

2003

Improving the programming language translation process via static structure abstraction and algorithmic code transliteration

Robert W. Chandler
Edith Cowan University

Follow this and additional works at: https://ro.ecu.edu.au/theses_hons



Part of the [Management Information Systems Commons](#)

Recommended Citation

Chandler, R. W. (2003). *Improving the programming language translation process via static structure abstraction and algorithmic code transliteration*. https://ro.ecu.edu.au/theses_hons/134

This Thesis is posted at Research Online.
https://ro.ecu.edu.au/theses_hons/134

Edith Cowan University

Copyright Warning

You may print or download ONE copy of this document for the purpose of your own research or study.

The University does not authorize you to copy, communicate or otherwise make available electronically to any other person any copyright material contained on this site.

You are reminded of the following:

- Copyright owners are entitled to take legal action against persons who infringe their copyright.
- A reproduction of material that is protected by copyright may be a copyright infringement.
- A court may impose penalties and award damages in relation to offences and infringements relating to copyright material. Higher penalties may apply, and higher damages may be awarded, for offences and infringements involving the conversion of material into digital or electronic form.

Improving the Programming Language Translation Process via Static Structure Abstraction and Algorithmic Code Transliteration

**A thesis submitted in partial fulfilment of the requirements for the degree of
Bachelor of Science Honours (Software Engineering)**

**By: Robert W. Chandler
Student ID: 2003078**

**Faculty of Computing, Health and Science
Edith Cowan University**

Supervisor: Michael Collins

Date of submission: 14th November 2003

USE OF THESIS

The Use of Thesis statement is not included in this version of the thesis.

Abstract

Fully automated programming language translation has been described as an unrealistic goal, with previous research being limited by a ceiling of 90% successful code translation. The key issues hindering automatic translation efficacy are the:

- maintainability of the translated constructs;
- full utilisation of the target language's features; and
- amount of manual intervention required to complete the translation process.

This study has concentrated on demonstrating improvements to the translation process by introducing the programming-language-independent, Unified Modelling Language (UML) and Computer Assisted Software Engineering (CASE) tools to the legacy-system language migration project. UML and CASE tools may be used to abstract the static framework of the source application to reduce the so-called "opaqueness" of the translated constructs, yielding a significantly more maintainable product.

The UML and CASE tools also enhance use of the target language features, through forward engineering of the native constructs of the target language during the reproduction of the static framework. Source application algorithmic code translation, performed as a separate process using transliteration, may preserve maximum functionality of the source application after completion of the static structure translation process. Introduction of the UML and CASE tools in conjunction with algorithmic code transliteration offers a reduction of the manual intervention required to complete the translation process.

Table of Contents


1	Introduction.....	1
2	The Problem.....	4
2.1	Background to the Study.....	4
2.2	Significance of the Study.....	5
2.3	Statement of the Problem.....	6
2.4	Research Questions.....	7
2.5	Chapter Summary.....	8
3	A Review of the Literature.....	9
3.1	Studies into System Evolution through Code Migration.....	9
3.2	Studies Similar to this Study.....	13
3.3	Grammar development.....	16
3.4	Application Selection.....	20
3.5	Chapter Summary.....	25
4	Research Design.....	26
4.1	General Method.....	26
4.2	Specific Procedures.....	27
4.3	Potential Enhancements not Incorporated in this Study.....	33
4.4	Chapter Summary.....	34
5	Implementation and Findings.....	35
5.1	Phase 1: The Static Structure.....	35
5.2	Phase 2: The Algorithmic Code.....	51
5.3	Phase 3: The Analysis and Findings.....	62
5.4	Findings.....	65
5.5	Discussion.....	66
5.6	Evidence Found To Support the Research Questions.....	69
5.7	Chapter Summary.....	72
6	Conclusions.....	73
	Appendices.....	75
	Appendix A: Sample application – Towers of Hanoi.....	75
	Appendix B: The generated JADE Towers of Hanoi schema file.....	77
	Appendix C: Rational Rose model file grammar.....	79
	Appendix D: A subset of C++ grammar.....	85
	Appendix E: The JADE language grammar.....	88
	Appendix F: Sample application – Building Inheritance.....	95
	Appendix G: The generated JADE Building Inheritance schema file.....	97
	Appendix H: The converted Towers of Hanoi schema file.....	99
	Appendix I: The converted Building Inheritance schema file.....	103
	Appendix J: Glossary of terms.....	108
	References.....	113

Table of Figures

Figure 1: Example UML class diagram showing inheritance in a building context..	12
Figure 2: Framework properties associated with JADE models.....	16
Figure 3: A simple assignment statement grammar (Sebesta, 1999, p. 113).....	17
Figure 4: A sample parse tree, (Sebesta, 1999, p. 114).....	17
Figure 5: Building a parser with ProGrammar (NorKen, 2003, p. 14).	19
Figure 6: Terms used in program transformation (Harsu, 2000, p. 6).	20
Figure 7: The Towers of Hanoi problem.	21
Figure 8: A high-level view of the process for the study.....	27
Figure 9: Reverse engineered VC++ application.....	36
Figure 10: The JADE root-schema class diagram.....	37
Figure 11: VC++ model class attribute properties.	39
Figure 12: JADE model class attribute properties.	39
Figure 13: The Towers of Hanoi program at run-time.....	40
Figure 14: Inheritance sample application class diagram.	41
Figure 15: Re-assigning the application to the target language.	42
Figure 16: The JADE connection dialog.....	42
Figure 17: Assigning the class objects to the JADE schema.	43
Figure 18: Class relations, the parent class and the map file.	44
Figure 19: Re-assigning attribute types.....	45
Figure 20: The manipulated inheritance model	46
Figure 21: Import/Export progress report dialog.	47
Figure 22: The converted model Design Object	48
Figure 23: The blank model Design Object	48
Figure 24: Converted model file Tower object.....	49
Figure 25: Class method references to VC++.....	49
Figure 26: VC++ path reference.	50
Figure 27: Extra VC++ node definitions.....	50
Figure 28: Towers of Hanoi parse tree.....	52
Figure 29: Regularly used algorithm example in pseudocode.....	54
Figure 30: A mapping of the data types (Terekhov and Verhoef 2000, p. 105)	55
Figure 31: Setup of a target file parser.....	56
Figure 32: Searching a method for algorithmic code.....	57
Figure 33: Contents of current_statement_list.	58
Figure 34: Towers of Hanoi addDisks() parse tree.....	59
Figure 35: for_list node value.	60
Figure 36: JADE equivalent to Figure 35.	60
Figure 37: Converts Figure 35 to Figure 36.....	61
Figure 38: Method of inc_statement conversion.....	61
Figure 39: A converted schema imported into JADE.....	63
Figure 40: The building inheritance output as depicted in Figure 39	64
Figure 41: Converted array assigning 'for loop'.....	67
Figure 42: The original C++ 'for loop'.....	67
Figure 43: Method signature alteration.....	68

Declaration

I declare that this thesis does not incorporate without acknowledgment any material previously submitted for a degree in any institution of higher education, and that, to the best of my knowledge and belief, it does not contain any material previously published or written by any other person except where due acknowledgment is made.

Signature: 

Date: Jan 8th 2004

Acknowledgements

Firstly, I would like to acknowledge the support Michael Collins has provided before and during this project. His encouragement and belief in me has been immeasurably helpful. In addition, I would like to acknowledge Dr. Leisa Armstrong and Chris Bolan, both of whom acted as reviewers for the proposal of this project, along with Daphne Brosnan for her grammar and structure insights during the early stages. Their comments and suggestions were invaluable.

I would like to thank the members of the 'M-team', the 'mentees' and honours students wading doggedly through the mire each Wednesday evening. In addition, thanks must go to Judy Clayden for late night editing and her pedantic approach to all manner of references and to Dr. Karen Anderson for allowing me to step out of another project, thereby giving me the opportunity to engross myself in this investigation.

Thanks to my family and especially my wife Karen, who provided me with an environment in which to work comfortably and quietly, for what must have seemed like unending hours, days, weeks and months... Her endurance in entertaining our children throughout this time was inspirational; I hope that soon we will be able to have normal adult conversations once again.

Finally, Harry and Emily, my babies, now maybe we can have some fun. It may not be long before you are using the computer all day and night!

1 Introduction

This chapter introduces the problems associated with legacy system programming language conversion projects, a description of the aims of this study and a synopsis of the remainder of this document.

The literature in the area of programming language translation, e.g. Harsu (2000), Moynihan and Wallis (1991) and Terekhov (2001), suggests that fully automated translation of one programming language to another is an unrealistic goal. Problems cited with the traditional process may be listed under the following points:

1. maintainability of the translated “objects” or “constructs”;
2. utilisation of the features of the target language; and
3. need for manual intervention, either before or after the translation process.

Moynihan and Wallis (1991, p. 396) expressed concern over the first point regarding the conversion of the constructs of the source application to another High-Level Programming Language (HLPL), resulting in “opaque” constructs that are difficult to maintain. Also of concern to Moynihan & Wallis (1991), is the second point in that a target system, created by the translation, may not benefit fully from those features that made the target language attractive for the translation. The third point relates to the amount of source-code that may be translated automatically Harsu (2000), Moynihan and Wallis (1991) and Terekhov (2001). Harsu (2000), for example, reports the amount of code translated automatically at 90% of her legacy-system project’s source-code, a significant improvement over the 70% - 80% success rate reported by Markosian, Newcomb, Brand, Burson, and Kitzmiller (1994), 6 years earlier.

This study establishes the Unified Modelling Language (UML) and Computer Aided Software Engineering (CASE) tools as essential components, capable of

enhancing the maintainability and efficiency of translated software and reducing the amount of source code requiring manual intervention. A consequence of the use of such tools is the reduction of costs normally associated with manual language translation processes.

Chapter 2 presents a background to the study and outlines why researchers suggest that modern applications must evolve. The significance of the study is presented followed by a description of the problems normally associated with the traditional methods of programming language translation. The research questions are then stated.

Chapter 3 provides a review of the literature relevant to the field of programming language translation and the use of the UML and CASE tools. The review describes system evolution, Source-to-Source translation and highlights similar studies. The literature reviewed is used to support the justification for the approach taken in this project.

Chapter 4 combines the needs outlined in the introduction and background with the foundations provided by the literature review to develop the concepts presented in this study. The research design and method are described, detailing the specific processes used to generate the verifiable outcomes of this study.

Chapter 5 describes the findings of this study and presents evidence to answer the fundamental research questions. The chapter provides relevant components of those source and target model schemas that were compared and contrasted to support the evidence that validates the findings of this study.

Chapter 6 concludes the study. Implications of this study are discussed together with the potential for further investigation and research in this field. A summary of the initial study proposal and the outcomes and strategies developed during the course of the investigation are also outlined in the chapter. For the

reader's convenience, a glossary of terms used in this document has been provided in Appendix J.

In summary, conventional automatic translation of legacy systems leaves, at best, 10% of the total Lines of Code (LOC) for manual intervention to complete and/or refine the process. Where non-trivial systems are to be converted, such manual intervention involves considerable costs. The study concludes that such costs may be minimised via conjoint activities of translation of both static and algorithmic source application components.

2 The Problem

2.1 Background to the Study

The term “legacy-system” is used to describe outdated applications built using obsolescent languages (Ducasse, 2001). However, Ducasse (2001) concedes that some applications, although written using modern, Object-Oriented (OO), programming languages such as C++, Java and Smalltalk, may be considered as legacy-systems. Those who adopted the OO paradigm early, according to Demeyer, Rieger, & Tichelaar (1998), may now be faced with evolving existing OO systems. Ducasse (2001) lists the following reasons why information systems must evolve:

- original developers may no longer be available;
- outdated development methods;
- monolithic systems;
- code bloat;
- lack of documentation;
- misuse of language constructs; and / or
- Business Process Re-engineering (BPR).

Another compelling reason for evolving an existing system is that some of the internal algorithmic functionality within a legacy-system is too valuable to discard and too expensive to reproduce (Skarmstad, Khan, & Rashid, 1999). If such internal code is worth saving, then language translation may be one method of taking advantage of the features of a more versatile programming language. Few modern programming languages match the versatility of JADE (O'Sullivan, 2000), an application programming technology capable of deployment on most modern platforms.

According to O'Sullivan (2000, p. 6), JADE provides such versatility via features including:

- easily developed web functionality;
- automatic Hyper-Text Mark-up Language (HTML) and Java generation; and
- smart client technology.

JADE connects to existing relational databases and to its own persistent OO database management system. Its versatility renders JADE an effective choice as the target language, when planning legacy Information System (IS) evolution. Another valid reason for selecting a language such as JADE is presented by Terekhov and Verhoef (2000), who state that "Freshmen would expect that the more equal [sic] the languages are, the more easy a conversion would be". When translating between similar languages, for example, C++ to JAVA, the developer must contend with "semantic differences that we cannot even detect syntactically" Terekhov and Verhoef (2000). Such problems associated with similar language translations are added to the problems of language translation associated with syntax and type conversion. Hence deciding on the target language is only one of the planning decisions required prior to commencement. Another essential planning decision involves weighing the costs of a fully automatic translator against the effort required for manual translation of the same source-code (Moynihan & Wallis, 1991).

2.2 Significance of the Study

The cost of manual language translation of source-code was estimated by Ben Wilson, cited by Cowley (2003), at between \$US8.00 and \$US20.00 per LOC: a considerable expense in large translation projects.

One such conversion performed by Terekhov (2001) was from a system containing more than 1.5 million LOC in High-Productivity System (HPS) source language to the target languages of Visual Basic and COBOL. In that conversion, Terekhov achieved between 80% and 90% automatic translation of the original

system. To estimate the cost involved in the manual translation of the remainder, we use the figures presented by Cowley (2003). Using the upper extreme of Terekhov's (2001) 90% success in automatic translation, there remained approximately 150,000 LOC requiring manual intervention. At the lowest rate per LOC estimated by Cowley (2003), i.e. \$US8.00 per LOC, the cost of residual manual translation of Terekhov's project would have exceeded \$US1.2 million.

In a smaller example, where Kontogiannis et al. (1998) translated 300,000 lines of PL/IX code to C++, approximately 30,000 LOC may have required manual intervention. Again, using a basis of \$US8.00 per LOC, the cost of residual manual translation for this project would have exceeded \$US240,000.

Both of the cost estimation examples immediately above involved the use of the traditional method of translating programming languages. In this, the source application is mapped statement-by-statement to an equivalent representation in the target language: a method referred to by Waters (1988) as transliteration. Waters (1988) presented the idea of translating applications from one programming language to another, via abstraction and reimplementation. It was concluded by Waters (1988, p. 1227) that the benefits of translation via abstraction and reimplementation, at that time, were "more of a promise than a reality". This study shows that with the CASE tools available today, Waters' (1988) idea is now closer to reality.

