Defence Research and Development Canada
Recherche et développement pour la défense Canada

DEFENCE R&D DÉFENSE

Development of OASIS v2

Development Environment Description

M. Savard
Neosapiens

Thales Canada, Systems Division
1 Chrysalis Way
Ottawa, ON
K2G 6P9

Project Manager: R. Proulx, 418-844-1879

Contract number: W7701-053134/001/QCL

Contract Scientific Authority: P. Charland, 418-844-4000 Ext. 4491

Defence R&D Canada – Valcartier
Contract Report
DRDC Valcartier CR 2008-332
October 2008

Canada

Development of OASIS v2

Development Environment Description

M. Savard
Neosapiens

Thales Canada, Systems Division
1 Chrysalis Way
Ottawa, ON
K2G 6P9

Project Manager: R. Proulx, 418-844-1879

Contract number: W7701-053134/001/QCL

Contract Scientific Authority: P. Charland, 418-844-4000 Ext. 4491

Defence R&D Canada – Valcartier

Contract Report

DRDC Valcartier CR 2008-332
October 2008

Author

Marco Savard

Approved by

Philippe Charland

Defence Scientist

Approved for release by

Christian Carrier

Chief Scientist

The scientific or technical validity of this Contract Report is entirely the responsibility of the Contractor and the contents do not necessarily have the approval or endorsement of Defence R&D Canada.

Development of OASIS v2

Development Environment Description

March 31, 2008

Prepared for:

Defence Research and Development Canada - Valcartier
2459, Pie-XI Blvd North
Québec (Quebec) G3J 1X5

Prepared by:
Marco Savard

Thales Canada, Systems Division
1, Chrysalis Way
Ottawa (Ontario) K2G 6P9

REVISION CONTROL SHEET

Rev. #	Pages Affected	Description	By	Date of Issue
1.0	All	New document.	M. Savard	12 May 2006
2.0	p. 4 (sec. 2.2.2), p. 6 (sec. 3.2.1), p. 11 (sec. 3.2.3), p. 17 (sec. 3.5), p. 19 (sec. 4).	Include comments on Rev. 1.0.	R. Proulx	24 May 2006

DOCUMENT APPROVAL SHEET

Name	Organization	Signature	Date of Approval

— RESTRICTIONS ON DISCLOSURE —

The information contained in this document has been prepared by Thales Systems Canada, a Division of Thales Canada Inc. for the Government of Canada in accordance with Contract W7701-053134/C. The restriction and rights on disclosure shall be in accordance with the contract terms and conditions.

TABLE OF CONTENTS

1 Introduction

The Opening up Architectures of Software-Intensive Systems (OASIS) project was initiated at DRDC Valcartier in late September 2004. When the development of the OASIS prototype itself started (April 2005), one of the first issues to address was which technologies should be chosen to develop the prototype. These technologies included the programming language in which the prototype would be coded and also the integrated development environment (IDE).

Java and C# were the two main candidates for the programming language. We were interested in a mature programming language, but also in an environment for which free open source software (FOSS) tools were abundant, especially in the software engineering area.

We had to choose the IDE, not only as the environment within which the OASIS prototype would be developed, but also as a platform on which the prototype could run. Running OASIS within an existing platform has several advantages, including not having to develop the graphical user interface (GUI) from scratch, taking advantages of existing plugins, and to demonstrate the interoperability of OASIS with other software.

The three most interesting IDEs considered were Visual C# .NET (for the C# programming language), the Eclipse platform (for Java) and the NetBeans IDE (for Java).

This document lists the criteria that were used to select the most appropriate programming language and IDE for the development of the OASIS prototype, as well as the evaluation of each criterion with respect to these languages and IDEs.

A brief description and rationale for the selected criteria, languages, and development environments identified is presented in the next section. The comparative evaluation criteria for each language and environment is performed in Section 3. The last section presents our conclusions.

2 Collection of the Needs

2.1 Required Criteria

This section lists the selected technical, domain-oriented, and market-oriented criteria on which we will compare the candidate programming languages and integrated development environments. Some of these criteria may be relevant only to compare programming languages and others to compare IDEs.

2.1.1 Technical Criteria

The technical criteria below are commonly accepted by the software development community to evaluate languages and/or development environments. These criteria do not necessarily focus on specific needs or aspects of the OASIS project. They are more general, broader in scope, and intended to provide a minimal assessment of conformance to best practices commonly encountered in software engineering activities.

2.1.1.1 Criterion #1 – Expressiveness and Programming Easiness

This criterion describes the ability to support object-oriented concepts, design pattern constructs, and the easiness for a programmer/user to use efficiently a given technology in order to develop software.

2.1.1.2 Criterion #2 - Library Completeness

This criterion estimates the number of features (number of classes and methods) provided by the standard library. Using libraries that are more complete results in lesser needs to develop missing features.

2.1.1.3 Criterion #3 - Performance

The criterion evaluates the responsiveness, memory usage, and speed of a given application or technology.

2.1.1.4 Criterion #4 - User Interface

This criterion assesses the compliance to graphical user interface standards and the respect of native look and feels.

2.1.1.5 Criterion #5 - Scalability

This criterion describes the ability to support the development of large applications, to integrate a large amount of widgets, plugins, and third-party software.

2.1.1.6 Criterion #6 - Portability

This criterion evaluates the number of platforms on which a given technology can be deployed. Because we have selected Windows as the target platform, this criterion is less important for the OASIS project.

2.1.2 OASIS Domain of Application Criteria

The OASIS domain of application criteria are the ones which are particularly important for the OASIS project. The principal domains of application of OASIS are architecture recovery and program comprehension.

