.com/](http://www.IBM.com/), (Last seen January 2009).
- [45] P. Grogono, N. Lameed and B. Shearing, *Modularity + Concurrency = Manageability*, University of Concordia, Montreal, (2008).
- [46] P. Chalin, P.R. James and G. Karabotsos, *JML4: Towards an Industrial Grade IVE for Java and Next Generation Research Platform for JML*, Dependable Software Research Group, University of Concordia, Montreal, (2008).
- [47] P. Chalin, P.R. James and G. Karabotsos, *An Integrated Verification Environment for JML: Architecture and Early Results*, Dependable Software Research Group, University of Concordia, Montreal, (2008).

- [48] J. Overbey and C. Rasmussen, *Instant IDEs: Supporting New Languages in the CDT*, Eclipse'05 October 16-17 2005, San Diego, CA, (2005).
- [49] A.B. Perez, Y. Cheon and A.Q. Gates, *Canica: An IDE For Java Modeling Language*, University of Texas at El Paso, Texas, (2006).
- [50] J. Tidwell, *Designing Interfaces*, O'Reilly, (2006).
- [51] D. Benyon et al., *A Guide to usability : human factors in computing*, Addison-Wesley, (1993).
- [52] J. Tidwell, *The Java developer's guide to Eclipse*, Addison-Wesley, (2003).
- [53] T. Stahl and et al., *Model-driven software development : technology, engineering, management*, John Wiley, (2006).
- [54] C. Stan, *Testing the ergonomics*, Machine Design, pp108-109, (2004).
- [55] C. Stahl, *Testing for Usability can head off Disaster*, Computerworld, vol. 21 pp. 83-89, (1987).
- [56] I. Sommerville, *Software Engineering*, 7th Edition, Addison-Wesley (2004).
- [57] L. Frenzel, *Experience Report: Building an Eclipse-based IDE for Haskell*, Proceedings of the 2007 ACM SIGPLAN International Conference on Functional Programming (ICFP 2007) pp220-222 (2007).
- [58] B.A. Myers, *A Taxonomy of Window Manager User Interfaces*, IEEE Computer Graphic and Applications, pp65-84 (September 1988).
- [59] B.A. Myers, S.E.Hudson and R. Pausch, *Past, Present and Future of User Interface Software Tools*, ACM Transactions on Computer-Human Interaction, Vol.7, No. 1, pp3-28 (March 2000).
- [60] S. Shavor et al., *The Java Developer's Guide to Eclipse*, International Business Machines Corporation, (2003).

- [61] S. Shneiderman, *Designing the User Interface*, Addison-Wesley Longman, 3rd Edition, (1998). Addison-Wesley, (2003).
- [62] J. JanCure and D. Prusa, *Generic Framework for Integration of Programming Languages into NetBeans IDE*, Sun Microsystems, (2008).
- [63] C. Benson, A. Elmanand andS. Nickell and C. Robertson, *GNOME Human Interface Guildlines*, (2002).
- [64] B. E. John, *Evaluating Usability Evaluation Techniques*, ACM Computing Surveys, Volume 28, Issue 4es, (1996).
- [65] [Http://www.msdn.com/](http://www.msdn.com/), (Last seen March 2009).
- [66] P. Charles and Et al, *SAFARI: a meta-tooling framework for generating language-specific IDE's*,21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications table of contents, Volume 28, Issue 4es, pp722 - 723,(2006).
- [67] ISO/IEC 9126-1, *Software engineering Product quality Part 1: Quality model*, The International Organization for Standardization (ISO), First edition 2001-06-15, (2001).
- [68] ISO/IEC TR 9126-2, *Software engineering Product quality Part 2: External metrics*, The International Organization for Standardization (ISO), First edition 2003-07-01, (2003).
- [69] ISO/IEC TR 9126-3, *Software engineering Product quality Part 3: Internal metrics*, The International Organization for Standardization (ISO), First edition 2003-07-01, (2003).
- [70] ISO/IEC TR 9126-4, *Software engineering Product quality Part 4: Quality in use metrics*, The International Organization for Standardization (ISO), First edition 2004-04-01, (2004).

- [71] ISO/IEC 9241-11, *Ergonomic Requirements for Office Work with Visual Display Terminals (VDTs) Part 11: Guidance on Usability*, The International Organization for Standardization (ISO), (1998).

Appendix A

Implementation in Eclipse

The Eclipse as host application provides services which the plug-in can use, including registering themselves and data exchanging with ANTLR and Erasmus plug-ins. In this way these plug-ins are dependent on these services provided by the host application and do not usually work by themselves. Conversely, the Eclipse is independent of the plug-ins, making it possible for plug-ins to be added and updated dynamically without changes to the host application [23]. Architecture of ELIDE in this Framework supports extensibility by providing well defined extension points where other plug-ins can add functionality. We know that a plug-in is the smallest pluggable component identified by the Eclipse. We can leverage the work done by other developers by integrating their plug-ins with our tool. This feature is very useful for developing a complex IDE such as ours that requires support for multiple functionalities. Loading of plug-ins is delayed until the corresponding functionality is requested. We can use this specialty to decompose ELIDE into small plug-ins, each being loaded only when required.

The overall plug-in architecture of ELIDE with extension points is shown in figure 33. It includes a generic set of components from Eclipse IDE and ELIDE plug-in set. The ELIDE plug-in set consists of ELIDE Core which is the central part of the IDE where the new service is actually composed, a set of wizards for creating projects and gathering resource information and helper view. The ELIDE Perspective binds all into a single integrated