2.3 Statement of the Problem

This study offers improvements in automatic programming-language translation through a process that:

- reverse engineers an existing, operational C++ legacy application's source-code into a UML 'class model' schema file;
- converts the C++ UML schema file into a JADE equivalent schema file;
- imports the JADE root-schema into the model;
- exports the features of the converted model to a JADE working schema file;

- extends the generated JADE schema file to include the necessary sections, rendering the schema file syntactically correct; and then
- generates the algorithmic content of each class method using dynamic code transliteration.

This process produces a JADE schema file, ready for importation into the JADE development environment. The improvement of the language translation process, in consequence of the application of Rational Rose implementation of UML (Rose/UML) and versatility offered in JADE, is shown to reduce significantly the cost of legacy system evolution, by reducing the need for manual intervention.

2.4 Research Questions

Where separation of static and algorithmic components of code for forward engineering of a legacy system is achieved, then may a reduction of manual intervention be realised in automated code conversion?

2.4.1 The major components of the above question are:

1. Which model properties within a Rational Rose / UML model file are associated with the reverse engineered application's programming language?
2. Which components of the JADE schema file, produced by the RoseJADELINK add-in, may be used to construct the static framework in preparation for code migration?
3. What improvement in the ratio of automatically to manually translated LOC in a legacy system may be achieved using the abstraction and re-implementation approach?

2.5 Chapter Summary

The problems associated with programming language code migration were introduced and described. Traditionally, code migration is considered an expensive solution; a reason why programming language translation is often overlooked as an option for legacy system evolution. Such expense of traditional methods provides a justification for the investigation into alternative methods of code migration and, hence, to justify the significance of this study. Finally, the research questions associated with the study were presented.

3 A Review of the Literature

3.1 Studies into System Evolution through Code Migration

Terekhov & Verhoef (2000, p. 123) offer the following warnings regarding system evolution and language conversion:

- conversions are difficult;
- conversions are always more difficult than you think;
- the more semantic-equivalence is necessary, the more impossible [sic] it (the conversion) becomes;
- going from a rich language to a minimal language is impossible; and
- easy conversion is an oxymoron.

Notwithstanding the warnings of Terekhov & Verhoef (2000, p. 123), research teams, for example, Kazman, O'Brien, & Verhoef, (2002), Seacord, Comella-Dorda, Lewis, Place, & Plakosh, (2001), Ducasse (2001) and Harsu (2000), have attempted to overcome the problems associated with the migration of one programming language to another.

Seacord, Plakosh, & Lewis, (2003) recognise that the goals of legacy-system modernisation projects often differ from those involved in the engineering of new applications. When engineering a new application the goals of a project usually revolve around providing the client with a product of the quality specified, delivered on time and within the agreed budget.

Seacord et al., (2003) define the goals of legacy-system modernisation as the minimisation of:

- development and deployment costs;
- the time required to develop and deploy the modernised system;
- risks to the successful completion of the modernisation process;
- the modernised system's complexity;
- and the maximisation of the modernised system's performance; and
- quality of both the product and the modernisation process.

However, not all of the goals defined by Seacord et al. (2003) may be achievable in all circumstances. In some situations tradeoffs may be necessary. For example, the minimisation of the complexity of a modernised system might involve significantly more time for deployment and development than the time required to develop a new equivalent application. Therefore the developer must employ a strategy to take into account the goals of the planned modernisation project.

R. Seacord et al. (2001) believe that a prerequisite to developing a modernisation strategy requires a developer to understand the structure of the legacy-system. One method available to a developer to gain an understanding of the structure of a legacy-system is to use reverse engineering as part of the modernisation process (Chikofsky & Cross, 1990, p. 15). Chikofsky and Cross (1990) explain that the modernisation of a legacy-system usually includes:

- reverse engineering; followed by
- inspection of the system's architecture; and then
- forward engineering.

Reverse Engineering: To begin the process of reverse engineering, a CASE tool, such as ROSE/UML, scans the source code of an application, collecting the following static elements, listed by Boggs and Boggs (2002, p. 365) :

- classes;
- attributes;
- operations;
- relationships; and
- packages.

Reverse engineering reveals the structural components of the application together with their inter-connecting relationships. A diagrammatic representation of the components and their relationships, forming the static structure of the application, is then presented via UML class diagrams.

Booch et al. (1999, p. 459) define a class as “a set of objects that share the same attributes, operations, relationships and semantics.” Each of the classes in a class diagram shows the data-holding qualities, or attributes, of the class as well as the internal and externally visible methods or operations. The qualities of a class diagram are highlighted in Figure 1, which shows a UML model of a building inheritance application.

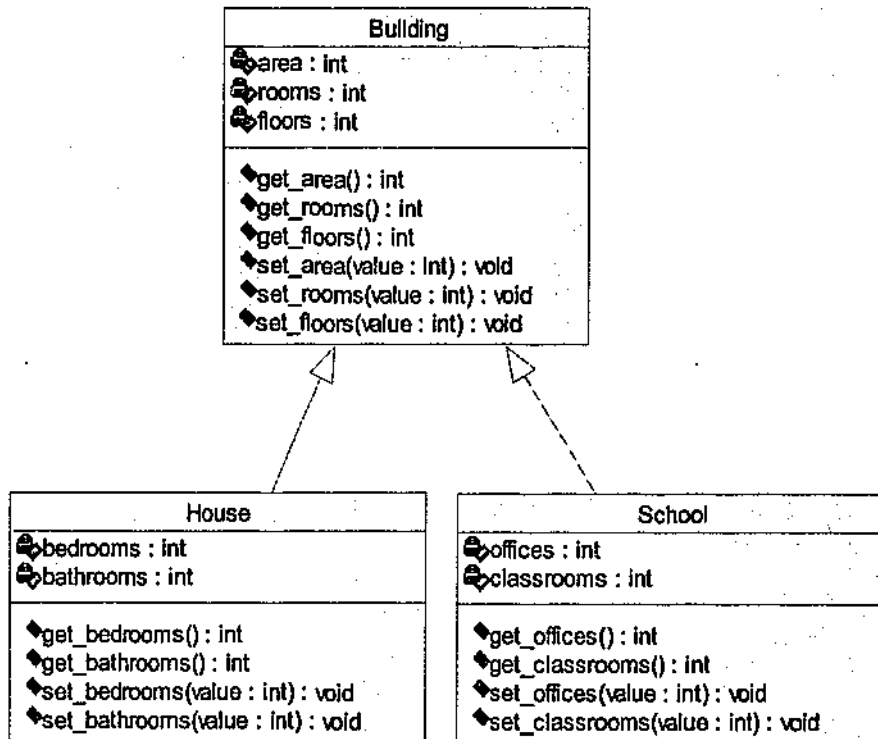


Figure 1: Example UML class diagram showing inheritance in a building context.

Inspection of the System's Architecture: On completion of the reverse engineering process, the developer is able to inspect and alter the static structure of the application. However, ROSE/UML does not capture the algorithmic source code, within the reverse engineering process as its focus is on the static structure. In consequence, during the forward engineering process, the developer is obliged to implement manually any source code within the new system's methods Krishnamoorthy (2003).

The UML gives a developer a clearer understanding of the functionality of the legacy-system, by exposing the operations and attributes associated with each of the classes within the application. Furthermore, the exposure of the components and their relationships improves the perceived transparency of the converted internal

constructs by using the UML in the forward engineering process. The lack of transparency of the traditionally converted constructs, referred to by Moynihan & Wallis (1991) and Harsu (2000) has been a significant problem with contemporary language translation processes. Such a lack of transparency is referred to as “opaqueness”.

Forward Engineering: The target static structure generated by ROSE/UML during the forward engineering or schema export process is representative of the elements created in the UML during the reverse engineering of the legacy-system. Completion of the conversion of the target system is then achieved by the translation and inclusion of the algorithmic-source-code into that static structure.

3.2 Studies Similar to this Study

Waters (1988, p. 1207) suggested that traditional source-to-source translators render the maintenance of a translated system difficult to understand. Furthermore, Waters (1988, p. 1225) estimated that of the translation systems available at the time, most were “capable of handling only 90% of the source language”. Waters’ estimate has been supported by the experiments of Harsu (2000) and Terekhov (2001), suggesting that no significant improvement in automated language translation process has been realised since 1988. Additionally, Waters (1988, p. 1225) states that source-to-source translators should not be referred to as “automatic systems”, instead they should be referred to as “human-assisted translation systems”. In order to achieve an accurate translation, Waters (1988) deduced that the developer must alter the source code of either or both of the source and target programs before, during or after the translation process.

Waters (1988) proposed an alternative approach to the language translation process to overcome problems associated with traditional source-to-source translators. Waters (1988, p. 1208) suggested that the process should begin with the

source program being analysed to “obtain a programming-language-independent abstract description” of the source application.

Echoing Waters’ (1988) suggestion, in a report on the evolution of legacy systems, Weiderman, Bergey, Smith, & Tilley, (1997, p. 25) offer the following summary recommendations:

- understand the legacy system at a high level of abstraction using some kind of system-understanding technology, paying particular attention to interfaces and abstractions; and
- find the encapsulate-able components of the legacy system on which to build.

Both points are directly applicable to this study in the way they relate to the use of the UML in reverse and forward engineering. Waters (1988) recognised the significance of abstracting both constructs and statements from within a source program during programming language translation. Other researchers, Kontogiannis et al. (1998); Skarmstad et al. (1999); Terekhov and Verhoef (2000); Weiderman et al. (1997), have noted the benefits of abstracting the OO component-like constructs within source applications for translation purposes.

The Object Management Group (OMG) has identified a need to standardise legacy transformation processes in order to “help build on prior experiences and best practices” OMG (2003, p. 2). The OMG anticipates that standardisation of legacy transformation processes will “enable integration and interoperability between solutions and vendor tools” OMG (2003, p. 2). The OMG-proposed standardisation includes the use of tools such as Metamodel Driven Architecture (MDA) and the UML. The platform independent MDA enables the creation of a UML model of a reverse engineered application “for the purpose of importing it into an MDA-enabled development environment” OMG (2003, p. 3).

Meta_Object Facility (MOF), also defined by the OMG (2002), is a specification used to describe an abstract language and a framework for specifying,

constructing and managing technology neutral metamodels (OMG, 2002, p. 15). The MOF, UML and eXtensible Mark-up Language (XML) Metadata Interchange (XMI) are intended to provide a foundation for the MDA. The OMG proposes the development of a standardised meta-language that may be used to describe UML models to provide a complete alignment of the UML and the MOF (OMG, 2002). The introduction of such a standardised language “would assist in the process of translating these models into software implementations” OMG (2002, p. 26). Potentially improving on the structure of a Rational Rose Enterprise Edition 2002 model file.

The Rational Rose Enterprise Edition 2002 development environment produces a proprietarily structured model file containing the properties associated with the current model. A framework “wizard” template is used to determine the structure of a Rational Rose Enterprise Edition 2002 model file. A framework in Rose/UML is a set of predefined model elements that are needed to model a certain kind of system (Rational, 2001). However, when developing a new framework a developer may associate additional descriptors with any or all of the properties in a model. This flexibility in the framework development process allows for the properties in a model to be described using different fields and values. For example, some of the extra properties e.g. Map File and subschema properties, associated with a JADE model may be seen in Figure 2.

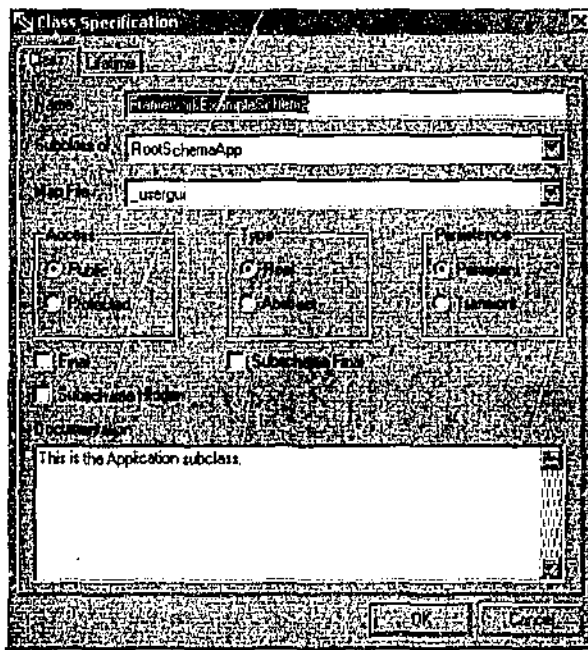


Figure 2: Framework properties associated with JADE models

The map file and subschema properties shown in Figure 2 represent a sample of the properties that may be considered unique in a JADE model, in similar manner to the peculiar model properties associated with 'unsigned short int' objects in a C/C++ model. Consequently, it was necessary to develop a grammar to validate any modifications made to an application's model files during the translation process.

3.3 Grammar development

A grammar is a description and depiction of the syntax of a programming language (Sebesta, 1999). It is beyond the scope of this document to detail the history of programming language generation mechanisms. However, a simple example may be useful to demonstrate the processes required to define and describe a small language. Figure 3 defines a grammar for the simple assignment statement:

$A := B * (A + C)$ (Sebesta, 1999, p. 113).

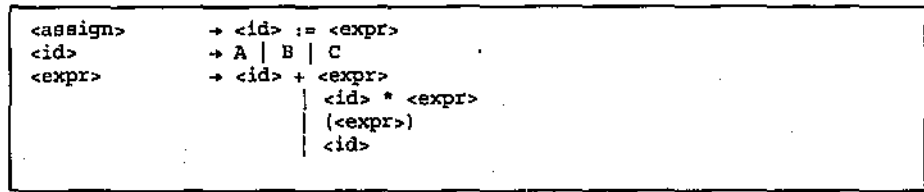


Figure 3: A simple assignment statement grammar (Sebesta, 1999, p. 113).

Analysis of the assignment statement may be performed in any of three manners: lineally, semantically or hierarchically (Aho, Sethi, & Ullman, 2003, p. 4). Initially, linear analysis reads the characters of an input stream from left to right. Then, semantic analysis ensures the sequence of characters or words forms a meaningful statement. Finally, hierarchical analysis groups the contents of an input stream into a set of hierarchically linked nodes representing the input stream as a parse tree (Aho et al. 2003, p. 4-5).