2.1.2.1 Criterion #7 - Tools Integration

The criterion describes the ability for a given technology to interact with a build tool or with configuration management tools.

2.1.2.2 Criterion #8 - Source and Bytecode Analysis

The criterion describes the ability and easiness to parse source code and XML documents, to analyze bytecode (.class files in the case of Java) and to manipulate it in order to insert callback instructions.

2.1.2.3 Criterion #9 - Generation Facilities

This criterion describes the metrics (software measurements) generation facilities, template engine technologies (scripts that generate source code, XML documents, or other kinds of text files).

2.1.2.4 Criterion #10 - Modeling Facilities

This criterion lists the number of modeling tools available in a given technology and the number of implementations of unified modeling language (UML) and meta-object facility (MOF) frameworks.

2.1.2.5 Criterion #11 - Relational Databases

This criterion describes the connectivity with relational databases, the ability to support stored procedures and object-relational mapping (ORM) technologies.

2.1.2.6 Criterion #12 - Graphical Environment

This criterion describes the availability of 2D drawing technologies and the support of graphical layouts and figures for diagramming.

2.1.3 Market-Oriented Criteria

The market-oriented criteria are related to the popularity and acceptance of a given technology, without taking the technological factors into account. The popularity of a technology may reflect its technological superiority, but also the marketing efforts performed by the sponsor companies.

2.1.3.1 Criterion #13 - Availability of Users

This criterion evaluates the easiness to find competent programmers and/or users mastering a given technology.

2.1.3.2 Criterion #14 - Availability of Documentation

This criterion assesses the number of paper or electronic documents related to a given technology and thus evaluates the easiness to find documentation on a feature of interest of this technology for OASIS.

2.1.3.3 Criterion #15 - Availability of Third-Party Software

This criterion evaluates the easiness to find compatible software developed by other parties and that can be integrated into OASIS, particularly free open-source software.

2.1.3.4 Criterion #16 - Availability of Target Applications

This criterion describes the easiness to find and use target applications (applications analyzed by OASIS), particularly military-domain applications developed at DRDC.

2.1.3.5 Criterion #17 - History

This criterion describes the history of the candidate technologies; technologies with a longer history are best known, their limitations are more documented.

2.2 Environments to Evaluate

This section lists the environments selected for evaluation with a brief justification for the selection; refer to following subsections for an overview of the environments.

2.2.1 Programming Language Candidates

Java and C# are currently the most serious candidates to start a new project such as OASIS. C++ is also widely used for legacy systems, but it will eventually be replaced by C# and Java, so we do not consider it as a candidate language for the development of OASIS. Table 1 hereafter shows significant milestones in the development of both languages.

Jun 1991	Oak - Java's ancestor.	Jun 2000	Start of C#.
Jan 1996	Java 1.0.	Jun 2002	Java 1.4 (assert).
Feb 1997	Java 1.1 (event model).	Jul 2003	C# 2.0.
Mar 1998	First version of Swing.	Sep 2004	Java 5.0 (generics, enums).
Dec 1998	Java 1.2 (collections, strictfp).	Sep 2005	C# 3.0.
May 2000	Java 1.3 (XML).		

Table 1: Java and C# Timeline

2.2.2 Visual C# .NET

Visual C# is the Microsoft IDE for the development of C# applications over .NET (Microsoft's framework for connecting systems, information, and devices through Web services). We will evaluate Visual C# in details only if C# is chosen as the programming language to develop OASIS.

2.2.3 Java with Eclipse IDE

Eclipse is an open source development project initiated by IBM. It is written in Java and is a descendant of VisualAge, an IDE developed by IBM and originally written in Smalltalk. Eclipse is mainly used to develop Java applications, although other programming languages could be

supported. For example, the C++ Development Tools (CDT) project allows a programmer to develop C++ applications using Eclipse. Figure 1 illustrates a view of the Eclipse IDE.

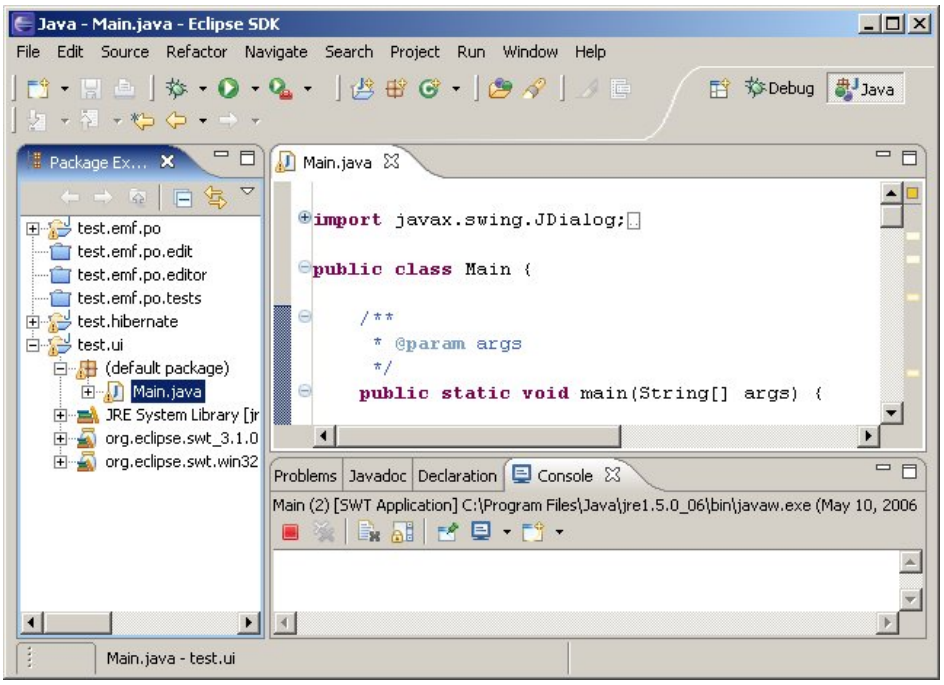


Figure 1: The Eclipse IDE

2.2.4 Java with NetBeans IDE

NetBeans is an IDE supported by Sun Microsystems, the company that developed the Java programming language. The NetBeans development started earlier than the one of Eclipse, as can be seen from Table 2.