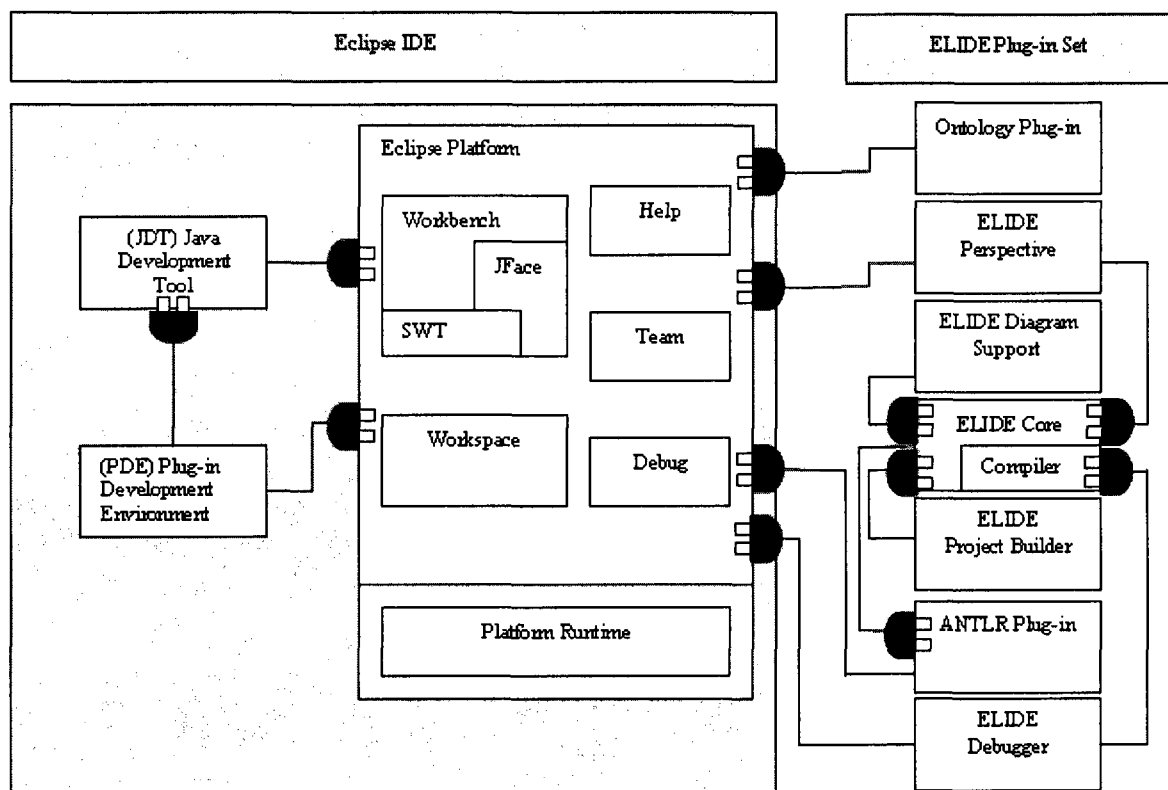


Figure 33: ELIDE Architecture using Eclipse and Antler

framework.

- **ELIDE Core:** this is the main framework for Erasmus programmer; loading, Building, Running and Testing Erasmus files as well as viewing XML will start there. It then initiates loading of concept ontology as soon as the Erasmus compiler creates AST file related to that Erasmus projects. The AST file then is used for ELIDE Diagram support and ELIDE Debugging components. We need Antlr plug-in for online recognition of tokens, highlighting and syntax checking. Also we can use the AST table generated while typing an Erasmus program.
- **Wizards:** New project creations that are repetitive and may involve a sequence of activities to be performed are made simpler in ELIDE by using guided user dialogs or wizards. The project creation wizard called, **ELIDE Project Builder Wizard**, enables Erasmus programmer to specify locations of all the necessary resources for ontology processing.
- **Help Views:** It displays a resource tree representing the organization of the project in several views. These views are developed using various existing plug-ins demonstrating easy integration with available tools. **ELIDE Perspective:** ELIDE Core and Help Views are encompassed in a single component using a facility provided by Eclipse known as perspective. A perspective is a visual container for integration among set of views and editors. The ELIDE perspective controls visibility of items of the model in the user interface.
- **Ontology Views:** It enables the user to browse through the concept ontology as well as its graphical view.
- **ELIDE Diagram support:** The ELIDE Diagram Support will be used for Graphical Representation which is specific to Erasmus language. The graphical view of Erasmus programs or projects is similar to figure 14.

the user to the next stage as a logical step from the current activity (NFR.3, NFR.4 and NFR.5). Since Erasmus IDE is produced as a plug-in, so future addition of plug-ins is possible (NFR.8). Since Eclipse is capable for adaptation, scalability and security, so does ELIDE (NFR.10, NFR.13, and NFR.15). The rest of non-functional requirements Except NFR.9 and NFR.11 are verified since ELIDE uses Eclipse. The remaining non-functional requirements are proven by performing test cases. The functional requirements are also verified by performing test cases.

Using Eclipse IDE has several benefits and drawbacks. It will be easy for user to switch between existing languages and new Erasmus in Eclipse environment. The drawback is the ELIDE would be Java, Antlr and host application dependent.