Aho et al. (2003, p. 6) describe the process of hierarchical analysis as 'parsing' the input. A grammar such as that shown in Figure 3 may be used to develop a parse tree representing the input that the grammar is to define (Sebesta, 1999). The parse tree shown in Figure 4 describes the assignment statement using the grammar shown in Figure 3.

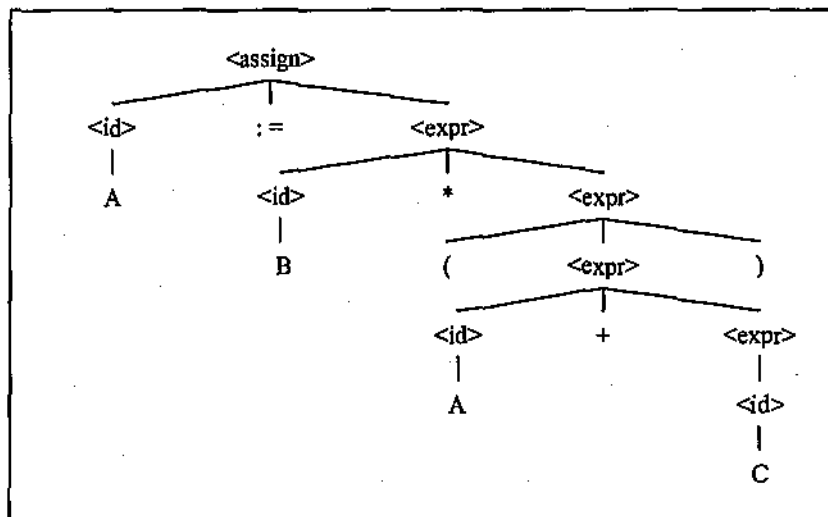


Figure 4: A sample parse tree, (Sebesta, 1999, p. 114)

The parser used in the investigation, ProGrammar (NorKen, 2003), enabled the converter application to extract nodes or entire lines of code from the parse tree. The parser applies a numbered index to each node in the parse tree and may return a line-number-id for the current line of code on which a specific node is found. Having both these resources available during the translation process allowed the converter to extract node values to test conditions on the values contained in the nodes of the parse tree or in a LOC of the source application. For example, the converter may request that only the children of a node with a certain value be returned. Alternatively, return an entire LOC if the value of the first node, in a sub-branch of the parse tree, matches a certain condition.

Such flexibility in the parsing tool provided the converter with enough processing power to concentrate specifically on the algorithmic code contained within each class method. Use of an existing tool with such flexibility was far more appealing than creating a parser / compiler tool using Lex and Yacc.

Lex and Yacc are tools that together, enable the developer to create programs capable of transforming structured input (Levine, Mason, & Brown, 1995). Lex is used to build a lexical analyser that takes streams of input and returns tokens representing the items in the input stream. Yacc builds parsers created from rules and grammars that describe the syntax of the input stream being analysed (Aho et al., 2003). The limited time available for this study, and the accessibility of a suitable parsing tool, were reasons for not employing Lex and Yacc.

ProGrammar is such a parsing tool and was employed during the investigation. It provides a visual environment for building parsers that are platform-independent, programming language-independent and reusable (NorKen, 2003). ProGrammar spared the researcher the burden of designing and developing the lexical analyser and parsing tools with the ability to work in three languages (JADE, C++ and Rose / UML), as well as a converter to use them. Figure 5 depicts the steps necessary to build and use a parser with the ProGrammar tool.

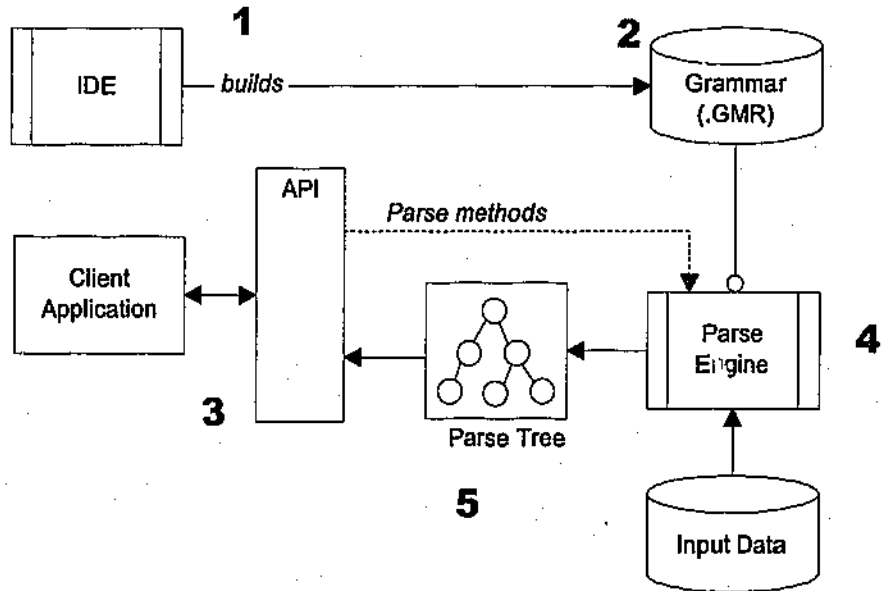


Figure 5: Building a parser with ProGrammar (NorKen, 2003, p. 14).

Each of the numbered stages shown in Figure 5 is outlined below.

1. Define the grammar for the input to be parsed in the IDE;
2. ProGrammar then generates a binary grammar file;
3. The parser is called from the client application via an API;
4. The runtime parse engine creates the parse tree representing the source application as input data; and
5. The client application may then retrieve the data from the parse tree via an API (NorKen, 2003, p. 15).

According to Aho et al. (2003, p. 1) parsing input streams is the basis for compiling computer programs. In most situations the direction of language-generation or compilation, by a compiler, is from a high-level programming language to a low-level 'machine code' language that the computer may understand. However, some language compilers, for example: Safe C, Eiffel and Cfront, work between high-level languages. Safe C was developed by Michael Collins (1993) as a high-level compiler used for translating an 'ADA-Like' language to Safe C, which he developed as a cheaper alternative for use in embedded systems. Eiffel, developed

by Bertrand Meyer “has all the typical features of a high-level language” Gutschmidt (2003) and translates it to C. **Cfront** is described by Wikipedia (2003) as “the original compiler for C++, which converted C++ to C”.

Harsu (2000, p. 6) uses different terms to describe the concepts of programming language transformation. Figure 6 shows that, according to Harsu (2000), compilation generally works on high-level languages being transformed into low-level languages, while the interchangeable terms, ‘conversion’ and ‘translation’, describe language transformations at the same level.

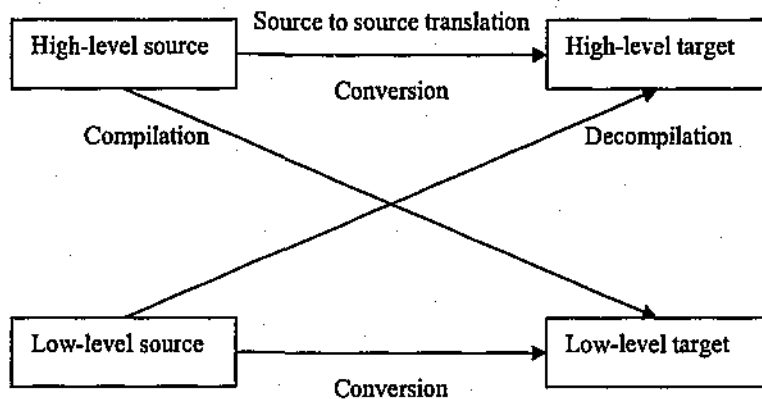


Figure 6: Terms used in program transformation (Harsu, 2000, p. 6).

3.4 Application Selection

The applications selected for translation during this investigation are widely available classical programs. The first deals with the Towers of Hanoi problem (Hill, 1995; Roeder, 2003; Suh & Allain, 2003), while the second describes inheritance in an object-oriented environment (Liberty, 2001; Schildt, 2003). The implementation of the Towers of Hanoi application used in this investigation was selected from many available on the Internet.

The Towers of Hanoi problem, the character of which is depicted in Figure 7, requires a solution that moves all four rings, one at a time, from one tower to another, without allowing any ring to be placed on top of a smaller ring. The Towers of

Hanoi application used in this investigation was developed by Chris Roeder (2003) and details of the source code are included in Appendix A.

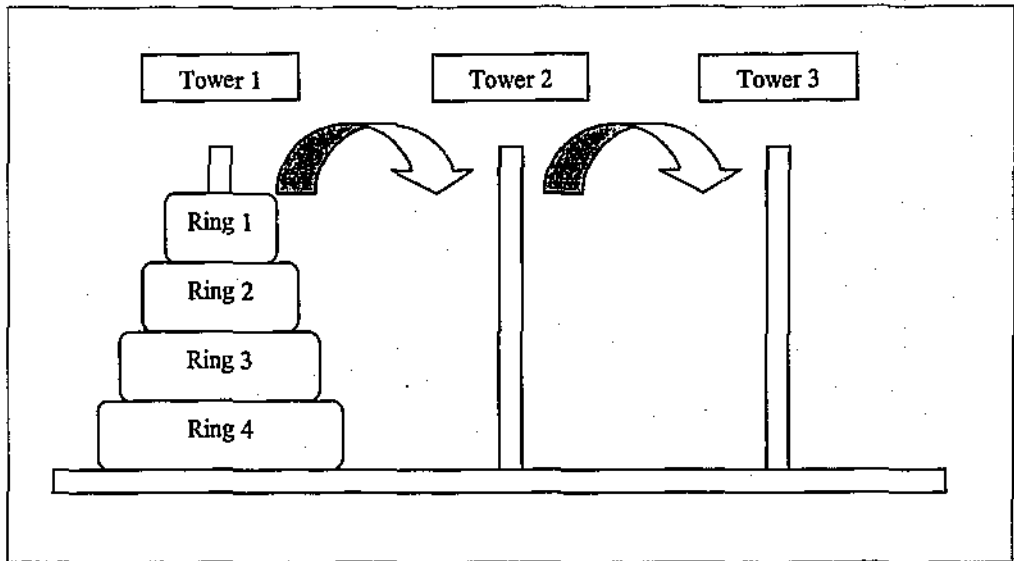


Figure 7: The Towers of Hanoi problem.

Programmatically, a solution to the Towers of Hanoi problem usually employs recursion to move the rings within the rules. While recursion does not make the program complex, it adds a degree of complexity to the demonstration of this investigation's concept. Without the recursion factor included in the application, the numbers of independent paths or conditions tested during the application at run-time are few. Sultanoglu (1998) suggests that McCabe's Cyclomatic Complexity (MCC) "measures the number of independent paths in a program, thereby placing a numerical value on the complexity" of the application module. The formula for the MCC metric used to measure the complexity of the Towers of Hanoi sample application is:

$$\text{MCC} = \text{edges} - \text{nodes} + 2;$$

where the nodes "represent computational statements or expressions, and the edges represent transfer of control between nodes" (Watson & McCabe, 1996). The MCC was used during this investigation to provide a measure of the complexity of the sample application's decision structure. The number of nodes in the Towers of Hanoi application amounted to 46 while the number of edges totalled 47 yielding:

$$\text{MCC} = 47 - 46 + 2 \Rightarrow 3$$

Hence the Towers of Hanoi represents an MCC of 3. The MCC generally maintains a maximum limit of 10 for extremely complex application modules as recommended by Watson and McCabe (1996). An earlier study by McCabe and Butler (1989, p. 1416) reported that the modules of the evidently non-trivial AEGIS Naval Weapons System approximated 4.6 MCC. Tieman (2001) suggests that where a MCC result lies between 6 and 10 a developer should consider ways of simplifying a module. Consequently, it was considered by the author that an MCC of 3 represented a module of reasonable complexity for the purpose of “proof of concept” for the study in both the static structure abstraction and the transliteration processes.

The second application converted during this investigation, Schildt’s (2003, p. 280) building inheritance example shown in Figure 1, was measured using a different set of metrics. The building inheritance application is highly OO in nature and the MCC was unable to reflect its overall complexity. Accordingly, a suite of metrics based on measurement theory developed with the insights of experienced OO software developers, presented by Chidamber and Kemerer (1991, p. 197) was applied. The tools presented within the Chidamber and Kemerer (1991) Metrics Suite (CKMS) include the:

- Weighted Methods per Class (WMC);
- Depth of Inheritance Tree (DIT);
- Number of Children (NOC);
- Coupling Between Objects (CBO);
- Response for a Class (RFC); and
- Lack of Cohesion in Methods (LCM).

Each of these tools is described briefly below.

WMC is a measure of the number of methods in a class. Chidamber and Kemerer (1991, p. 202) state that “the number of methods and the complexity of the

methods involved is an indicator of how much time and effort is required to develop and maintain the object". When the number of methods in a parent class increases, the overall number of methods available to the combined inherited classes in a module also expands, thereby increasing the complexity of the application (Chidamber & Kemerer, 1991).

In describing DIT as an appropriate metric for OO software application measurement, Verbruggen (2003) cites Chidamber and Kemerer (1991) quoting "the deeper a class is in the hierarchy, the greater the number of methods it is likely to inherit, making it more complex." Deeper inheritance trees "constitute greater design complexity, since more classes and methods are involved" (Chidamber & Kemerer, 1991, p. 202).

Verbruggen (2003) alludes to the NOC metric as indicating both good and bad properties in a class. Notably, higher NOC may indicate either "greater re-use, since inheritance promotes re-use" or "improper abstraction of the parent class", (Verbruggen, 2003). Notwithstanding, an increase in the NOC equates to an increase in a module's complexity.

CBO is a measure of "the degree of interdependence between modules" (Chidamber & Kemerer, 1991, p. 203). The less dependent an object is upon other modules, the better equipped it is for re-use. Simple connectivity, or low coupling, between modules produces applications which are "easier to understand" and "less prone to the ripple effect" (Pressman, 2001, p. 354). The ripple effect is aptly described by Pressman (2001, p. 354) as being "caused when errors occur at one location and propagate through the system", making error detection and location more difficult.

RFC is an indication of the number of methods that are visible publicly to objects communicating with the specific module. "The larger the number of methods

that may be invoked from a class, the greater the complexity of that class” (Verbruggen, 2003).

LCM is a “measure of the attributes of an object” (Chidamber & Kemerer, 1991, p. 204) and provides an indication of the level of cohesion or encapsulation of an object. “Low cohesion increases complexity” potentially leading to an increase in the number of errors during the development process (Chidamber & Kemerer, 1991, p. 204).

The following table summarises the building inheritance application’s complexity using the CKMS. An average of the values for each metric associated with the classes in the source application is calculated and presented in the right column of Table 1.