1996	Xelfi (NetBeans' ancestor) developped in Prague, Czechoslovakia.	Dec 2004	NetBeans 4.0.
2000	NetBeans open-source, sponsored by Sun.	May 2005	NetBeans 4.1.
Nov 2001	Eclipse foundation begins.	Jun 2005	Eclipse 3.1.
Jun 2003	NetBeans 3.5.	Jan 2006	Eclipse 3.1.2.
Mar 2004	NetBeans 3.6.	Jan 2006	NetBeans 5.0.
Jun 2004	Eclipse 3.0.		

Table 2: Eclipse and NetBeans timeline

3 Evaluation

3.1 Scope and Methodology

This section gives a brief description of the methodology used for evaluating the candidate technologies.

Because of the limited time to perform the study (five days to understand and collect the project needs, to get familiar with technologies, to perform searches, and to write this document), many arguments are commonly accepted statements rather than statements based on a detailed analysis. The source references include Internet surveys on similar studies. Because the author has no working experience with C#, the survey also includes informal interviews with C# developers.

Neither configuration nor experimentation with the evaluated environments was performed within the scope of this study.

3.2 Technical Comparison

3.2.1 Comparing C# and Java

Java and C# are two modern object-oriented (OO) languages sharing several features, such as bytecode intermediate language, garbage collection, reflection, single inheritance for classes, inner classes, exception handling, enumerations, generics, etc. Design patterns are common in both Java and C# frameworks. The language expressiveness and learning curve are also equivalent.

Some Java programmers consider that Java is closer to the UML principles than C#. For instance, both Java and UML suggests lowercase identifier for methods, while the C# usage suggests using capitalized identifiers for methods.

The standard libraries of both Java and C# are comparable. As two mainstream languages, their libraries include functionalities to develop graphical user interfaces, input-output accesses, network and database connections, collections, and so on.

C# is usually considered faster than Java because it is highly integrated to .NET and Windows low-level libraries.

Java is considered portable on a higher number of platforms than C#, as a general language, but Visual C# for .NET takes advantage of the .NET framework to facilitate development of smart client applications, i.e., applications using XML based Web services that can be deployed and communicate over a wide range of devices and enterprise Microsoft applications (e.g., Office XP) combined with rich client offline capability.

Java and C# are somewhat equivalent from a technical point of view, and each of them could have been chosen to develop OASIS.

3.2.2 Comparing Java/Eclipse and Java/NetBeans

Eclipse is an IDE based on an open-source standard widget toolkit (SWT) for Java, while NetBeans is based on the abstract widget toolkit (AWT) and Swing. Before comparing Eclipse and NetBeans, we will compare Swing and SWT to figure out which one is the best choice for the needs of OASIS. Comparing Swing and SWT may be a factor to select Eclipse rather than NetBeans (but this is just one factor among several other ones).

3.2.2.1 Comparing AWT/Swing to SWT

AWT uses heavyweight (native) widgets. Because AWT runs on a large number of platforms, only the most common widgets are supported. The fact that only a small set of common widgets are available in AWT almost disqualifies it to develop modern GUI-based applications.

Swing uses lightweight widgets. Swing interacts with the underlying operating system with only low-level drawing functions, and all complex widgets are emulated with basic drawing functions. The advantage is portability: all Swing applications look the same on all platforms. The lack of performance is the price to pay, because the operating system optimizations for drawing widgets are by-passed.

Swing uses *pluggable look and feels* to render applications more compliant to Windows GUI. Pluggable look and feels give good (but not perfect) results for the look, but often disappointing results for the feel (drag-and-drop features, resize of windows, and so on). Also, pluggable look and feels are released after a new platform look and feel is available.

SWT is a middleweight-component framework. Its approach lies between AWT and Swing: SWT uses native widgets if they exist, and emulates them otherwise. This approach combines the best of the two worlds: the performance of SWT is as good as AWT because native widgets are used, and the number of widgets supported equals (and even exceeds) the number of Swing widgets.

Point-by-point comparison of Swing and SWT

Number of widgets supported: The small amount of supported widgets in AWT almost disqualifies it to develop modern GUI-based applications. Swing offers a large choice of widgets, but lacks common widgets such as a directory chooser, a wizard, or a font chooser. Often, developers must wait for availability of a widget (spinners were not offered before Java 1.4).

Look and feel: An unfair criticism against Swing is that applications developed with this technology do not look as Windows applications. Several Swing applications were developed without using the Windows *pluggable look and feel* (PLAF), giving a look not compliant with the Windows standard. Using the Windows pluggable look and feel gives fairly good results, but not entirely compliant with Windows standards. Also, pluggable look and feels are released after the release of a new version of Windows, making applications look old. See appendix A for a visual example.

Performance: Because all widgets are emulated in Swing, the performance is not as good as using native components.

Memory usage: Emulated widgets in Swing are often greedy in their use of memory. They are not manually disposed; they are garbage collected when the Virtual Machine (VM) decides to run the garbage collector. The disposal of widgets in Swing allows a better memory management.