Appendix B

Erasmus programs in ELIDE

B.1 Sample Program of Client Server 1: testode1

B.1.1 Source code

B.1.2 Execution

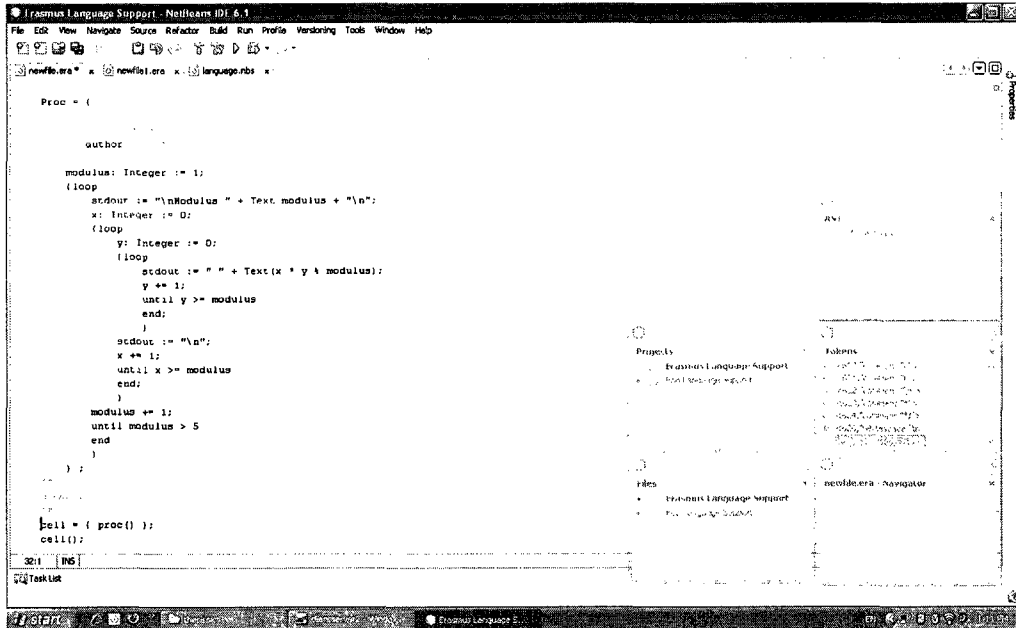


Figure 35: Sample Program 1 Source- Testodel in in ELIDE

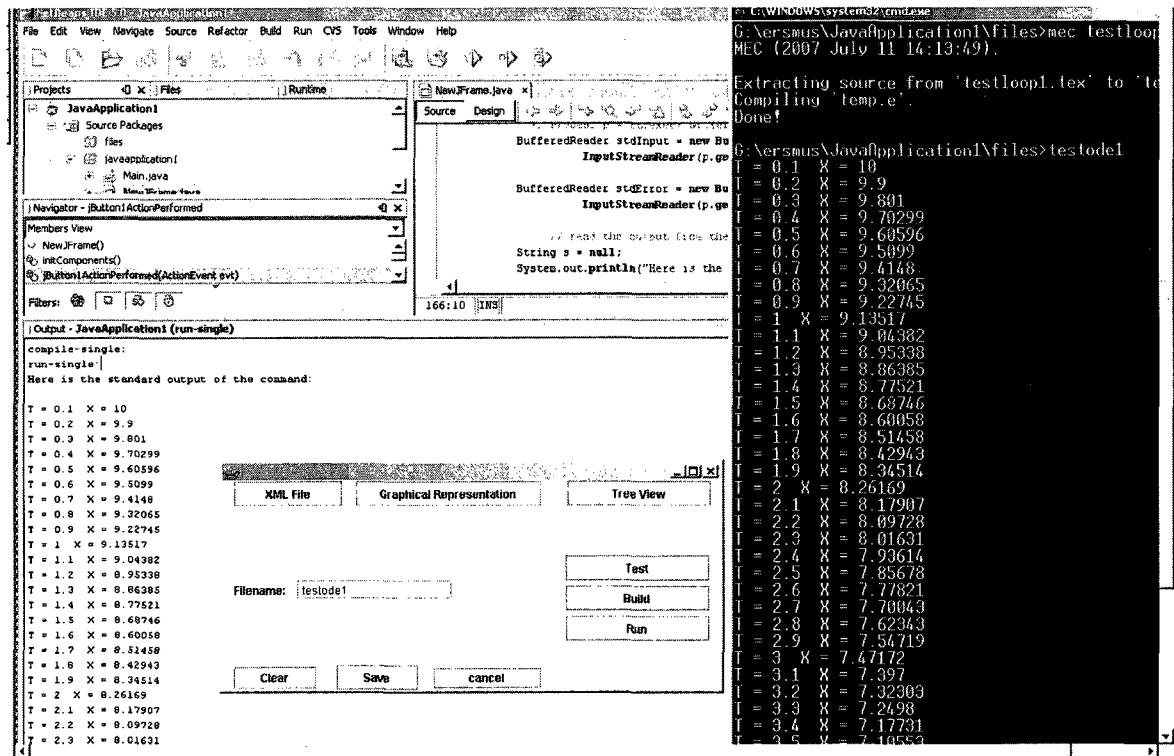


Figure 36: Sample Program 1 Execution- Testodel in ELIDE

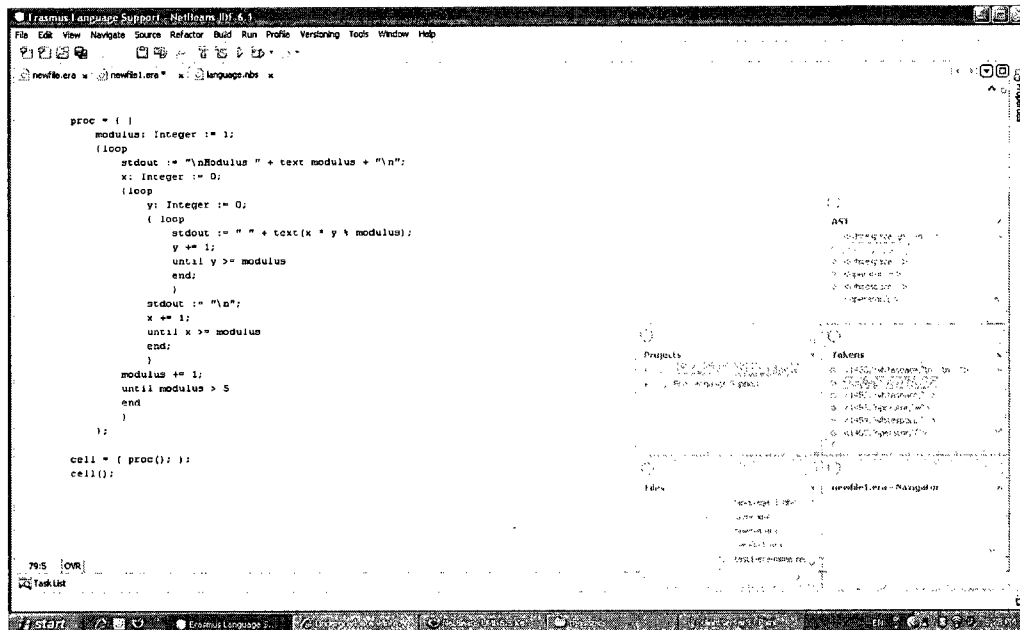


Figure 37: Sample Program 2 Source- Testloop1 in ELIDE