Table 1: CKMS metric evaluation of building inheritance.

	Class Building	Class House	Class School	Average
WMC	6	4	4	4.66
DIT	1	2	2	1.66
NOC	2	0	0	0.66
CBO	0	6	6	4.00
RFC	6	10	10	8.66
LCM	3	2	2	2.33
Class CKMS	3.00	4.00	4.00	
	Total			21.97
	Total number of classes = Application CKMS			7.32

The application CKMS is the result of dividing the Total by the number of classes in the application. Verbruggen (2003) suggests that a class CKMS level of 4 to 5 is considered “very good”. Unfortunately, a typical overall application CKMS level for use as a comparison has not been located in the literature reviewed by the author.

3.5 Chapter Summary

Previous studies have been reviewed to highlight the difficulties associated with the translation of programming languages using traditional source-to-source translation methods. It was suggested that no significant improvement in translation system achievements had been realised between the time Waters (1988) presented the abstraction and reimplementing idea, and those recent projects still using transliteration, e.g. Harsu (2000). The goals of legacy-system translation projects were discussed along with the prerequisite strategies to be considered prior to the commencement of such projects.

The UML was presented during this chapter as a method of describing the static structure of a legacy system, as suggested by Waters (1988) and Weiderman et al (1997). Furthermore, ROSE/UML was offered as a CASE tool capable of reverse engineering and then presentation of the static structure of a source application. Programming language grammars were described before the methods of calculating the complexity of the selected applications were discussed. The studies reviewed in this chapter were provided for justification for this study's purpose and approach.

4 Research Design

4.1 General Method

The Research Design is presented in three phases, each comprising multiple steps.

Phase	Description
Phase 1 – <i>The Static Structure</i>	<ul style="list-style-type: none">a) selection of source application(s) for translation;b) reverse engineering of each source application; followed by thec) manipulation of the model properties to produce a valid target language version of the model; and finally thed) exportation of the target language schema file.
Phase 2 – <i>The Algorithmic Code</i>	<ul style="list-style-type: none">e) development of the grammars describing each of the source and target languages used during the investigation;f) generation of the application parse trees;g) extension of the target language [i.e. JADE] schema file, with the details of the static structure produced during phase 1; and finallyh) translation and insertion of the algorithmic code in the equivalent target methods of the target schema file.
Phase 3 – <i>The Analysis and Findings</i>	<ul style="list-style-type: none">i) collection and correlation of the data resulting from the translation of the sample application(s); and thej) conclusion of the investigation by answering the research questions with the findings of the data analysis.

Figure 8 describes a high-level view of phases 1 and 2 at the right and left of the diagram respectively. The details of these phases are described in section 4.2.

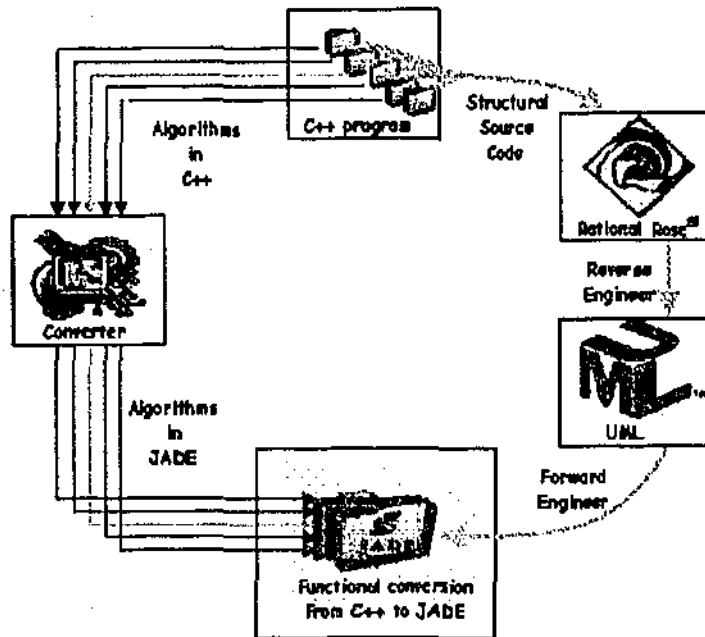


Figure 8: A high-level view of the process for the study

4.2 Specific Procedures

The steps of the phases introduced in 4.1 are detailed in this section and associations that each may have with the research questions posed in 2.4 are clarified.

4.2.1 Phase 1 – The Static Structure

4.2.1.1 Selection of the source applications

The applications to be translated during this investigation were selected for their availability in various forms; because they embody challenging concepts in the field of programming; and because each offers reasonable complexity. These applications were also selected for their object-oriented implementations which are recognised by both the source and target languages and, importantly, to demonstrate that the applications were not purpose built for the study.

4.2.1.2 Reverse engineering

To provide an answer to the first sub-question in section 2.4.1, the investigation needed to compare the properties of a reverse engineered model file to the properties in an equivalent model file associated with the target language. To achieve a comparison the source Microsoft Visual C++ (VC++) application was reverse engineered, using ROSE/UML, producing a static structure model file. The author then created a second static structure model of the same application using Rose/UML's development environment, instead associating the second model with the target language, in this case JADE.

The comparison of the properties in the two model files revealed the property names and their values where each model is associated with the different programming languages. This comparison process also allowed the author to recognise the options, available in the Rose/UML development environment, where the property values may be manipulated to reflect the programming language associated with the model. Data collected during this step in Phase 1 provided the information required to answer sub-question 1 of section 2.4.1, which is repeated here for convenience:

Sub-question 1: *Which model properties within a Rational Rose model file are associated with the reverse engineered application's programming language?*

4.2.1.3 Model manipulation

Changing the reverse engineered model options in the Rose/UML development environment enabled the author to alter the model's association with the original source application's programming language. The author then imported the target language's root-schema, or base classes, allowing the model to be associated with JADE. Each of the elements in the model was then manipulated to reflect the equivalent element type in the target language. The elements being manipulated involved attribute types along with the names of some of the elements in the original application. Following the completion of the model element manipulation, the modified model was ready for export to JADE.

4.2.1.4 Export to the target language

Initiation of the export process from within the Rose/UML development environment produced a JADE schema file representing the basic static structure of the original source application in the target language. Completion of this step in phase 1 allowed the collection of data and the inspection of the exported schema file to determine an answer to the second sub-question in 2.4.1 repeated here for convenience:

Sub-question 2: Which components of the JADE schema file, produced by the RoseJADELink add-in, may be used to construct the static framework in preparation for code migration?

4.2.1.5 Phase completion

Phase 1 took a complete and working version of a VC++ application and, using Rose/UML, produced a UML model representing the static structure of that application. The options within the development environment were then altered to remove the model's associated programming language. The target language base classes were then imported and the model's options associated with JADE. The attributes and operations contained in the model were then manipulated to reflect the target language equivalent attribute types and names. The completed model was then exported producing a JADE static structure schema file in readiness for extension and population with the translated algorithmic code.

4.2.2 Phase 2 – The Algorithmic Code.

4.2.2.1 Grammar development

In translating the algorithmic content of the source application into the target language, each word or token used in the source application was scanned and inserted into a parse tree. In order to produce a parse tree, the structure of the language must be known and syntactically correct. Consequently, a grammar was required for each application source-file used by the converter application to enable it to recognise the components of each line of code in the source file.

4.2.2.2 Parse tree generation

Using the grammars developed according to activities in section 4.2.2.1 and a parser application developed outside this investigation, by Norken Technologies, parse trees were created from each of the source files associated with the translation investigation. The parser queries the parse trees to locate nodes representing the equivalent element in the source file. The parse trees enabled the parser to return the value stored at each of the parse tree nodes, when and as it was requested by the converter application, during the translation process.

4.2.2.3 Schema file extension

The JADE schema file, exported from Rose/UML, does not contain all the section headings required by the JADE environment, for example, the `schemaViewDefinitions`, `_remapTableDefinitions`, `externalFunctionSources` and `typeSources` headings. Consequently, before adding any operational code to the JADE schema file, the missing headings were appended to the end of the existing content. Next, the classes and their methods, and the application schema methods were appended to the JADE schema file. With each of the application and class methods extracted from the parse tree, the algorithmic code for each was translated and inserted during the appending process.

4.2.2.4 Translation of algorithmic code

As each algorithmic LOC in the source application parse tree was queried, the parser returned the type of LOC being queried. The grammar categorised each algorithmic LOC with a specific name, for example, the parser would return “for_statement” when a ‘for loop’ was encountered and “if_statement” when an ‘if’ statement was encountered. The attributes and values making up the conditions or expressions used in each case were then supplied as parameters to a translating method which returned the formatted equivalent statement as a string which, in turn, was then appended to the appropriate position in the target schema file.

4.2.2.5 Phase completion

Phase 2 involved the development of the tools needed by the translation application to produce the translated algorithmic code for insertion into the target schema file. The tools included grammars for each of the programming languages and another grammar used to validate Rose/UML model files. Other items used during the translation were the parse trees and the parser that queried the contents, then returning the values contained in the parse tree nodes. The converter application used these tools to append the translated algorithmic code to the appropriate position in the target JADE schema file. Data collected during this phase enabled the provision of answers to the third sub-question in 2.4.1 and to the main research question in 2.4, both of which are repeated here for convenience:

Sub-question 3: What improvement in the ratio of automatically to manually translated LOC in a legacy system may be achieved using the abstraction and re-implementation approach?

Main question: Where separation of static and algorithmic components of code for forward engineering of a legacy system is achieved, then may a reduction of manual intervention be realised in automated code conversion?

4.2.3 Phase 3 – The Analysis and Findings.

4.2.3.1 Data collection and analysis

The JADE schema files, produced by the abstraction and transliteration process, were imported into the JADE development environment for testing. The testing performed on the translated schema files included the importation process itself. A schema fault report is produced where a schema does not conform to the rules associated with the JADE language.

The testing during this step also included invoking the translated applications in the JADE environment and then recording any changes required to enable the translated application to operate entirely as it did in the original language

environment. The following variables were found to have an influence on the study, each being identified in Table 2:

Table 2: The conversion data for analysis

Item	Description
Original_Loc	The number of LOC in the original application
Converted_Loc	The number of LOC in the converted version of the original application
Manual_Loc	The number of LOC requiring manual intervention, either before or after the translation process, to produce a successful translation
Automatic_Loc	$\text{Original_Loc} - \text{Manual_Loc}$
Time_Automatic_Loc	The time taken to translate Automatic_Loc
Time_Manual_Loc	The time required to translate Manual_Loc manually
Conversion_Time	$\text{Time_Automatic_Loc} + \text{Time_Manual_Loc}$
Environment	Details of the computer performing both the conversion and the compilation, for example: <ul style="list-style-type: none"> • the platform; • available memory; and • processor speed.

Analysis of the data relating to the variables listed in Table 2 enabled the comparison of equivalent data from both of the application conversions during this investigation. To determine whether an improvement in the process had been achieved, the percentage of Automatic_Loc derived from the translation of the Original_Loc was compared with the previous research results reported by Moynihan & Wallis (1991), Harsu (2000) and Terekhov (2001).

4.2.3.2 Findings and conclusions

Once the testing and analysis steps were concluded, the findings were then developed and associated with the research questions to evaluate the investigation. After the data analysis, conclusions were made regarding the abstraction and transliteration process and whether further investigation was warranted.

During the investigation some processes may have been improved had certain enhancements been incorporated into this study. However project constraints, chiefly those of time, prevented their inclusion. Those enhancements not included will now be explained.

4.3 Potential Enhancements not Incorporated in this Study

As some of the enhancements recognised during this investigation were outside the scope of this project they were not included. However, in the event that further investigation in the field may be considered, these enhancements are mentioned. The enhancements omitted and the reasons for their non-inclusion are discussed below.

- Automation of the Rose/UML model conversion process, using Rose's internal scripting language to provide the GUI and triggers for the translation process.
 - Although Rose/UML includes a scripting language, the time required to reveal the processes necessary to make the conversion was estimated to be more than that available to warrant its inclusion.
- Model alterations to remove the external function section being included in the reverse engineering process.
 - Further investigation of the options available within the Rose/UML development environment may reveal alternative methods of implementing the changes necessary to remove the external function association with each of the class methods during the conversion.
- Inclusion of the entire set of C++ statements and expressions in the translation process.
 - The complexity of the C++ language along with the ability to instantiate objects within expressions makes the mapping of statements from C++ to any other language extremely time consuming.
- GUI front end;
 - The creation of a Graphical User Interface (GUI) for the converter was considered to have aesthetic appeal only.

Currently, the converter application presents text based messages to the user within a console window during the conversion process.

4.4 Chapter Summary

The three phases of the project were described. Each of the three phases was presented as a series of sub-tasks that were followed to address relevant components of the research questions posed in section 2.4. The initial phase addressed the development and realisation of the static structure of the original applications being translated. In describing the second phase, the processes of translation of the algorithmic code and target schema method population were outlined. The final phase outlined the testing of the translated applications and analysis of the data that would be generated from those tests. In addition, potential enhancements that were not addressed in the study were identified.

5 Implementation and Findings

In chapter 4, the three phases of the study were introduced together with their subtasks and the relationship each may have to the research questions. This chapter relates the phases and the subtasks introduced in chapter 4 in terms of how the investigation's goals were implemented and the findings that were realised.

5.1 Phase 1: The Static Structure

5.1.1 Selection of the source applications

The investigation commenced with a comparison of two Rose/UML model files. The model files used were a reverse engineered VC++ sample-application model file and a purpose built JADE model file representing the same application functionality. The applications used during this procedure are described in section 3.2. Each of the selected applications represents a readily available classical program. The implementations in C++ were not custom built for this study and may be considered typical of programs of this type and complexity that may be translated in a "real world" situation.

5.1.2 Reverse engineering

The reverse engineering process performed using Rose/UML produces a model containing source code components and a class diagram representing the static structure of the source application. Each of the source code components represents a source code file included in the original application (Quatrani, 2000). The highlighted "Main" component of the Towers of Hanoi application may be seen in the left window of Figure 9.