Programming easiness: The SWT application programming interface (API) is simpler than the one of Swing. For instance, the functionality of the SWT Button class is provided by at least five classes (JButton, JRadioButton, JCheckBox, JtoggleButton, and AbstractButton). The composition pattern used in SWT to associate a child widget to its parent is less error-prone than the aggregation pattern, preventing the association of a child widget with more than one composite. The untyped listeners in SWT allow more efficient code. On the other hand, the widget disposal demands more discipline from a SWT programmer to prevent memory leaks caused by non- disposed widgets.

Number of platforms: SWT is currently implemented on only four platforms (Windows, Solaris, Red Hat Linux, and Macintosh), compared to AWT and Swing that run on all platforms on which Java is implemented (currently, more than 20 different platforms).

Table 3 outlines the comparison between AWT, Swing, and SWT.

Comparison Item	AWT	Swing	SWT
Number of widgets supported	low	high	**very high**
Look and Feel • Look	poor	good (with the PLnF)	**very good**
• Feel	poor	poor	**good**
• Up-to-date	medium	medium	**very up-to-date**
Performance	**good**	medium	**good**
Memory usage	**good**	medium	**good**
Programming easiness • API	**very simple**	very complex	complex
• Composition	aggregation	aggregation	**composition**
• Widget Disposal	**automatic**	**automatic**	manual
• Untyped Listeners	none	none	**supported**
Number of platforms	**high**	**high**	medium

Table 3: Comparison of AWT, Swing, and SWT

The preceding table indicates that SWT would be technically superior to the two other alternatives. The fact that AWT scores better than Swing is misleading, since the number of widgets supported and the look and feel are much more important issues than the other points.

3.2.2.2 Comparing Eclipse to NetBeans

Eclipse and NetBeans share several similarities, such as:

- Eclipse has plugins; NetBeans too, called modules.
- Both have a manifest file describing the plugin (the module).
- Both can extend the main menu bar and the tool bar.
- Both support editors (editing a particular kind of file).

Both can add a view (called window component in NetBeans). Table 4 hereafter describes the Eclipse and NetBeans features.

Eclipse	NetBeans
• Better architecture of plugins. • GUI is faster, because of SWT. • Support of perspectives. • More scalable. • Eclipse has a more powerful refactoring feature.	• Built-in GUI builder (Matisse)[1]. • Better integration to J2EE and web development (EJB, JSP, etc.). • Integrated profiling profiling tool[2].

Table 4: Comparison of the Eclipse and NetBeans architecture

In Eclipse, with the exception of a small core, most anything is a plugin. In NetBeans, modules represent a small part of the application (Figure 2). Also, NetBeans modules extend NetBeans but they can hardly be extended.

[1] Eclipse has some plugins to build GUIs, e.g., the VisualEditor, but are not as powerful as Matisse.
[2] Eclipse has TPTP, but it is more difficult to use.

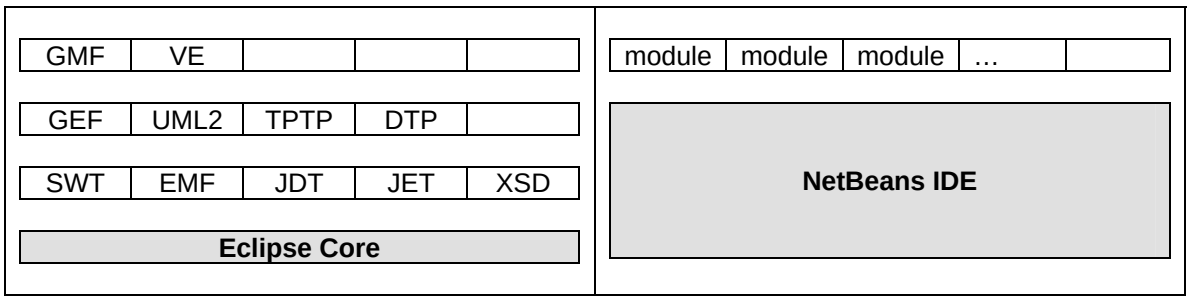


Figure 2: Comparison of Eclipse plugins and NetBeans modules

Adding a window component to NetBeans requires specifying where to put the new component to allow a new tab to be displayed on the designated pane. For example:

- output window, on the bottom pane;
- navigator, on the left pane;
- properties, below the navigator;
- editor, on the central pane;
- common palette;
- debugger pane, and
- left sliding side.

Inserting too many plugins makes NetBeans overloaded because there is nothing such as the perspective concept of Eclipse.

3.2.3 Summary of Technical Comparison

The following table summarizes the technical comparison between C# and Java, as well as between the Eclipse and NetBeans development environments.

Criterion	Visual C# .NET	Java / Eclipse	Java / NetBeans
Criterion #1 Expressiveness, programming easiness	C# equivalent to Java	Eclipse and NetBeans equivalent	Eclipse and NetBeans equivalent
Criterion #2 Library completeness	C# equivalent to Java	Eclipse libraries (org.eclipse packages) are more complete than the NetBeans ones	NetBeans less complete than Eclipse
Criterion #3 Performance	C# is considered faster than Java	Because of SWT, Eclipse is more responsive than NetBeans	Because of Swing, NetBeans is slower than Eclipse
Criterion #4 User Interface (programming)	C# equivalent to Java	Because of SWT, Eclipse is more compliant to Windows standards	Because of Swing, NetBeans is less compliant to Windows standards
Criterion #5 Scalability	C# equivalent to Java	The Eclipse architecture makes it more scalable than NetBeans	NetBeans is less scalable than Eclipse
Criterion #6 Portability	Java is considered more portable than C# as a language but .NET	Eclipse and NetBeans are equivalent	Eclipse and NetBeans are equivalent

	allows better support to design smart client applications		

Table 5: Summary of technical comparison

3.3 Domain-Oriented Comparison

3.3.1 Comparing C# and Java as the Programming Language for OASIS

Here are the identified features of interest for OASIS to select a programming language (Java or C#):

- **Tools Integrations:**
  - Build tools, configuration manager.
- **Reading Files:**
  - Source code parsers, XML parsers, bytecode analyzers.
- **Generating Files:**
  - Metrics generators, template engines.
- **Modeling:**
  - MOF implementation, FOSS UML CASE tools.
- **RDBMS:**
  - Connectivity, stored procedures, triggers, API, schema mapping.
- **Graphics:**
  - 2D drawings, figures and layout, diagramming framework.