B.2 Sample Program of Client Server 2: testloop1

B.2.1 Source code

B.2.2 Execution

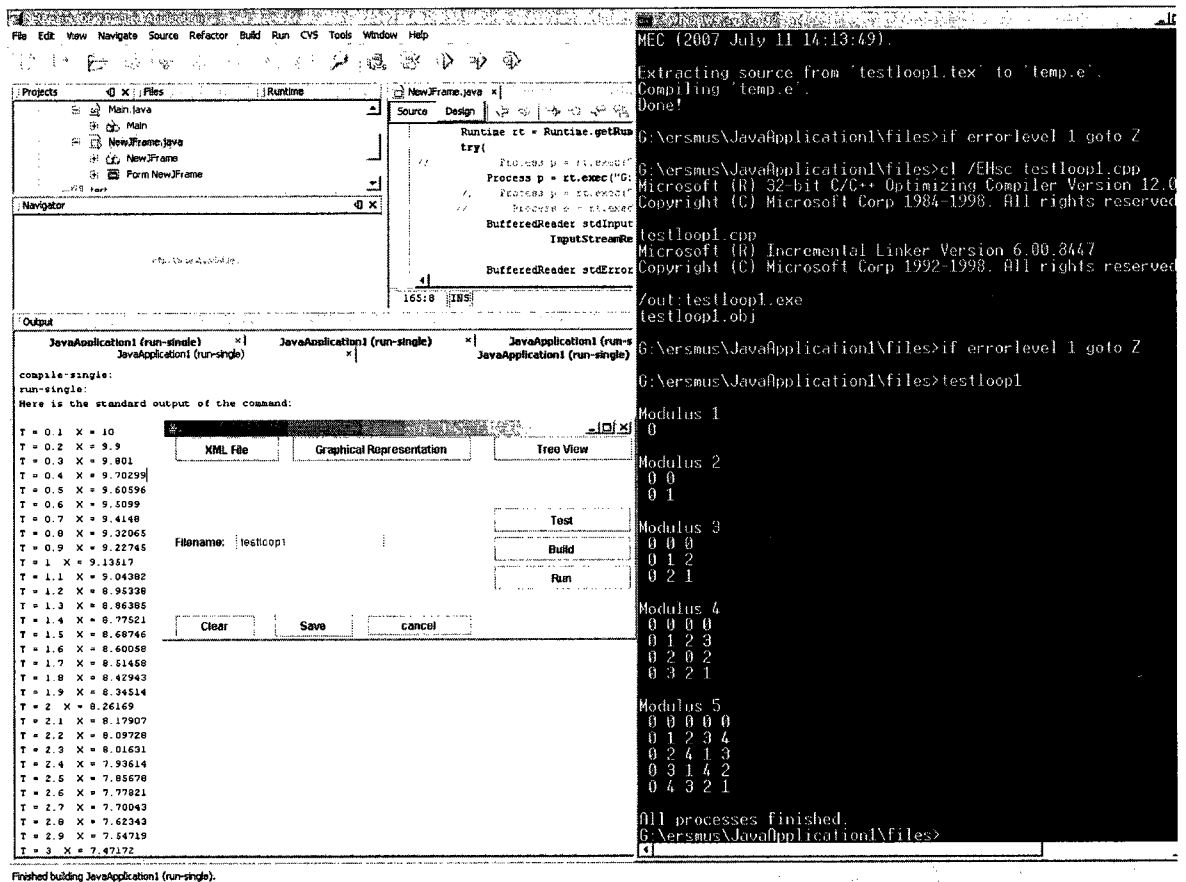


Figure 38: Sample Program 2 Execution Testloop1 in ELIDE

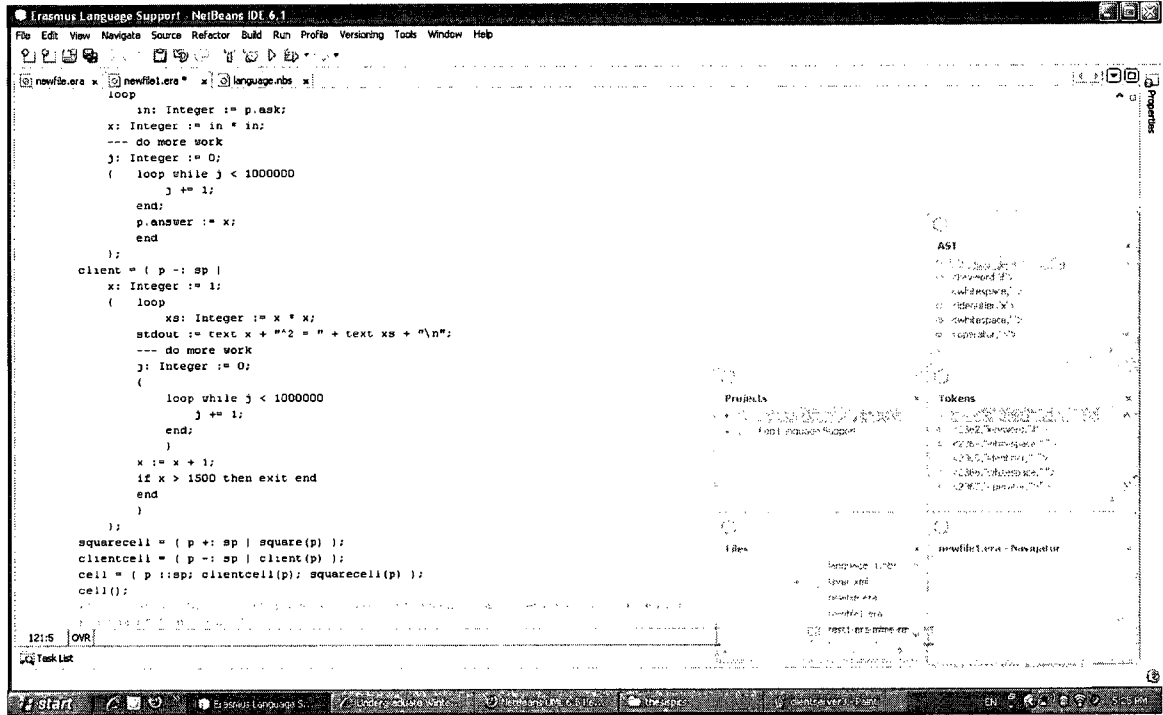


Figure 39: Sample Program 3 Source Testmod1 in ELIDE

B.3 Program sample of Client Server 3: testmod1

B.3.1 Source code

B.3.2 Execution

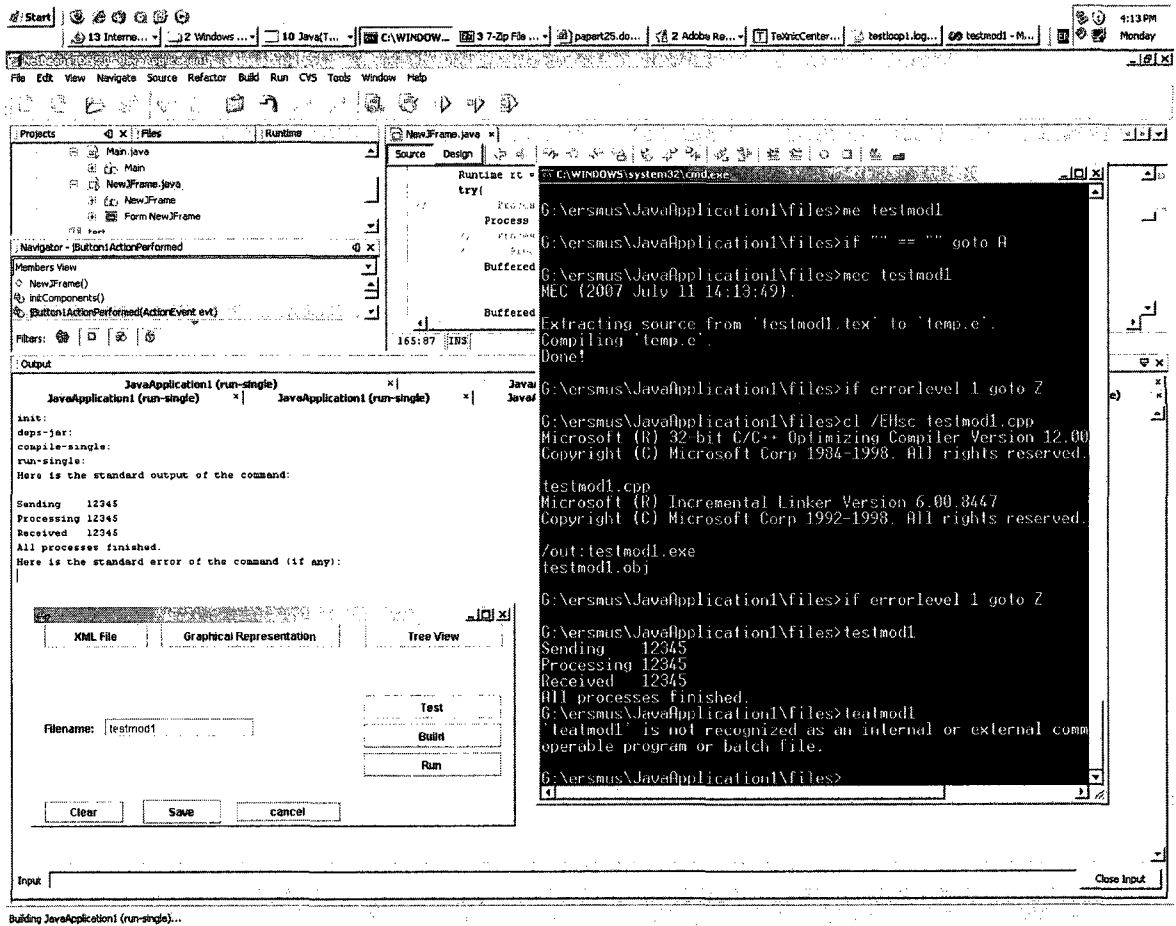


Figure 40: Sample Program 3 Execution Testmod1 in ELIDE