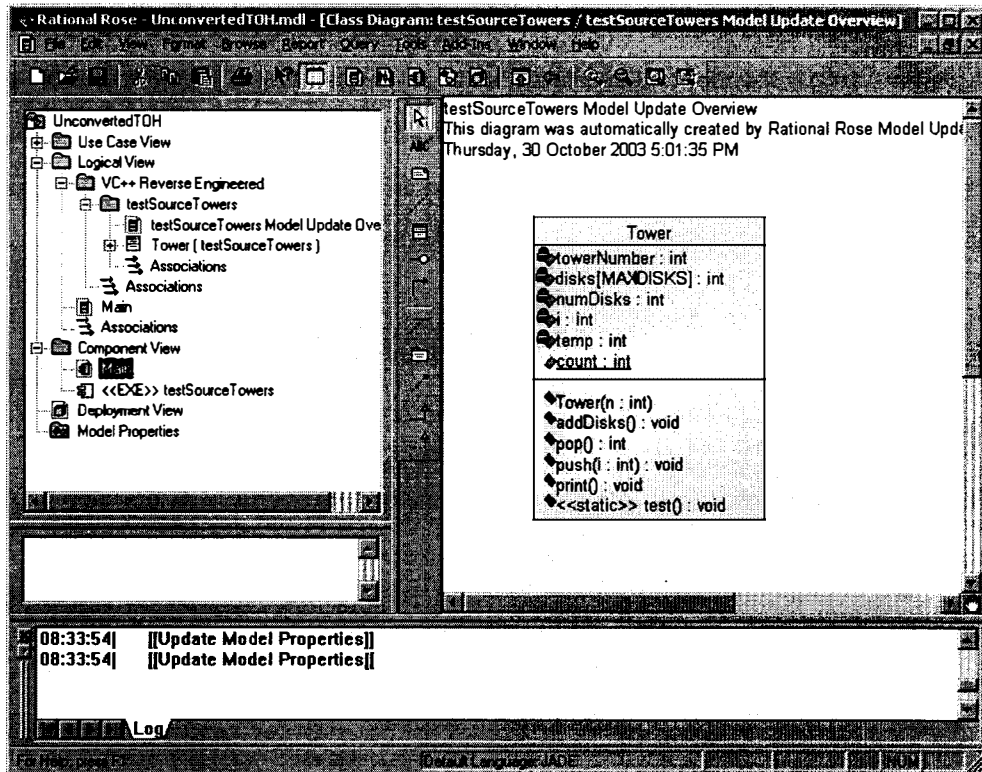


Figure 9: Reverse engineered VC++ application.

Both the model that Rose/UML creates during a reverse engineering process and/or a model created by a developer generate Rose/UML model file(s). These contain a hierarchy of nodes and values representing the properties associated with a model displayed in the Rose/UML development environment.

In section 4.2.1.2 it was stated that to arrive at an answer to the first research sub-question, it was necessary to compare the contents in a reverse engineered model file with the contents in a model file specifically built with an association to JADE, the translation's target language. A comparison of the model file contents is necessary to determine those reverse engineered model file properties associated with the source application's programming language.

The comparison made between the two model files yielded some significant discoveries. For example, to provide a definition of the target language model, the purpose built JADE model file used more than twice the number of LOC than the number required to describe the VC++ version of the same model. The Rose/UML

model file representing the VC++ application contained 4,815 LOC, with 18,644 property nodes defining the model. The equivalent JADE model file required 85,257 LOC and 383,177 property nodes to define the equivalent model associated with the target language. The reasons for this apparent block are now explored.

A comparison of the nodes in the model files confirmed that the majority of the extra data was related to the JADE root-schema. This is essential to the application and is generated as a matter of course for all JADE applications. The JADE root-schema is similar in purpose to Microsoft's Foundation Classes (MFC). Both architectures, i.e. the JADE root-schema and the MFC, are libraries of object-oriented classes structured into their respective hierarchies. A small example of the JADE root-schema may be seen in Figure 10. The libraries included in both JADE and the MFC allow developers to include a wide range of visual components in an application (White, Scribner, & Olafsen, 1999). The JADE root-schema also includes the native types required by the language.

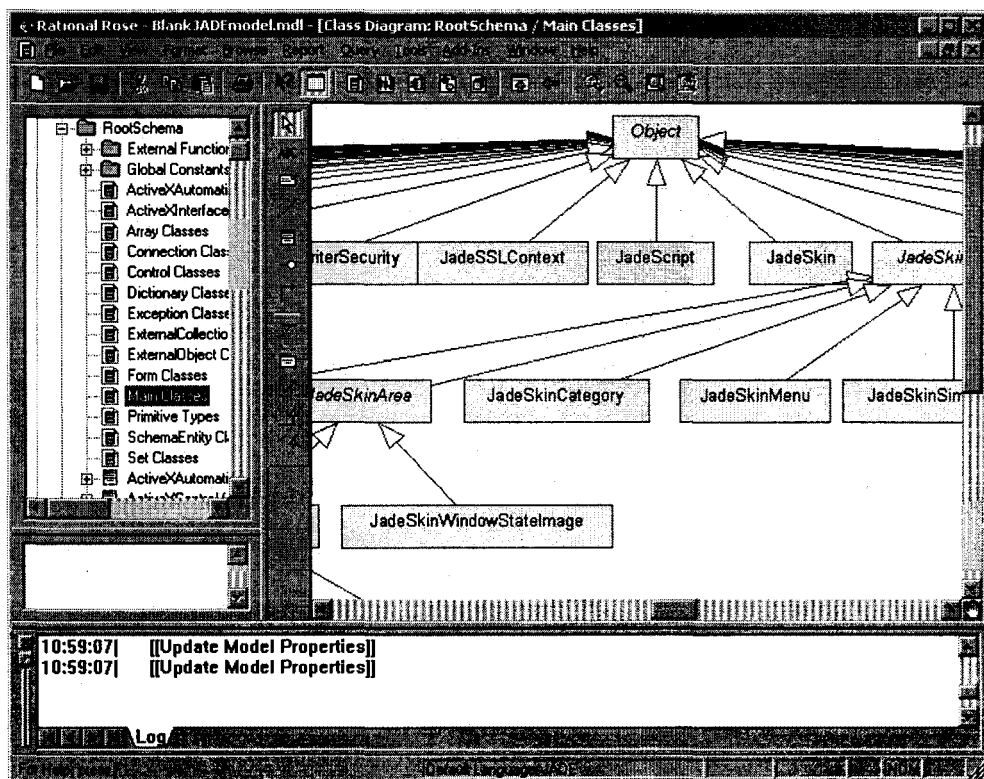


Figure 10: The JADE root-schema class diagram.

Although the MFC was not included in the original reverse engineering process, when the MFC was imported into the schema the difference in file contents was still significant. With 30,824 LOC and 127,128 nodes, including the MFC, the source application file still proved significantly smaller than the equivalent JADE target model.

It is these major size and syntactic differences in the model files, representing the same application, which led to the development of a third grammar during this investigation. The Rose/UML model file grammar was developed to provide the parser with the rules used by Rose/UML to check a model file for syntactic correctness after the manipulation of a model's properties.

5.1.3 Model manipulation

The RoseJADELINK add-in used to export a model to JADE requires more properties and associated values to define a model's objects than the process used to export a VC++ model. Some of the properties required by the RoseJADELINK add-in are unique to JADE models. This difference in properties and values is the result of different development teams being responsible for building the add-ins used by each of the programming languages recognised by Rose/UML.

For the RoseJADELINK add-in to produce a useable JADE schema file during the export process, certain properties must be present in the model file being exported to JADE. Unless the properties defining each object in the model are correct, the export process either fails or produces a faulty schema. A brief example of the differences in the sample application's class attributes may be seen in the following code examples in Figure 11 and Figure 12. The code examples are taken from the original reverse engineered VC++ model file and from the equivalent JADE model file.

```

class_attributes (list class_attribute_list
  (object ClassAttribute "towerNumber"
    quid "3F94B4300253"
    type "int")
  (object ClassAttribute "disks[MAXDISKS]"
    quid "3F94B4300261"
    type "int")
  (object ClassAttribute "numDisks"
    quid "3F94B4300262"
    type "int")
  (object ClassAttribute "i"
    quid "3F94B4300271"
    type "int")
  (object ClassAttribute "temp"
    quid "3F94B4300272"
    type "int"))

```

Figure 11: VC++ model class attribute properties.

Notice the ClassAttribute object property referring to the disks[MAXDISKS] item on the fifth line in Figure 11. The MAXDISKS component is not defined any further than this in the VC++ model file, whereas in Figure 12, the JADE model file devotes 13 LOC to define the MAXDISKS object.

```

class_attributes (list class_attribute_list
  (object ClassAttribute "towerNumber"
    quid "3F93CD380344"
    type "Integer"
    quidu "3F93D1CD0083")
  (object ClassAttribute "disks[MAXDISKS]"
    quid "3F93CD3803B2"
    type "IntegerArray"
    quidu "3F93D04B00D1"
    exportControl "Protected")
  (object ClassAttribute "numDisks"
    quid "3F93CD3803BC"
    type "Integer"
    quidu "3F93D1CD0083")
  (object ClassAttribute "i"
    quid "3F93CD3803D0"
    type "Integer"
    quidu "3F93D1CD0083")
  (object ClassAttribute "temp"
    quid "3F93CD3803DA"
    type "Integer"
    quidu "3F93D1CD0083")
  (object ClassAttribute "MAXDISKS"
    attributes (list Attribute_Set
      (object Attribute
        tool "JADE"
        name "Read Only"
        value TRUE))
      quid "3F93DDDB0312"
      stereotype "const"
      type "Integer"
      quidu "3F93D1CD0083"
      initv "4"
      exportControl "Protected"
      Containment "By Value"))

```

Figure 12: JADE model class attribute properties.

Examination of the model files provided helpful insights into the object properties requiring alteration and where those object specification options were to be found in the Rose/UML class-modelling environment.

To begin the conversion process, the Towers of Hanoi sample application, described in section 3.2, was coded and compiled in VC++. The resulting application runs in a console window as shown in Figure 13 below.

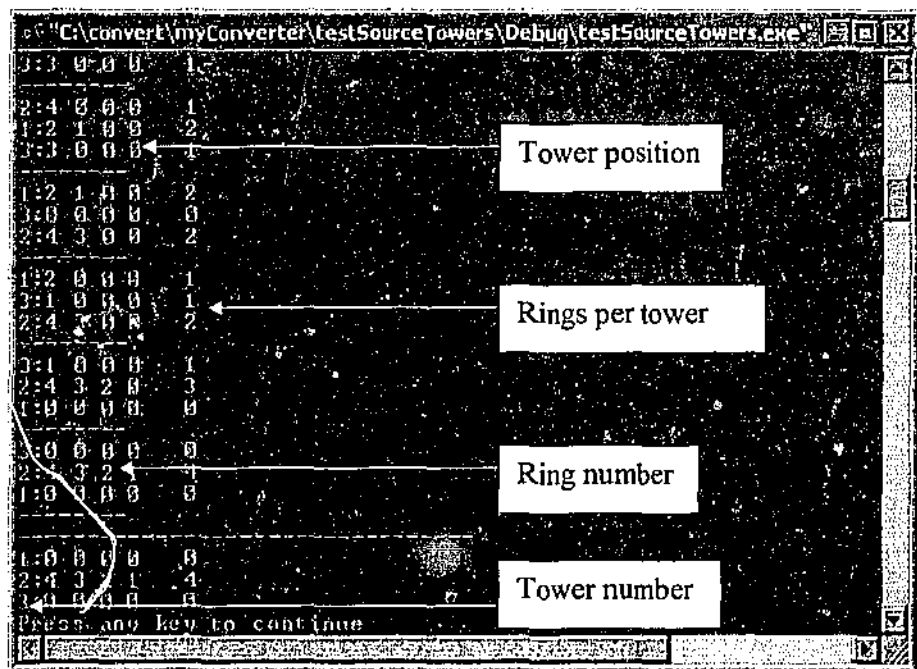


Figure 13: The Towers of Hanoi program at run-time.

The Towers of Hanoi application was then reverse engineered using Rose/UML, which produced a UML model represented in a class diagram shown in Figure 9. The Towers of Hanoi application contains one class, making it a simple example of a UML class diagram. Consequently, as a more complex UML conversion process, the investigation was also occupied with the language migration for a second application, based on an example of inheritance from a text by Schildt (2003, p. 280), also described in detail in section 3.2.

The class diagram rendered from the reverse engineering process of the second application, Schildt's "building inheritance example" (2003, p. 280), is shown in Figure 14 below.

testSourceInheritance Model Update Overview
 This diagram was automatically created by Rational Rose Model Update Tool.
 Friday, 24 October 2003 12:47:12 PM

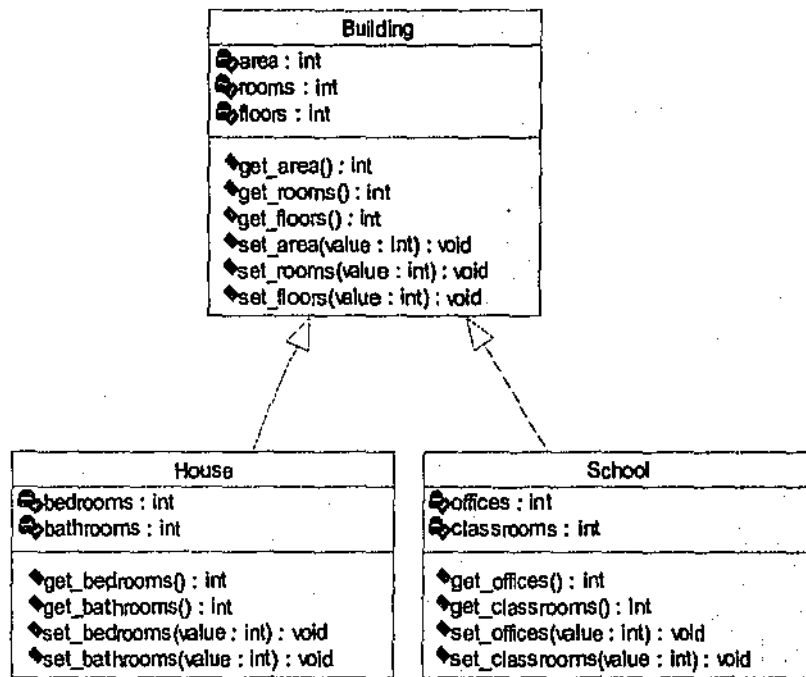


Figure 14: Inheritance sample application class diagram.

The next stage of the UML model language migration of the Towers of Hanoi application, to the target language, was to remove the association that objects in the class diagram have with VC++. This was achieved by reversing the processes described by Quatrani (2000, p. 211) for assigning a language to Rose/UML model components. The selection of the target language, shown in Figure 15 below, associates the overall model with the language option selected in the Rose Component Specification dialog.