Table 6 hereafter lists a number of products (commercial or FOSS) available that address these features.

Features of Interest	Java	C#
Tool Interactions		
Build Tools	Ant[3], tightly integrated with Java.	Makefile and Nant.
Configuration Management	API to interact with CVS and Subversion.	Microsoft Visual SourceSafe
Reading Files		
Source Code Parsing	JavaCC, ANTLR, SableCC, Coco/R, JDT.	Coco/R[4]
XML Parsers	DOM, SAX built-in in Java 1.3+, Xerces free parser for older versions.	System.XML namespace (limited to DOM[5]).
Bytecode Handlers	BCEL and some others.	Nothing found[6]
Generating Files		
Metrics Generators	JMetrics, Eclipse Metrics plugin.	SLOCCount
Template Engines	JET, Velocity.	CodeSmith[7]
Modeling		
MOF implementations	EMF.	Nothing found.
FOSS UML CASE Tools	Tigris ArgoUML, Omondo Eclipse, Oracle JDeveloper and ten others[8].	No free CASE tools found[9].

[3] Although Apache Ant scripts can invoke C# targets, Ant is a Java-based build tool that is better integrated with Java.
[4] Although JavaCC and ANTLR are Java-based parsers, they provide grammars to parse C#, but the parsers generated by these tools are in Java.
[5] This namespace contains classes to construct a DOM tree, but without SAX callback methods.
[6] By examining the classes contained in the System.Reflection.Emit namespace, we have found a functionality to examine MSIL code, but not to manipulate it (http://msdn.microsoft.com/library/).
[7] http://www.codesmithtools.com/, but it is not a free tool.

RDBMS		
DBMS Connectivity	JDBC for all major DBMS.	ADO.NET for SQL Server, Oracle 8i, DB2[10] and ODBC.
Stored Procedures, Triggers, API	Oracle, DB2, Informix, Sybase.	SQL Server
FOSS DBMS Mapping	Hibernate, EJB.	Nothing found[11]
Graphics		
2D drawing	Java2D, draw2D.	System.Drawing namespace.
Figures and Layouts	draw2D	None[12]
Diagramming Framework	GEF	None

Table 6: Related features for available products

3.3.2 Comparing C# and Java Target Applications

A target application is an application being analyzed by OASIS, in order to allow an architect to understand it. In theory, the programming language used to develop target applications is independent of the choice of the programming language used to develop OASIS itself. OASIS written in Java could analyze Java, C#, or C++ target applications, while OASIS written in C# could analyze Java applications. In practice, if OASIS is developed in Java, it could use internal libraries to parse and analyze Java applications more easily than target applications written in another language.

It is currently easier to parse Java applications than C# applications for two reasons: the existence of FOSS parser generators and the complexity of the C# syntax.

There are several FOSS Java parser generators e.g., ANTLR and JavaCC. These tools generate Java parsers. Of course, they can parse languages other than Java. They can parse C# code because both tools provide C# grammars). If an analyzer application written in C# has to parse C# code using ANTLR or JavaCC, it will have to invoke Java code, because the parsers generated by ANTLR and Java are written in Java.

The C# syntax is more complex, and thus harder to parse, because it has more keywords than Java, its grammar contains more rules, it supports preprocessor directives and unsafe blocks (with pointers), it implements more language constructs (properties, indexers, delegates, operators overloading) and supports more predefined types. Of course, one could say that once the C# grammar is available, the syntax complexity is not an issue. But grammars never include semantic actions. Semantic actions are actions inserted in the grammars and invoked when the generated parser detects a syntax rule while parsing. A parser generated with a grammar with no semantic actions is nothing more than a syntax validation tool. It is of limited interest for the OASIS project. The higher the number of semantic rules, the higher is the number of semantic actions we have to insert in the grammar to analyze the source code.

Because C# is harder to parse and to analyze than Java, it is preferable to choose target applications written in Java. Consequently, if we choose target applications written in Java, it is preferable to develop OASIS itself in Java.

[8] Based on CASE tools listed at http://www.objectsbydesign.com/, web page consulted the Apr 25, 2006.
[9] Ibid.
[10] "ADO.NET uses managed providers to connect to a database. Managed providers are database drivers that expose APIs by using classes that are based on managed code (which is Microsoft Intermediate Language, or MSIL, under the covers). Today, managed providers exist for SQL Server, Oracle 8i, and DB2"' Microsoft .NET and J2EE Interoperability Toolkit, Simon Guest, Microsoft Press, 2004, page 242.
[11] Some tools, such as VBeXpress, exist but are not free.
[12] By examining the classes found in the System.Drawing and System.Drawing.Drawing2D namespaces, we did not find any figure, layout or diagramming functionality. (ref: \http://msdn.microsoft.com/library/)

3.3.3 Comparing Java/Eclipse and Java/NetBeans

As we did in the previous section, because Eclipse is based on SWT and NetBeans is based on Swing, it could be useful to compare Swing and SWT before comparing Eclipse and NetBeans.

3.3.3.1 Comparing AWT/Swing to SWT

The technical points previously discussed do not have an equal weight for the purpose of the OASIS project.