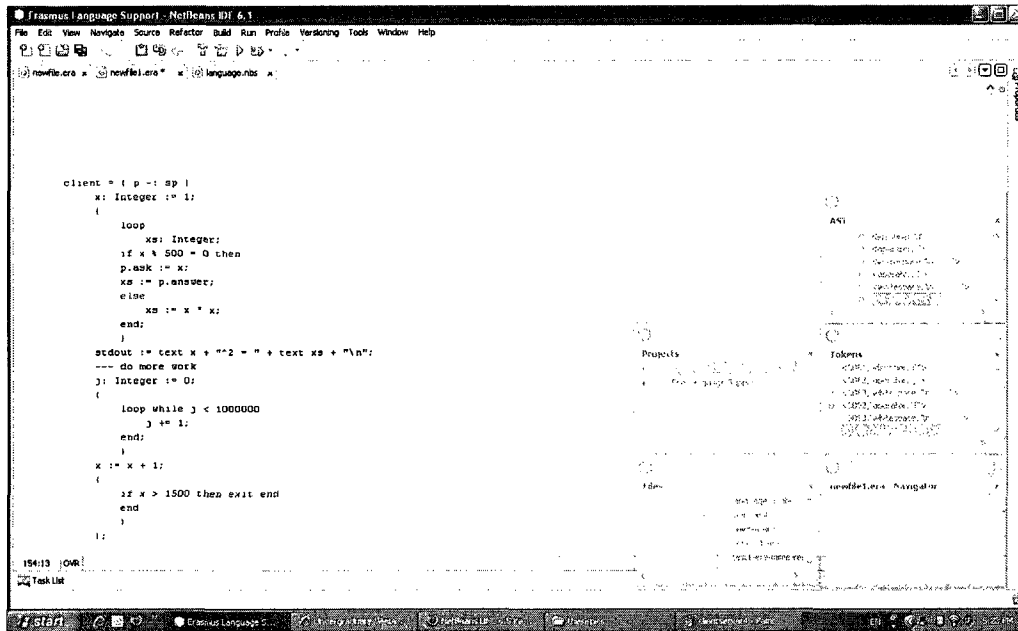


Figure 41: Program sample 4 Source- Testextract in ELIDE

B.4 Sample Program of Client Server 4: testextract

B.4.1 Source code

B.4.2 Execution

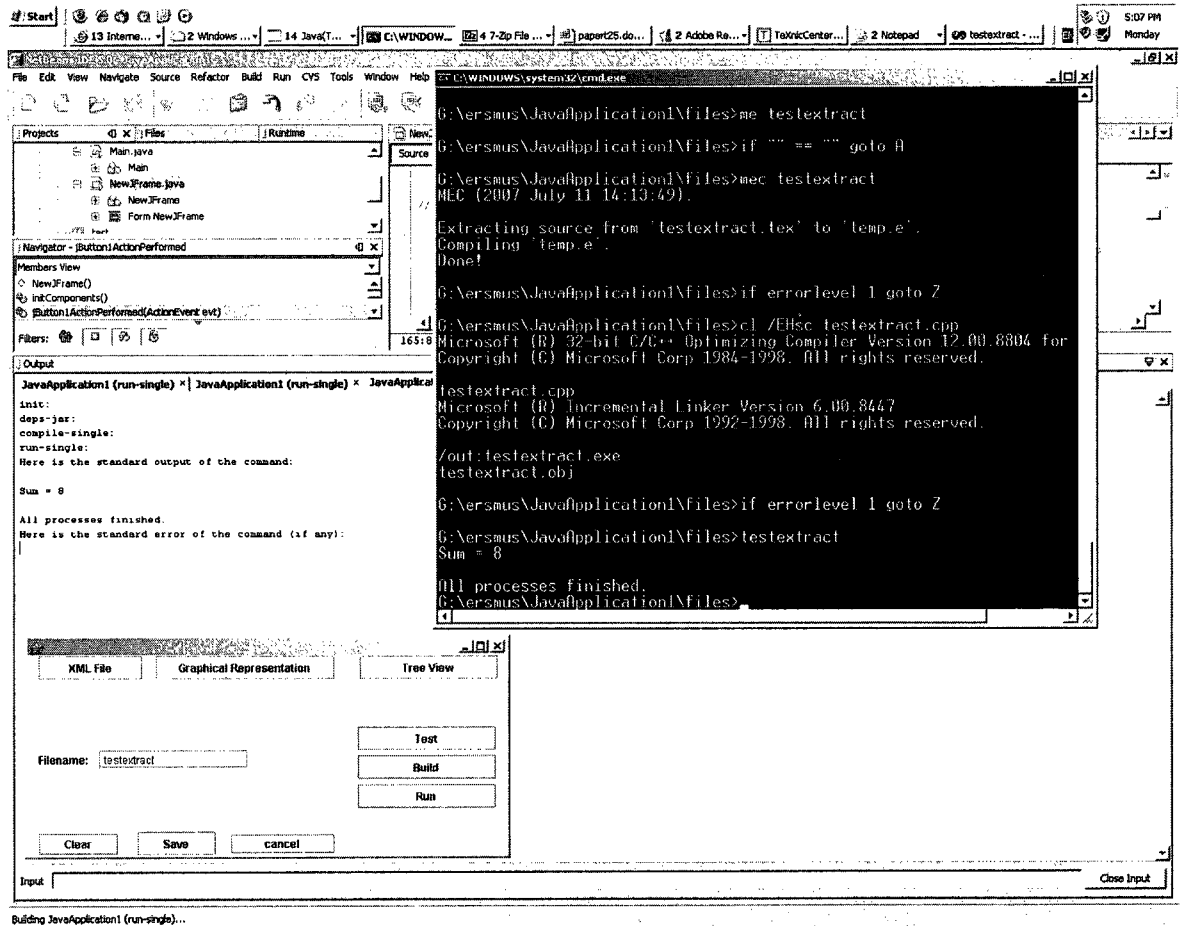


Figure 42: Sample Program 4 Execution- Testextract in ELIDE

Appendix C

Current Text Editors and IDEs

The abbreviations used in the next table are as follows:

- 1=Syntax Highlighting,
- 2=Multiple Undo/Redo,
- 3=Rectangular Block Selection,
- 4=Bracket Matching,
- 5=Auto Indentation,
- 6=Auto Completion,
- 7=Code folding,
- 8=Text Folding,
- 9=Compiler integration,
- SDI=Single Document Interface,
- SDWS=Single Document Window Splitting,
- MDI=Multiple Document Interface,
- TDI=Tabbed Document Interface,
- WS=Window Splitting,
- F=File Transfer Protocol,
- H= Hyper Text Transfer Protocol,
- S=Secure Shell Handling,