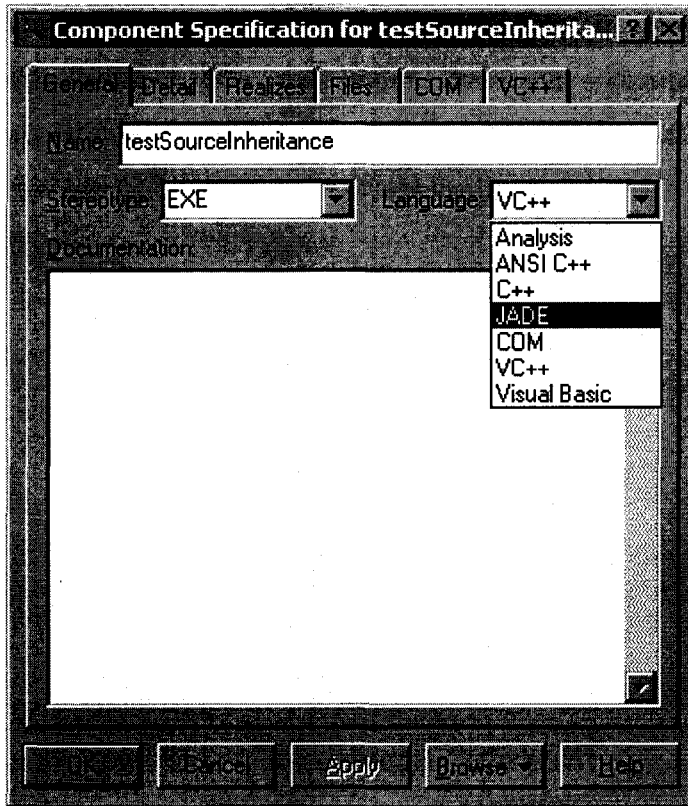


Figure 15: Re-assigning the application to the target language.

Once the components of the source application were associated with the target language, the JADE root-schema was imported into the model; this is initiated by selecting the option from the Tools → JADE menu, which opens the JADE import dialog shown in Figure 16.

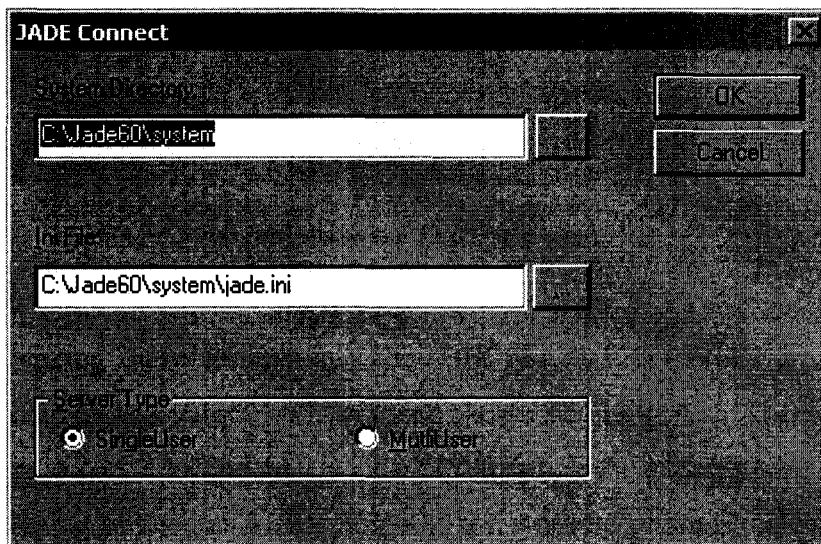


Figure 16: The JADE connection dialog.

On completion of the importation process, a new application schema was added to the model. With this addition, the model contains all the necessary classes and components needed to generate the shell of a JADE schema file. An example of the Towers of Hanoi schema file is attached at Appendix B. Before commencement of the schema generation process, the individual class components must be assigned to the newly created application schema as shown in Figure 17. The source application's original 'base class' must then be allocated to a new parent class which, in this case, is JADE's fundamental base class of 'object'.

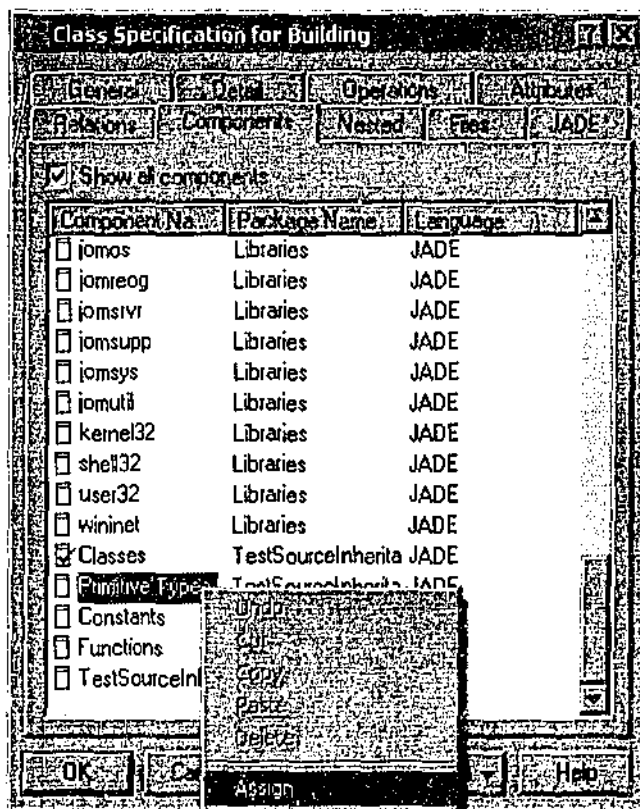


Figure 17: Assigning the class objects to the JADE schema.

Making these assignments alters the model file, thereby creating the extra property fields and values such as those shown in Figure 12. These properties are necessary to create a valid JADE schema file during the export process introduced at the beginning of section 5.1.3. Some of the attribute property values in the model file require changing to allow the correct assignment to JADE types. Each class in

the model is then associated with a map file that contains the details of each item in the model, via the Class Specification dialog in Figure 18.

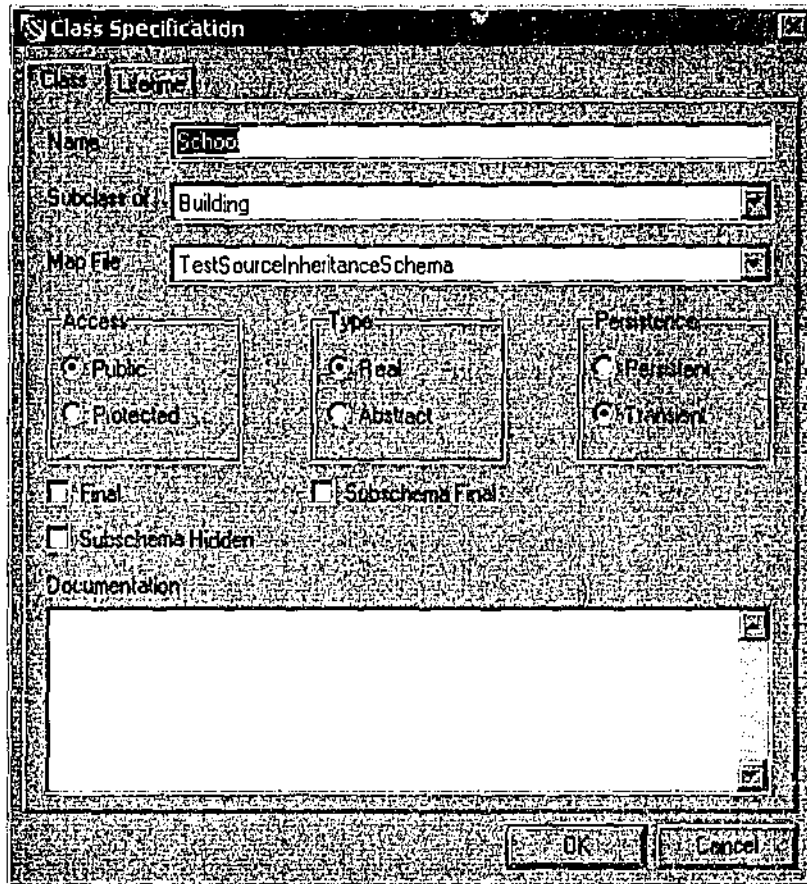


Figure 18: Class relations, the parent class and the map file.

Each of the attributes in the model was altered to reflect the equivalent target language type. For example, objects of type 'int' used in the VC++ application had to be changed to 'Integer' for JADE to recognise them. Another necessary alteration was the removal of the C++ keyword 'void' from any class methods not returning a value. There were multiple techniques available for performing such alterations, for example, a global search and replace provided by some text editors, although the process lends itself readily to automation. Though rudimentary, this method was tested during the investigation and was found to be successful and significantly quicker than using the specification dialog windows in Rose/UML shown in Figure 19. These dialog windows provide accuracy for the process, as the developer may introduce spelling errors during the process. However, if the source application

contains a large number of classes, the time required to make manual alterations would prove costly.

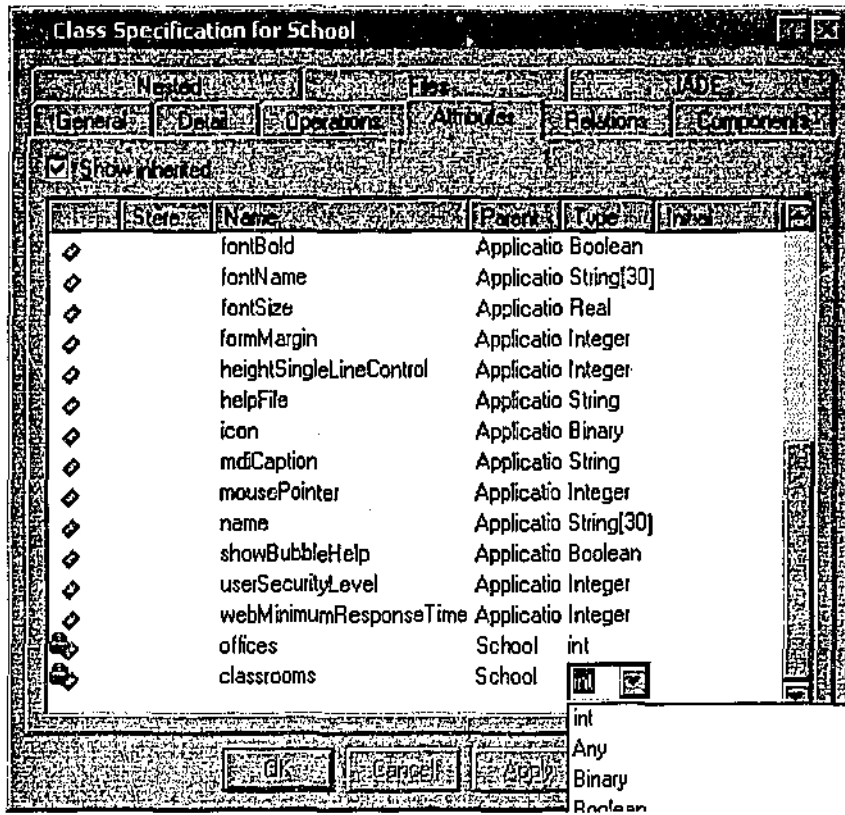


Figure 19: Re-assigning attribute types.

Upon completion of the model manipulation described in this section and in section 4.2.1.3, the model was ready for the final step in the first phase, to be exported to JADE.

5.1.4 Export to the target language

From the previous steps, the model included all the necessary properties and components required by the RoseJADELinK add-in to produce a syntactically correct JADE schema file. The model, shown in Figure 20, was ready for the export process to begin.

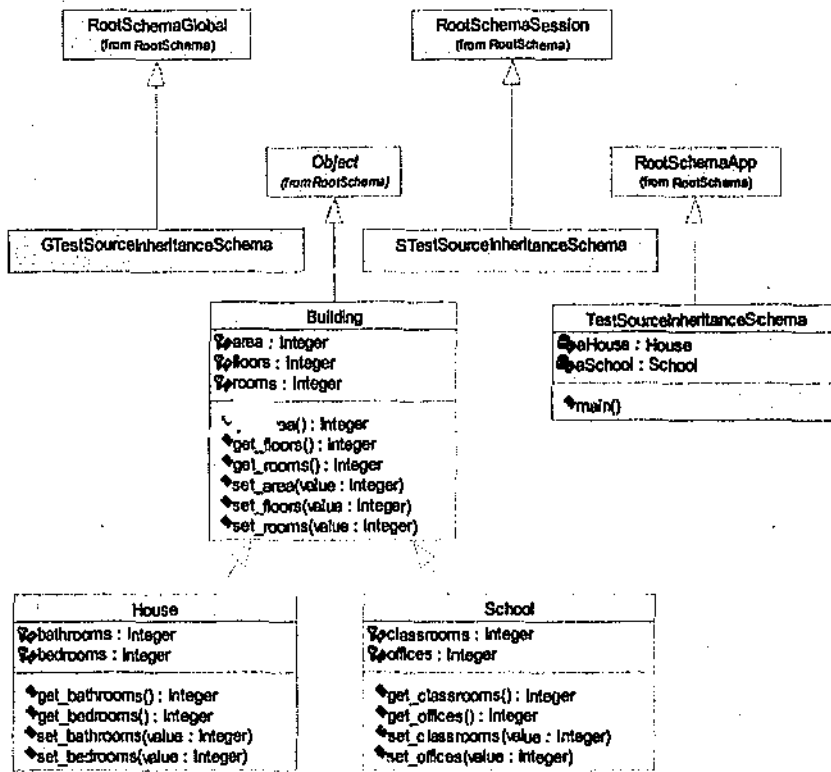


Figure 20: The manipulated inheritance model

Initiation of the export process is by selecting the Tools → JADE → Export to JADE... menu option. Selection of this option presents the developer with the JADE connection dialog, shown as Figure 16, providing the option of naming the output schema file. The final selection required before the export process begins is that of the schema to export. Once selected, the export process begins and a target schema file is generated. Finally, during the export process, the developer is presented with a report dialog, shown as Figure 21, which displays the progress of the export process through to its completion.