The number of widgets supported is the most important issue, since on a project with limited resources, money is better spent on developing comprehension functionalities rather than graphical widgets. Also, as a R&D project, OASIS uses the latest technologies available, and therefore, we cannot afford to wait for the availability of a given widget if this widget is already available in a different environment. On these two points, SWT scores better than Swing.

OASIS must analyze large applications and should be able to display huge diagrams. For these reasons, performance (rapid interactions with the GUI) and memory usage (free memory as soon as it is no longer required) issues are crucial. SWT again is a better choice than Swing for these issues.

OASIS should use software visualization features. The Mylar project offers visualization functionalities. Because Mylar is an Eclipse plugin, this is another point in favor of SWT.

3.3.3.2 Comparing Eclipse and NetBeans

Eclipse and NetBeans are equivalent for the integration of build tools, such as ANT scripts, or for the integration with a configuration management (CM) tool. The parser generators and XML parsers can be used in both Eclipse and NetBeans environments. Eclipse provides an editor plugin for the ANTLR parser generator, but it is mainly a syntax-coloring editor and not a major issue to discriminate between Eclipse and NetBeans.

Eclipse provides JET as its template engine, but Apache Velocity can be used in a NetBeans environment.

The Eclipse Modeling Framework (EMF), a meta-object facility (MOF) implementation, is a set of powerful modeling and code generator modules that has no known counterparts in NetBeans.

The relational database management system (RDBMS) connectivity is equivalent in both IDEs. Each IDE has its strengths and its weaknesses, enterprise Java Beans (EJB) are better supported in NetBeans, but EMF-based technologies such as DTP and Elver[13] have no counterparts in NetBeans.

The drawing2D features are more advanced in SWT than in Swing, as explained in the previous section. Specifically, diagramming layout (not to be confused with widget layout) and the support of multi-compartment figures (graphical element in a diagram) have no equivalents in Java2D (the Swing counterpart for SWT draw2D).

[13] See www.elver.org

3.3.4 Summary of Domain-Oriented Comparison

The following table summarizes the domain application comparison between Visual C# .NET, Java/Eclipse and Java/NetBeans languages and development environments.

Criterion	Visual C# .NET	Java / Eclipse	Java / NetBeans
Criterion #7 Tools Integration	More tools integration in Java.	Eclipse and NetBeans equivalent.	Eclipse and NetBeans equivalent.
Criterion #8 Source and Bytecode Analysis	More parser generators in Java.	Eclipse and NetBeans equivalent.	Eclipse and NetBeans equivalent.
Criterion #9 Generation Facilities	More generation facilities in Java.	The JET technology exists in Eclipse.	Apache Velocity can be used in place of JET.
Criterion #10 Modeling Facilities	More generation facilities in Java.	Some modeling plugins (e.g., the EMF framework).	No EMF equivalent.
Criterion #11 Relational Databases	C# equivalent to Java.	Connectivity features equivalent. The DTP plugin to model database.	Connectivity features equivalent. EJB better supported in NetBeans.
Criterion #12 Graphical Environment	C# equivalent to Java.	The SWT/draw2D more complete than its Swing counterpart.	The SWT/draw2D more complete than its Swing counterpart.

Table 7: The OASIS domain comparison summary

3.4 Market-Oriented Comparison

3.4.1 Comparing C# and Java

Java is currently used by more programmers[14] and endorsed by several major companies. Therefore, it is easier to find competent programmers in Java than in C#.

More documentation is available for java than for C#. A search for "Java" on the Amazon online bookstore gives 8,211 entries, while the same search for C# returns 514 results[15].

OASIS must, as much as possible, reuse free-open source libraries instead of re-inventing the wheel and re-implement existing software. Availability of open-source libraries is crucial for OASIS. Because Java is older and more mature, it is easier to find Java FOSS tools than C# FOSS tools[16].

We didn't conduct an exhaustive analysis to assess Java vs. C# penetration in the military application domain, but an informal observation of the projects at DRDC Valcartier and formal observations of Java vs. C# penetration in the development of enterprise applications in general show that there are far more applications developed in Java than in C#.

[14] Although it is hard to evaluate the number of programmers for each language, the Bureau of Labor Statistics identifies C, C++ and Java as the most often used languages (http://www.bls.gov/oco/ocos267.htm)
[15] Search performed on April 27th, 2006 on amazon.com.
[16] A short survey made on the sourceforge.net website, made on Apr 25th, 2006, identified 18972 projects written in Java and 3791 projects written in C#.

Since OASIS is a software comprehension tool and it is more likely that the applications subjected to the comprehension process are written in Java, it is of a greater interest to use the same language to develop OASIS as the one used to develop the analyzed applications.

Java is an older language than C#, so it is expected that its features and limitations are more known than the ones of C#. On the other hand, an older language does not necessary mean a more mature language. A new language usually benefits from lessons learned from an older language but it can take time for a new language to implement all the constructs from an older language. For instance, it took eight years for Java to support the generics (C++ templates) and enumerations (already supported in the C programming language).

Although C# is younger than Java, it reached a fair level of maturity very rapidly as Microsoft used lessons learned from Java users experience and it supports almost all its constructs.

Longevity is not always a discriminating factor between both languages in terms of technical maturity. It is however indirectly discriminating in terms of market and mind share among developers and IT managers.

3.4.2 Comparing Java/Eclipse and Java/NetBeans

AWT/Swing versus SWT

Swing is much more used than SWT, has a longer history, and it is easier to find programmers who know Swing rather than SWT. It is however worth mentioning that the learning curve for SWT is about three weeks for a developer who already knows Swing (which is not very high, but not insignificant in a five-month development project).