W=Web-based Distributed Authoring and Verification

Name	Programming Capability	User Interface	Remote File Editing
Acme	,2, ,4,5,6, , ,9	SDI,SDWS,TDI,MDWS	F,H,S,
Alpha	1,2,3,4,5,6, , ,9	SDI,SDWS,OW,MDWS	F, , ,
Alphatk	1,2,3,4,5,6,7,8,9	SDI,SDWS,OW,TDI,MDWS	F, , ,
Aquamacs Emacs	1,2,3,4,5,6,7,8,9	SDI,SDWS,OW,MDWS	F,H,S,W
BEdit	1,2,3,4,5,6,7,8,9	SDI,SDWS,OW,TDI	F, , ,
BDV Notepad	,2, , , , , , ,	SDI	, , ,
Bluefish	1,2,3,4,5,6, , ,	SDI,TDI	F,H, ,W
Boxer	1,2,3,4,5, , , ,	SDI,SDWS,OW,TDI,MDWS	F, , ,
ConTEXT	1,2,3,4,5,6, , ,9	OW,TDI	, , ,
Crimson Editor	1,2,3,4,5, , , ,9	SDWS,OW,TDI,MDWS	F, , ,
CRISP	1,2,3,4,5,6,7,8,9	SDI,SDWS,OW,TDI,MDWS	F,H,S,
Csued	1,2,3,4,5,6,7,8,	SDWS,OW,TDI,MDWS	, , ,
Diakonos	1,2, , ,5, , , ,	SDI	
EditPadLite	,2,3, ,5, , , ,	SDI,TDI	
EditPadPro	1,2,3,4,5, ,7,8,9	SDI,OW,TDI	F, , ,
EditPlus	1,2,3,4,5,6,7,8,	SDWS,OW,TDI,MDWS	F,H,S,W
gedit	1,2, ,4,5, , , ,9	SDI,OW,TDI	F,H,S,W
GNU Emacs	1,2,3,4,5,6,7,8,9	SDI,SDWS,OW,MDWS	F,H,S,W
JED	1,2,3,4,5,6,7,8,9	SDWS,MDWS	, , ,
jEdit	1,2,3,4,5,6,7,8,	SDI,SDWS,TDI,MDWS	F, ,S,W
JOE	1,2,3,4,5, , , ,9	SDW,TDI,MDWS	, , ,
Kate	1,2,3,4,5,6,7,8,	SDWS,OW,TDI,MDWS	F,H,S,W
MadEdit	1,2,3,4,5, , , ,	SDI,TDI	, , ,
NEdit	1,2,3,4,5,6, , ,9	SDI,SDWS,TDI,MDWS	, , ,
Notepad++	1,2,3,4,5,6,7,8,9	SDWS,OW,TDI,MDWS	, , ,
Professional Notepad	1,2, , ,5, , , ,	SDI	
PSPad	1,2,3,4,5,6, , ,9	SDWS,OW,TDI,MDWS	F, , ,
skEdit	1,2,3,4,5,6, , ,	SDI,SDWS,TDI	F, ,S,W
SlickEdit	1,2,3,4,5,6,7,8,9	SDI,SDWS,OW,TDI,MDWS	F,H,S,
Smultron		SDI,SDWS,TDI	
TextPad	1,2,3,4,5, , , ,9	SDWS,OW,TDI	, , ,
TextMate		SDI,TDI	F, , ,
TextWrangler		SDI,SDWS,OW,TDI,MDWS	
Vim	1,2,3,4,5,6,7,8,9	SDI,SDWS,OW,TDI,MDWS	F,H,S,W
SlickEdit	1,2,3,4,5,6,7,8,9	SDI,SDWS,TDI,MDWS	
XEmacs	1,2,3,4,5,6,7,8,9	SDI,SDWS,OW,TDI,MDWS	F,H,S,W

Table 7: list of Current Text Editors and IDEs

Appendix D

Study Questionnaire

External evaluation was performed by external users to check usability aspects. In this evaluation, we performed usability check with ten external users using User Testing and Questionnaire approaches. The goal of usability was to characterize the extent to which a software system can be used. The Questionnaire consisted of following questions:

- 1 How much development experience do you have?
- 2 How much do you know about NetBeans platform?
- 3 Briefly describe your development responsibilities.
- 4 Would you describe yourself as a NetBeans beginner, intermediate, or expert user? How much of your day do you spend in NetBeans?
- 5 Would you describe yourself as an Eclipse beginner, intermediate, or expert user? How much of your day do you spend in Eclipse?
- 6 Would you describe yourself as a Java/C++ beginner, intermediate, or expert user?
- 7 Would you describe yourself as an Erasmus beginner, intermediate, or expert user?
- 8 What do you like about Eclipse's navigation mechanisms and tree views? What don't you like?
- 9 What do you like about NetBeans's navigation mechanisms and tree views? What don't you like?
- 10 What do you like about ELIDE? What don't you like?

11 Please list anything else particular about ELIDE.

12 Do you think the ELIDE is user friendly?

13 Do you think the features and tools are useful?

Appendix E

Grammar and Syntax of Erasmus Language

In grammar rules, $[\dots]$ stands for an item that may be present or absent; $\{\dots\}$ stands for an item that may be present zero or more times; $\{\dots\}_c$ stands for a list of items with the character c used as a separator. A list separator is allowed at the end of a list: for example, the expression $\{X\}$, matches both “ X, X, X ” and “ $X, X, X,$ ”. Formally, with ϵ standing for ‘empty’:

$$[X] \equiv \epsilon \mid X$$

$$\{X\} \equiv \epsilon \mid X \mid X X \mid \dots$$

$$\{X\}_c \equiv [X \{c X\} [c]]$$

Fonts distinguish *non-terminal symbols*, **terminal symbols**, and information-bearing tokens such as names and literals. Symbols are written in quotes: e.g., ‘=’.

Keywords

Structures: protocol procedure thread process cell

Types: Bool Decimal Float Integer Char Text unsigned Void

Constants: false true

Functions: bool byte char decimal execute float int rand
text exists file_open_read text_open_read file_open_write
text_open_write text_content file_close file_ok file_eof
file_read file_write # %%

Qualifiers: alias fair ordered random

Operators: and div mod not or

Control: any assert cases do domain elif else end exit
for if import in loop loopselect range select skip
start step such that then to until while

Symbols

Assignment operators: <- := += -= *= /= %=

Binary operators: // + - * / % . @ &=

Comparison operators: = != < <= > >=

Protocol operators: ? * + | ; ^

Declaration operators: : ^

Separators: , ; | ->

Brackets: () [] { } <>

Characters A Character is any character in the character set used by the implementation.

(The character set is currently ASCII but will eventually be Unicode.)

A Letter is a character chosen from $\{ 'a', \dots, 'z' \} \cup \{ 'A', \dots, 'Z' \}$.

A Digit is a character chosen from $\{ '0', \dots, '9' \}$.

A HexDigit is a character chosen from $\text{Digit} \cup \{ 'a', \dots, 'f' \} \cup \{ 'A', \dots, 'F' \}$.

Identifiers Identifiers appear in the syntax as $\langle prefix \rangle$ Name, where $\langle prefix \rangle$ is chosen to suggest the role of the identifier. For example: CellName. The compiler sees only identifiers, and must infer the role from the context.

Identifier = Letter { Letter | Digit | '_' }.

The following symbols are used in the grammar to denote identifiers:

CellName ConstantName FieldName FileName MapName PortName
ProcedureName ProcessName ProtocolName TypeName VarName

Programs

Program = { *ImportDirective* | *Definition* | *Instantiation* }, .

ImportDirective = import { FileName }, .

Definitions

Definition = *ConstantDefinition*
| *TypeDefinition*
| *ProtocolDefinition*
| *ProcedureDefinition*
| *ProcessDefinition*
| *CellDefinition* .

ConstantDefinition = ConstantName ':' Type '=' Rvalue .