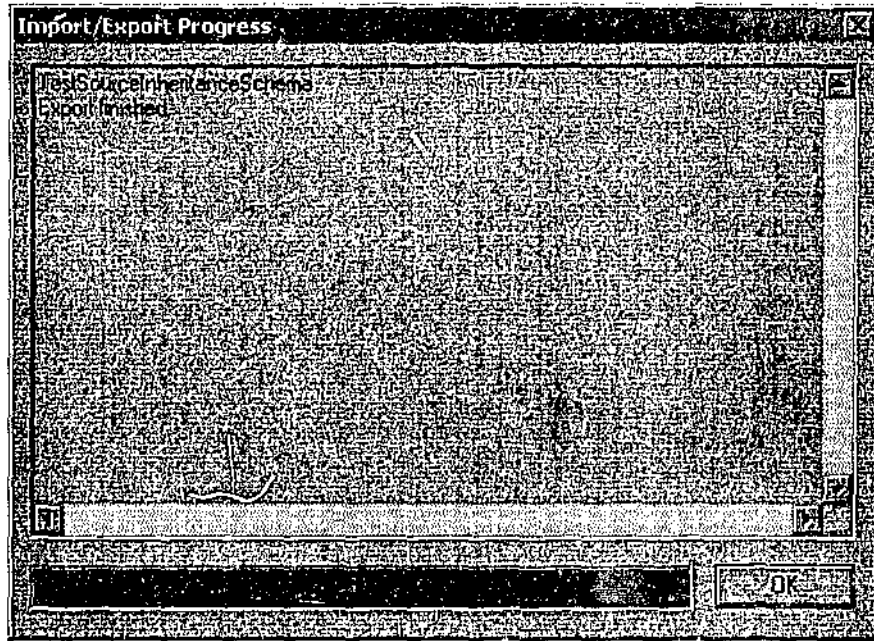


Figure 21: Import/Export progress report dialog.

To test whether the export process had succeeded, the schema was imported into the JADE development environment, which tests a schema file for errors both syntactically and semantically. This test showed a flaw in the conversion process, where the class methods were erroneously declared as external functions under the `externalFunctionDefinitions` section of the schema file. The JADE Developer's Reference (JADE, 2003, p. 134) describes external functions as those "which are not necessarily associated with any specific class". The `externalFunctionDefinitions` section is not normally added to the schema file by the JADE development environment unless the application is to access an external library or dynamic link library (dll) file.

The schema file was then compared to the purpose built schema file exported by the RoseJADELink add-in, revealing that the purpose built schema file contained no such `externalFunctionDefinitions` section. In order to determine the conditions that may have caused this anomaly, the schema files were scrutinised node by node. There are 93 property references made to 'VC++', in the converted Rose/UML Towers of Hanoi model file. In contrast, there are 97 references made to 'VC++', in an unconverted Rose/UML, reverse engineered C++ Towers of Hanoi model file. As

an experiment, the converted JADE Towers of Hanoi application was also reverse engineered. When the resultant Rose/UML model file was searched for references to VC++, it was revealed that this new reverse engineered model file made 76 references to the legacy VC++ language. A new blank JADE model was then created, without any UML components being added to it, or any reference made to any other language, other than the default language of JADE. The blank Rose/UML model file was then searched for references to VC++, revealing that a blank model file, associated with JADE as the default language, also refers to VC++ 76 times.

Of the 17 non-default references made to VC++ in the converted Rose/UML model file, the first is listed as an attribute property of the "Logical View" in the Design Object node, shown in Figure 22.

```

(object Design "Logical View"
  is_unit      TRUE
  is_loaded    TRUE
  attributes   (list Attribute_Set
    (object Attribute
      tool      "VC++"
      name      "Scripting"
      value     FALSE))

```

Figure 22: The converted model Design Object

The attribute property value in the Design Object in the blank model file refers to Java, shown in Figure 23, even though the default language in the Rose/UML development environment is set to JADE.

```

(object Design "Logical View"
  is_unit      TRUE
  is_loaded    TRUE
  attributes   (list Attribute_Set
    (object Attribute
      tool      "Java"
      name      "IDE"
      value     "Internal Editor"))

```

Figure 23: The blank model Design Object

The second and third references to VC++ in the converted model file are located in the Tower class object definition, shown in Figure 24. Both of these references to VC++ are made within nodes that are included in neither the Purpose built nor the new blank model files.

```
{object Class "Tower"
  attributes (list Attribute_Set
    (object Attribute
      tool      "VC++"
      name      "AppliedPattern"
      value     "none")
    (object Attribute
      tool      "VC++"
      name      "AfxSupportMacro"
      value     ""))
```

Figure 24: Converted model file Tower object.

The next six references to VC++ in the converted model file are in defining each of the class methods, an example of which may be seen in Figure 25. The third line in Figure 25 begins the object attribute reference to VC++, which concludes at the sixth line. Each of the class methods defined in the model file contains a similar reference.

```
(object Operation "tower"
  attributes (list Attribute_Set
    (object Attribute
      tool      "VC++"
      name      "Inline"
      value     TRUE)
    (object Attribute
      tool      "JADE"
      name      "Updating"
      value     TRUE))
```

Figure 25: Class method references to VC++.

The next four references were found to define the path to the original reverse engineered VC++ project and workspace files, each reference is shown in Figure 26.

```

physical_models (list unit_reference_list
  (object module "testSourceTowers" "NotAModuleType"
    "NotAModulePart"
      attributes (list Attribute_Set
        (object Attribute
          tool "VC++"
          name "ProjectFile"
          value
            "C:\\convert\\myConverter\\testSourceTowers\\testSourceTowers.dsp")
        (object Attribute
          tool "VC++"
          name "WorkspaceFile"
          value
            "C:\\convert\\myConverter\\testSourceTowers\\testSourceTowers.dsw")
        (object Attribute
          tool "VC++"
          name "Kind"
          value ("KindSet" 302))
        (object Attribute
          tool "VC++"
          name "ProjectName"
          value "testSourceTowers"))

```

Figure 26: VC++ path reference.

The final four references, displayed in Figure 27, describe properties in the model-attribute property section of the model file.

```

(object Attribute
  tool "VC++"
  name "ForwardReferences"
  value TRUE)
(object Attribute
  tool "VC++"
  name "IndentType"
  value 1)
(object Attribute
  tool "VC++"
  name "NumTabsOrSpaces"
  value 1)
(object Attribute
  tool "VC++"
  name "MaxCharsOfCommentLine"
  value 60))

```

Figure 27: Extra VC++ node definitions.

All of the code examples in Figure 22 and Figure 24 through to Figure 27 refer to model properties found to occur in the converted model file and not in the purpose built version of the same application. From the results of the comparison it was determined that it was one of these 17 nodes, still referencing the original programming language, which was causing the application methods to be considered as external methods by the RoseJADELink add-in. Removal of the offending externalFunctionDefinitions section solved the problem, leaving the static structure

conversion process complete and the schema ready to be populated with the translated algorithmic code.

5.2 Phase 2: The Algorithmic Code.

As described in section 4.2.2, phase 2 involves development of the tools needed by the conversion application to produce the translated algorithmic code for insertion into the target schema file. To provide the conversion application with the functionality necessary to translate the algorithmic code from the source language to the target language, grammars were required. An additional grammar was also required by the translation application to validate Rose/UML model files.

5.2.1 Grammar development

The parser and parse trees used during this investigation's transformation process employed grammars developed specifically for high-level to high-level translation described in section 3.2. The development of each of the individual grammars is described in the following sub-sections.

5.2.1.1 The JADE grammar

A copy of the JADE grammar developed during this study, is attached in Appendix E. From the JADE grammar and from the JADE schema file contents shown in Appendix B, it may be apparent that a JADE application schema file is highly structured. This inherent structure eased development of a grammar for JADE schema files.

The final grammar was tested successfully on several complex JADE applications, by parsing the schema files for the Erewhon example application found in the examples subdirectory of the JADE install location, used for demonstration of the JADE development environment, and the "StoryBook" application developed for handicapped children by a fellow student, (Church, 2003).

The subset C++ grammar was developed by studying the content of the applications. The same method was used in the development of the JADE grammar, described in 5.2.1.1, which had earlier proved successful. Using descriptive field names in the subset of C++ grammar enhanced the useability of the parse tree, by making recognition of the fields and their values easier than using the full C++ grammar. The C++ grammar provided by Norken Technologies was detailed and precise, but the complexity of the parse tree nodes made deciphering the values of the statements and expressions more difficult than expected. The knowledge gained from building the JADE grammar assisted the development of both the C++ and Rose grammars.

5.2.1.3 The Rose grammar

The Rose grammar was developed to validate alterations and their syntactic correctness before testing the model in the Rose/UML development environment. The Rose grammar and parser were tested on more than forty model files, including the entire MFC model, located in the Rose/UML application template subdirectories. The grammar successfully created a parse tree of the MFC model described in section 5.1.2. The tree contained more than 750,000 nodes and 255,000 LOC. This indicated that the correctness and accuracy of the grammar would be sufficient for validating the converted application model files.

5.2.2 Schema file extension

As stated in section 4.2.2.3, the JADE schema file exported from Rose/UML does not contain all the section headings required by the JADE environment. Consequently, before adding any operational code to the JADE schema file, it was necessary for the converter to append the missing headings to the end of the existing schema file.

The converter then used the JADE schema parse tree to find the name of the application schema, this was then used to create the container for the application methods. In a VC++ console application, a 'main' method is required as the entry point for the application. The 'main' method and other methods present in the C++ Towers of Hanoi application in Appendix A are not associated with any specific

class in the source application. Conversely, all application methods in JADE must be contained within either a class or the application schema.

To overcome the lack of an application class in the source program, the converter appends the name of the application schema to the end of the schema file and then opens a set of brackets, which define the boundaries of the schema's scope. The closing bracket is appended once all the relevant method details and converted algorithmic code have been inserted. The converter performs this functionality regularly throughout the conversion process. An example of such functionality is presented in Figure 29 using pseudocode:

```
For each class in the target schema file, append the class name;
  Open a bracket on a new line;
  For each method in the class append the method name;
    Append an opening brace on a new line;
    Translate and populate the method body;
    Append a closing brace on a new line;
  End For each method;
  Append a closing bracket on a new line;
End For each class;
```

Figure 29: Regularly used algorithm example in pseudocode.

As the converter reaches the 'Translate and populate the method body' step, of Figure 29, it calls the source application parse tree to provide the lines of code for each of the methods contained by the class or schema application currently being populated during the translation of the algorithmic code step.

5.2.3 Translation of the algorithmic code

One of the first tasks required by a programming-language conversion project, suggested by Terekhov and Verhoef (2000, p. 106), is a mapping of the constructs (or data types) between the source and target languages. According to Terekhov and Verhoef (2000, p. 105) many language conversion projects fail because this issue is not addressed early enough. This task was addressed in section 5.1.3 describing the model manipulation. A diagrammatic representation of the process based on their suggestion is presented in Figure 30 below.

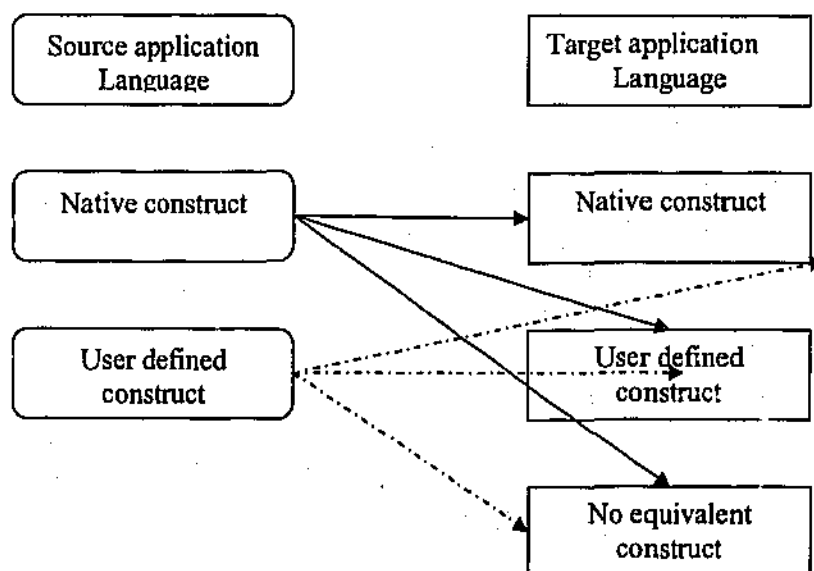


Figure 30: A mapping of the data types (Terekhov and Verhoef 2000, p. 105)

A mapping of the types associated with C++ to the recommended equivalent JADE type is to be found in the JADE Developer’s Reference (JADE, 2003, p. 144). The mapping takes into account the activation frame size of the native constructs and recommends an equivalent JADE type, shown in Table 3.

Table 3: C++ to JADE type mapping recommendations (JADE, 2003, p. 144)

C++ data type	Activation frame size / bytes	Recommended JADE type
Int	4	Integer
Long	4	Integer
Short	4	Integer
Char	4	Character
Float	4	Real[4]
Double	8	Real[8]
long double	10	Real[10]

The method shown as Figure 31 defines the process used to initialise a parser object in readiness for use, displays errors on the console as they occur. The code also shows the method of directing the parser to the input stream and the grammar used to define it. Each of these method calls is associating a file name with a stream in the parse engine, using the `setInputFilename()` and `setGrammar()` methods respectively.

Once instantiated and ready for use, the converter uses the parser to search the input file for algorithmic code contained in each of the methods within a class. Figure 32 shows a sample of code from a converter method that searches for a specific class method containing algorithmic code. If the name of the current method matches the name of the method being searched within the target class, then the value representing the code contained in that node is assigned to the `current_statement_list`. The current statement list is then returned to the calling method for analysis and conversion.

```

current_statement_list = "No statements available...\n";

do{
    //Find the next occurrence of the SearchID pattern.
    current_method_node_ID = pSourceParser->FindNext(SearchID);

    if(current_method_node_ID > 0){ //found a method
        //Get the method name for a comparison with the 'current_method_name'
        long method_nameID = pSourceParser->Find("method_name", current_method_node_ID);
        PGString this_method_name = pSourceParser->GetValue(method_nameID);

        if(this_method_name == current_method_name){
            //Access the Statements within the current_method_name from here
            //current_statement_list = "Some statement details to go here...\n";

            long s1 = pSourceParser->GetNextSibling(method_nameID);
            //cout << "\n" << pSourceParser->GetValue(s1) << "\n";
            long s2 = pSourceParser->GetNextSibling(s1);

            current_statement_list = pSourceParser->GetValue(s2);
            cout << "\n" << pSourceParser->GetValue(s2) << "\n";

        }
        }else cout << "\nNo methods..." << endl;
        //Repeat until no more methods
    }while(current_method_node_ID > 0);
}

```

Figure 32: Searching a method for algorithmic code

Results of the search for the algorithmic code contained in the `addDisks()` method, the contents returned in the `current_statement_list` object are shown in Figure 33.

```
for (i=0; i<MAXDISKS; i++) {  
    disks[i] = MAXDISKS - i;  
}  
numDisks=MAXDISKS;
```

Figure 33: Contents of `current_statement_list`.