Eclipse versus NetBeans

Better choice of plugins: A search on software development projects on sourceforge.net gives 543 results for "Eclipse plugin" and 171 results for "NetBeans plugin" (search performed on April 28, 2006).

More documentation available: A search on amazon.com gives 13,016 results for "Eclipse" and only 19 results for "NetBeans". The search results are misleading since Eclipse is a common noun and may refer to the astronomical phenomenon rather than the IDE. However, the 30 first results give 19 references on the IDE, and 11 are not related to the IDE. A search on camelot.ca (a bookstore specialized in computers and software) gives 3 results for NetBeans and 26 results for Eclipse.

More active community: The Eclipse annual conference (www.eclipsecon.org) is a four-day event with hundred of technical presentations while the NetBeans day, as its name suggests, is a one-day event with only seven technical presentations[17].

3.4.3 Summary of Market-Oriented Comparison

The following table summarizes the market-oriented comparison.

Criterion	Visual C# .NET	Java / Eclipse	Java / NetBeans
Criterion #13	More Java developers than C# developers on	More Eclipse developers	Less NetBeans developers on the

[17] Reference: http://www.netbeans.org/community/articles/javaone/2006/nb-day.html

Availability of users	the market.	on the market.	market.
Criterion #14 Availability of documentation	More Java documentation.	More documentation on Eclipse.	Less documentation on NetBeans.
Criterion #15 Availability of third-party software	More 3rd-party software, especially FOSS, in Java.	More Eclipse plugins than NetBeans modules.	Less NetBeans modules than Eclipse plugins.
Criterion #16 Availability of target applications	More target applications in Java.	Not a relevant point.	Not a relevant point.
Criterion #17 History	Longer history of Java.	Eclipse and NetBeans are comparable.	Eclipse and NetBeans comparable.

Table 8: Market comparison summary

3.5 Summary Table of Overall Results

Table 9 collects all the results from the previous sections and summarizes the comparison between the languages/development environments evaluated.

Criterion	Visual C# .NET	Java / Eclipse	Java / NetBeans
Criterion #1 Expressiveness, programming easiness	C# equivalent to Java.	Eclipse and NetBeans equivalent.	Eclipse and NetBeans equivalent,
Criterion #2 Library completeness	C# equivalent to Java.	Eclipse libraries (org.eclipse packages) are more complete than NetBeans ones.	NetBeans less complete than Eclipse.
Criterion #3 Performance	C# is considered faster than Java.	Because of SWT, Eclipse is more responsive than NetBeans.	Because of Swing, NetBeans is slower than Eclipse.
Criterion #4 User Interface	C# equivalent to Java.	Because of SWT, Eclipse is more compliant to Windows standards.	Because of Swing, NetBeans is less compliant to Windows standards.
Criterion #5 Scalability	C# equivalent to Java.	Because of the Eclipse architecture, Eclipse is more scalable than NetBeans.	NetBeans is less scalable than Eclipse.
Criterion #6 Portability	Java is considered more portable than C#. .NET better is suited for smart client applications.	Eclipse and NetBeans equivalent.	Eclipse and NetBeans equivalent.

Criterion #7 Tools Integration	More tools integration in Java than in C#.	Eclipse and NetBeans equivalent.	Eclipse and NetBeans equivalent.
Criterion #8 Source and Bytecode Analysis	More parser generators in Java than in C#.	Eclipse and NetBeans equivalent.	Eclipse and NetBeans equivalent.
Criterion #9 Generation Facilities	More generation facilities in Java than in C#.	JET technology available for Eclipse.	Apache Velocity can be used instead of JET.
Criterion #10 Modeling Facilities	More generation facilities in Java than in C#.	Some modeling plugins. The EMF framework.	No EMF equivalent.
Criterion #11 Relational Databases	C# equivalent to Java.	Connectivity features equivalent. DTP plugin to model database.	Connectivity features equivalent. EJB better supported in NetBeans.
Criterion #12 Graphical Environment	C# equivalent to Java.	SWT/draw2D more complete than its Swing counterpart.	Swing is less complete than its SWT/draw2D counterpart.
Criterion #13 Availability of users	More Java developers than C# developers on the market.	More Eclipse developers on the market.	Less NetBeans developers on the market.
Criterion #14 Availability of documentation	More documentation available for Java than for C#.	More documentation on Eclipse.	Less documentation on NetBeans.
Criterion #15 Availability of third-party software	More 3rd-party software, especially FOSS, in Java than in C#.	More Eclipse plugins than NetBeans modules.	Less NetBeans modules than Eclipse plugins.
Criterion #16 Availability of target applications	More target applications in Java than in C#.	Not a relevant point.	Not a relevant point.
Criterion #17 History	Java has a longer history compared to C#.	Eclipse and NetBeans equivalent.	Eclipse and NetBeans equivalent.

Table 9: Overall comparison summary

4 Conclusions and Recommendation

Based on the overall results presented in table 9 of Section 3.5, the following conclusions can be drawn:

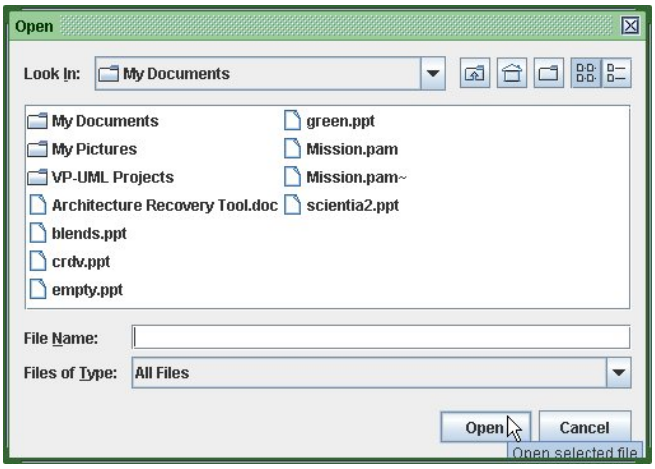
- Java rates more favorably than C# with respect to the needs of the OASIS project:
    - General technical criteria do not discriminate much both languages except perhaps for a slightly superior expected performance in execution for C# and, within the Microsoft .NET framework, better support to design smart client applications. However, the latter is not a very important issue, as OASIS is not designed as an enterprise wide deployment application.
    - Application domain and market criteria evaluation clearly indicate advantages of Java over C#.
- Given that Java will be the language of choice, the comparison between the Eclipse and NetBeans IDE is in favor of Eclipse. Of particular importance to OASIS are advantages of Eclipse over NetBeans with respect to features such as:
    - SWT advantages over AWT/Swing for graphics (completeness, performance).
    - Eclipse's number of plugins and tools (FOSS), size and activity of development community and projects.
    - Eclipse EMF model framework and richness of libraries.

These conclusions confirm the choices of Java and Eclipse as the language and development environment suited for the OASIS project, and further support our recommendation to continue project development based on these choices.

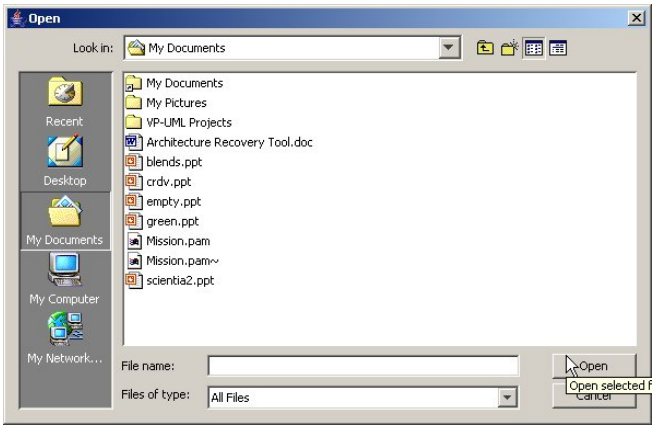
Appendix A

Look and feels of Swing and SWT

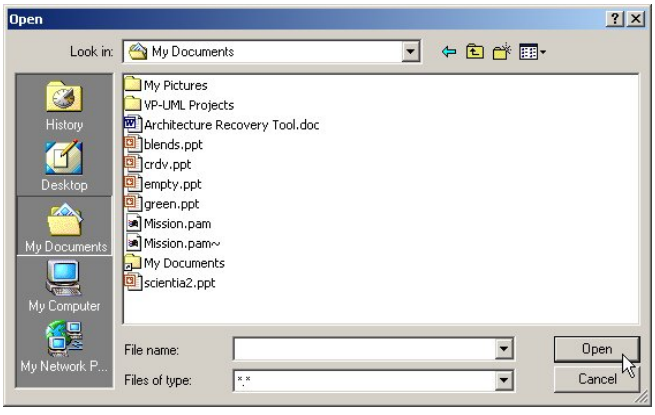
1. A Swing dialog that uses the Java look and feel (several Swing applications use this look and feel by default):



2. A Swing dialog that uses the Windows look and feel:



3. A SWT dialog (SWT applications are fully compliant with Windows UI guidelines):



5 Distribution List

INTERNAL DISTRIBUTION

1 - Philippe Charland (Scientific Authority)

2 - Document Library

EXTERNAL DISTRIBUTION

1 - DRDKIM (PDF file)

1 - Library and Archives Canada

DOCUMENT CONTROL DATA

1. ORIGINATOR (name and address)	2. SECURITY CLASSIFICATION (Including special warning terms if applicable)
Defence Research and Development Canada - Valcartier 2459, Pie-XI Blvd North Québec, Québec G3J 1X5 Canada	Unclassified

3. TITLE (Its classification should be indicated by the appropriate abbreviation (S, C, R or U)
Development of OASIS v2: Development Environment Description (U)

4. AUTHORS (Last name, first name, middle initial. If military, show rank, e.g. Doe, Maj. John E.)
Savard, Marco

5. DATE OF PUBLICATION (month and year)	6a. NO. OF PAGES	6b .NO. OF REFERENCES
October 2008	27	0

7. DESCRIPTIVE NOTES (the category of the document, e.g. technical report, technical note or memorandum. Give the inclusive dates when a specific reporting period is covered.)
Contract Report

8. SPONSORING ACTIVITY (name and address)
Defence Research and Development Canada - Valcartier
2459, Pie-XI Blvd North
Québec, Québec
G3J 1X5 Canada

9a. PROJECT OR GRANT NO. (Please specify whether project or grant)	9b. CONTRACT NO.
13jf	W7701-5-3134

10a. ORIGINATOR'S DOCUMENT NUMBER	10b. OTHER DOCUMENT NOS
DRDC Valcartier CR 2008-332	N/A

11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification)

- ☒ Unlimited distribution
- ☐ Restricted to contractors in approved countries (specify)
- ☐ Restricted to Canadian contractors (with need-to-know)
- ☐ Restricted to Government (with need-to-know)
- ☐ Restricted to Defense departments
- ☐ Others

12. DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in 11) is possible, a wider announcement audience may be selected.)
Unlimited

13. ABSTRACT (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual).

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus, e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

Defence R&D Canada

Canada's Leader in Defence
and National Security
Science and Technology

R & D pour la défense Canada

Chef de file au Canada en matière
de science et de technologie pour
la défense et la sécurité nationale

DEFENCE R&D DÉFENSE

www.drdc-rddc.gc.ca