TypeDefinition = TypeName '=' Type .

ProcedureDefinition = ProcedureName '=' Procedure .

ProtocolDefinition = ProtocolName '=' Protocol .

ProcessDefinition = ProcessName '=' Process .

ThreadDefinition = ThreadName '=' Thread .

CellDefinition = CellName ('=' | '+=') Cell .

Descriptions

Protocol = ProtocolName
| '[' ProtocolExpression '
| protocol ProtocolExpression end .

Process = '{' [{ Declaration };] '|' Sequence '
| process [{ Declaration };] '|' Sequence end .

Thread = thread { Parameter }; ['->' { Parameter };] '|' Sequence end .

Parameter = VarName ':' ['+' | '-'] *Type* .

Procedure = procedure { *Declaration* }; '{' *Sequence* end .

Cell = '(' [{ *Declaration* }; '|'] { *Declaration* | *Instantiation* }; ')'
| cell [{ *Declaration* }; '|'] { *Declaration* | *Instantiation* }; end .

Instantiation and Invocation

Instantiation = (*CellName* | *ProcessName*) '(' { *PortName* | *Rvalue* }, ')'

ProcedureCall = *ProcedureName* '(' { *Lvalue* }, ')'

ThreadCall = *ThreadName* '(' { *Rvalue* }, ['->' { *Lvalue* },] ')'

Types

Type = *TypeName*
 | *BasicType* [*RangeExpression*]
 | *EnumeratedType*
 | *MapType*
 | *ArrayType*
 | *Direction Protocol* .

BasicType = *Void*
 | *Bool*
 | *Char*
 | *Text*
 | [*unsigned*] *Byte*
 | [*unsigned*] *Integer*
 | *Float*
 | *Decimal* .

RangeExpression = *Rvalue CompOp* '(' ')' *CompOp Rvalue* .

CompOp = '<' | '<=' .

EnumeratedType = '<' { *Identifier* }, '>' .

MapType = *Type indexes Type* .

Direction = '+' | '-' .

Protocols

ProtocolExpression = ['^'] *Declaration*
 | [*Multiplicity*] *ProtocolExpression*
 | { *ProtocolExpression* },
 | { *ProtocolExpression* }|
 | '(' *ProtocolExpression* ')' .

Multiplicity = '?' | '*' | '+' .

Declarations

Declaration = *VariableDeclaration* | *PortDeclaration* .

VariableDeclaration = [**alias**] { *VarName* }, ':' *Type*
 [('=' | ':=' | '<-') *Rvalue*] .

ChannelDeclaration = { *PortName* }, ':' *Protocol* .

PortDeclaration = { *PortName* }, ':' *Direction Protocol* [':=' *Rvalue*] .

Direction = '+' | '-' .

Sequences

Sequence = { *Statement* }; .

Statements

```
Statement = skip
          | exit
          | Assertion
          | Declaration
          | Instantiation
          | ProcedureCall
          | ThreadCall
          | Start
          | Assignment
          | If
          | Cases
          | Loop
          | Any
          | For
          | Select
          | until Rvalue
          | while Rvalue
          | MapName @ ( 'start' | 'step' ).
```


Assertion = `assert '(' Rvalue [',' Rvalue] ')'` .

Execute = `execute VarName` .

Assignment = `{ Lvalue }, AssignOp Rvalue` .

AssignOp = `':' | '<-' | '+=' | '-=' | '*=' | '/=' | '%='` .

If = `if Rvalue then Sequence`
`{ elif Rvalue then Sequence }`
`[else Sequence] end` .

Cases = `cases [Rvalue] { Guard Sequence } end` .

Loop = `loop Sequence end` .

Any = `any Comprehension do Sequence else Sequence end` .

For = `for Comprehension do Sequence end` .

Comprehension = `VarName [':' Type] [in Set] [such that Rvalue]` .

Set = `Rvalue to Rvalue [step Rvalue]`
`| Rvalue '<=' '(' [Rvalue] ')'` (`'<' | '<='`) `Rvalue`
`| Rvalue '>=' '(' [Rvalue] ')'` (`'>' | '>='`) `Rvalue`
`| [domain | range] Rvalue`
`| Type` .

Start = `start { ThreadCall }; do Sequence end` .

Select = `(select | loopselect) [Policy] { Guard Sequence } end` .

Policy = `fair | ordered | random` .

Guard = `'|' [Rvalue] '|'` .

Values

Lvalue = VarName { '[' Rvalue ['.' Rvalue] ']' } ['.' fieldName] .

Rvalue = *Lvalue*
| *Literal*
| UnOp *Rvalue*
| *Rvalue* BinOp *Rvalue*
| *Rvalue* if *Rvalue* else *Rvalue*
| *FunctionName* *Rvalue*
| *FunctionName* '(' { *Rvalue* }, ')'
| '(' *Rvalue* ')'
| MapName @ ('finish' | 'key' | 'value') .

FunctionName = bool | char | text | byte | int | decimal | float |
rand | execute | exists | '#' |
file_open | file_close | file_ok |
file_eof | file_read | file_write .

Operators

UnOp = '+' | '-' | '#' | not | .

BinOp = '/' | '+' | '-' | '*' | '/' | div | '%' | mod |
'<' | '<=' | '>' | '>=' | '=' | '!=' |
and | or | @ | &= .

TernaryOp = if ... else .

Literals

Literal = *ArrayLiteral* | *MapLiteral* | **Bool** | **Char** | **Text** | *Numeric* .

ArrayLiteral = '{' { *Literal* }, '}' .

MapLiteral = '{' { *MapPair* }, '}' .

MapPair = '(' *Literal* ',' *Literal* ')' .

Bool = **false** | **true** .

Char = ''' **Character** ''' | """ **Character** """ .

Text = ''' { **Character** } ''' | """ { **Character** } """ .

Numeric = **Integer** | **Decimal** | **Hexadecimal** .

Integer = **Digit** { **Digit** } .

Decimal = **Digit** { **Digit** } ['.' { **Digit** }] [('e' | 'E') **Digit** { **Digit** }] .

Hexadecimal = '0' 'x' **HexDigit** { **HexDigit** } .