The content of the sample source file's algorithmic code is assigned to `current_statement_list` object in C++ form, one LOC at a time. Both the `current_statement_list` and the `node_id` are then passed to the converter's `get_statement_equivalent(long node_id, PGString current_statement_list)` method, which determines whether each LOC is either a statement or an expression.

Each node in the parse tree is defined by a node label, which may be seen in Figure 34 where, in the left window, the highlighted `assignment_statement` node represents the LOC in the code window on the right. Use of the parse tree to return the node label matched to the `node_id` parameter passed to the `get_statement_equivalent(long node_id, PGString current_statement_list)` method, allows the converter to concentrate on generating the equivalent JADE statement or expression.

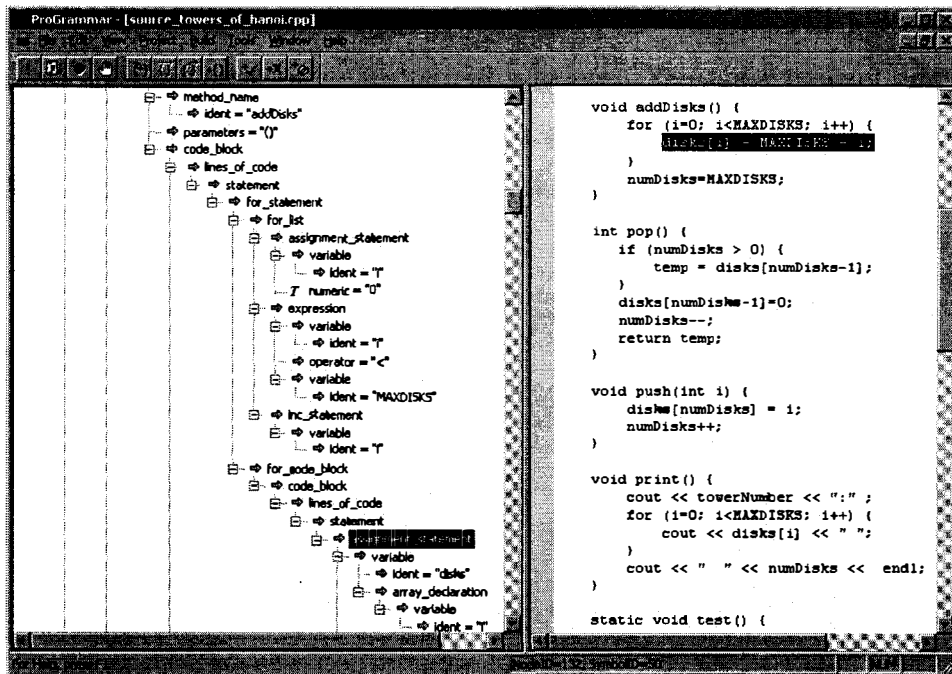


Figure 34: Towers of Hanoi addDisks() parse tree

Some statements, for example the one highlighted in the right side window in Figure 34, need very little alteration to transform them into the JADE equivalent. Statements assigning a value to a variable, even an array variable like that shown Figure 34, differ from source to target language only in the assignment symbol itself. Where in C++ the assignment uses an ‘equals’ symbol (=), in JADE the ‘colon – equals’ (:=) is used. The assignment statement translation is performed one character at a time. When the ‘=’ symbol is detected in an assignment statement, the ‘putback()’ function is used and a colon is inserted; then the rest of the LOC is processed. This process is not affected by the detection of the C++ test for equality symbol, i.e. “==”. The grammar and parser recognise the “==” pattern as part of an expression rather than as an assignment statement. Once the conversion of the assignment statement highlighted in Figure 34 is complete, the transformed assignment statement is written to the target method inside the for_statement within the JADE schema file.

Translation of a ‘for’ loop statement from the sample application source code, is performed in a similar fashion. If the statement type query for a line_of_code node returns a value equal to ‘for_statement’, each component of that line of code is

dealt with in a series of steps. Figure 34 shows the `for_statement` as a grandchild node of the `'lines_of_code'` node in the left window. The `for_statement` node has in turn two children of its own. These are shown to be the `for_list` and `for_code_block` nodes. The `for_list` node value represents the first line of the `for_statement` shown below as Figure 35.

```
for (i=0; i<MAXDISKS; i++) {
```

Figure 35: `for_list` node value.

The translation process converts the Figure 35 LOC to the JADE equivalent, shown as Figure 36, by dealing with each component in the `for_statement`'s child nodes or 'sub-tree'.

```
foreach i in 0 to MAXDISKS do
```

Figure 36: JADE equivalent to Figure 35.

A template writing method is used to produce the translated JADE equivalent in Figure 36 by using the parameters sent to it by the parser. When a 'for loop' is recognised by the converter, the component parts of the `for_list` are extracted and sent as parameters to the `get_new_for_list` method, shown in Figure 37, which then returns the re-formatted statement to the calling converter method.

```

string Statement::get_new_for_list(PGString counter, PGString start_val,
PGString end_val){
    string new_for_list = "foreach ";
    new_for_list.append(counter);
    new_for_list.append(" in ");
    new_for_list.append(start_val);
    new_for_list.append(" to ");
    new_for_list.append(end_val);
    new_for_list.append(" do");

    return new_for_list;
}
//Note: string's STL function append has been used for clarity,
//rather than its '+' operator.

```

Figure 37: Converts Figure 35 to Figure 36.

The converter uses a similar method to that in Figure 37 to transform incrementing or decrementing statements during a translation. When an `inc_statement` or a `dec_statement` is encountered during a conversion, the identifier value is sent as a parameter to the `get_new_inc_statement(PGString id)` or `get_new_dec_statement(PGString id)` method respectively. Figure 38 shows the incremental statement conversion method.

```

string Statement::get_new_inc_statement(PGString id){
    string new_inc_statement = id;
    new_inc_statement.append(" := ");
    new_inc_statement.append(id);
    new_inc_statement.append(" + 1");

    return new_inc_statement;
} //Returns id := id + 1

```

Figure 38: Method of `inc_statement` conversion.

Although simple in their coding, these methods provide the necessary translation to show proof of concept for the application translated in this investigation. Once all the algorithmic code had been converted and deposited in the target schema file, the analysis phase was initiated.

5.3 Phase 3: The Analysis and Findings

Recall from chapter 4 that, in order to achieve the goals of the investigation, it was necessary to deconstruct the processes involved in this study into 3 phases. To recapitulate:

- Phase 1 involved the selection and reverse engineering of the source applications, followed by the manipulation of the model properties and finally the export process to produce a valid target language version of the model;
- Phase 2 involved the development of language grammars used by the parser to produce parse trees that represent the subject input contents. This phase also involved the development of an application capable of extending the JADE schema file, produced by the RoseJADELINK add-in during the reverse and forward engineering and subsequent export processes. The parse trees built here provide input details used by the converter to populate the methods with the translated algorithmic code.
- Phase 3 Having investigated the processes necessary to provide a static structure schema file of the sample programs, and having built the application capable of translating the algorithmic code, the investigation proceeded to the collection and correlation of data for evaluation.

5.3.1 Data Collection and Analysis

The converted schemas were tested in the JADE environment to determine the usability of the converted code. When the sample inheritance schema was run, the code was unsuccessful due to the missing 'create' statements required to instantiate a class object. Consequently, as may be seen immediately after the 'begin' clause in Figure 39, the 'create' statements were added to the 'main' method as part of the automatic conversion process. This was necessary as C++ does not require the explicit use of a create statement after the declaration of the object. Therefore, as the statement does not exist in the source application, it is not translatable yet must be included in the process.

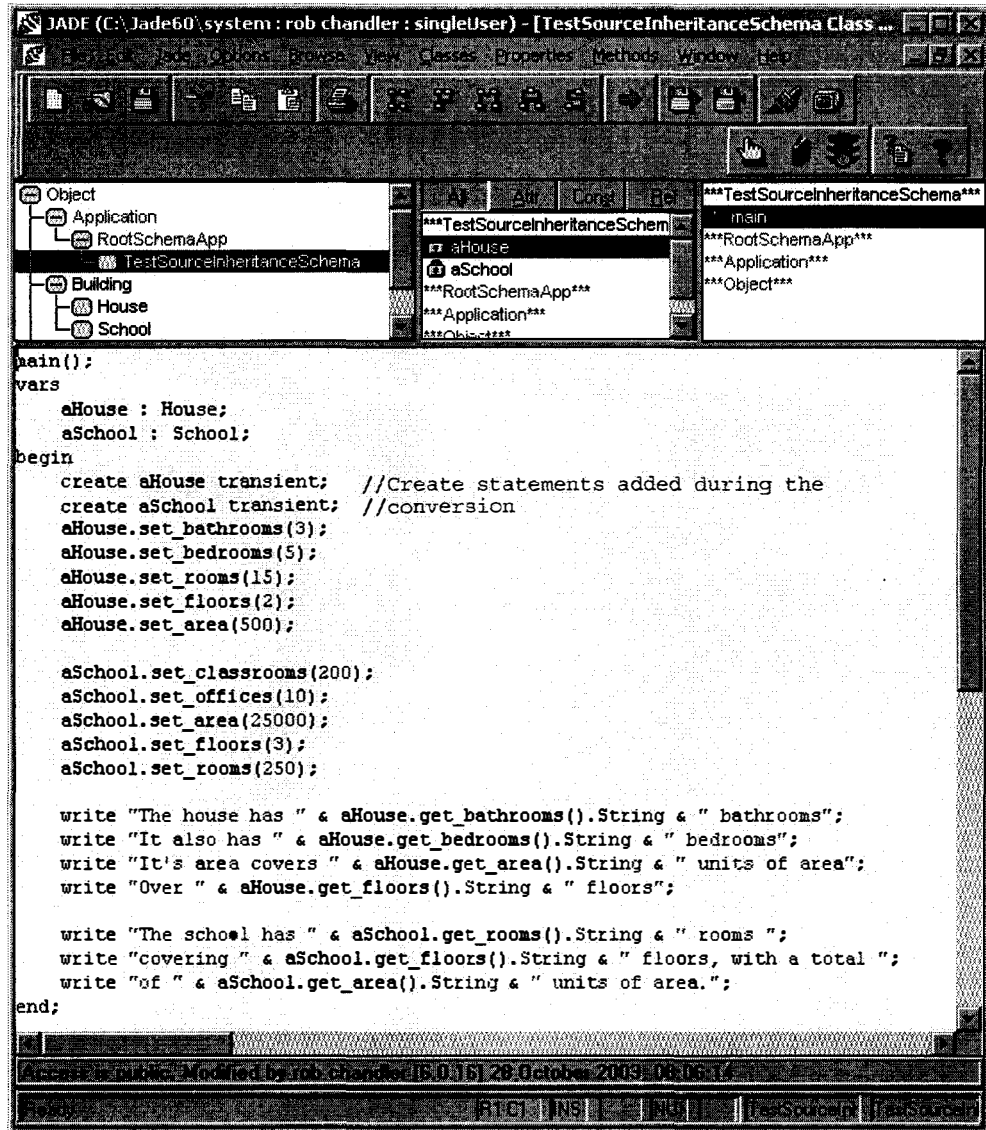


Figure 39: A converted schema imported into JADE.

The inclusion of the 'create' statements in the main method of both the applications translated during the investigation produced a complete sample inheritance schema, which was parsed successfully using the JADE grammar and one of which was operable from within the JADE environment. Invoking the converted application from within the JADE environment initiates the 'JADE Interpreter Output Viewer', as shown in Figure 40, which presents the application output.

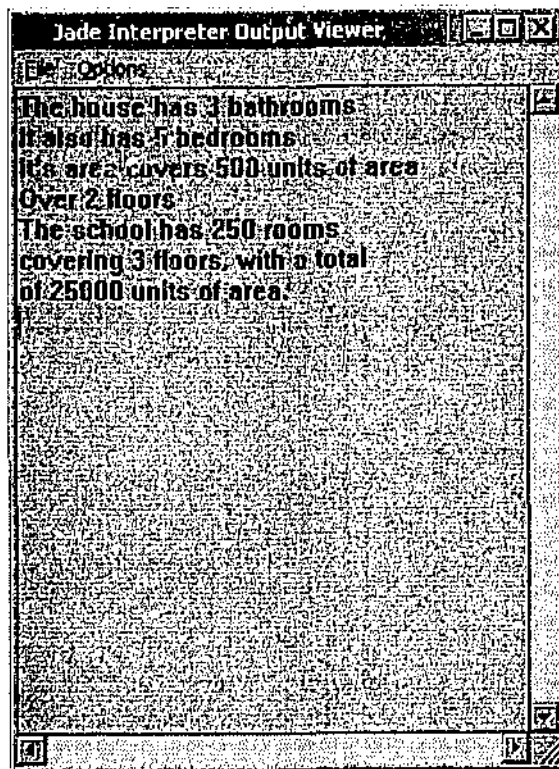


Figure 40: The building inheritance output as depicted in Figure 39

The output presented in Figure 40 is the successful culmination of using the static structure abstraction and transliteration method to translate Schildt's (2003) building inheritance application from VC++ to JADE. Use of the tools developed throughout the investigation, in conjunction with the existing parser application obtained from Norken Technologies, allowed the abstraction and transliteration method to be realised and tested.

5.4 Findings

5.4.1 Findings from the building inheritance application conversion.

The following findings relate specifically to the conversion of the sample inheritance application taken from Schildt's (2003, p. 280) text:

Table 4: building inheritance conversion data

Item	Description
Original Loc	71 LOC
Converted Loc	292 LOC
Manual Loc	ZERO
Automatic Loc	292 LOC
Time Automatic Loc	1 second
Time Manual Loc	20 minutes
Conversion Time	20 minutes 1 second
Environment	<ul style="list-style-type: none">• Windows XP• 512MB RAM• 2.0 GHz

5.4.2 Findings from the Towers of Hanoi conversion.

The following findings are specific to the conversion of the Towers of Hanoi application taken from Roeder's (2003) website:

Table 5: Towers of Hanoi conversion data

Item	Description
Original Loc	109 LOC
Converted Loc	268 LOC
Manual Loc	19
Automatic Loc	249 LOC
Time Automatic Loc	1 second
Time Manual Loc	15 minutes
Conversion Time	15 minutes 1 second
Environment	<ul style="list-style-type: none">• Windows XP• 512MB RAM• 2.0 GHz

Once the testing and analysis steps were concluded, the findings were processed and associated with the research questions.

5.5 Discussion

Manual intervention to the Towers of Hanoi schema was required to enable the schema to compile in the JADE environment. Although the schema compiled without any syntactic errors, the JADE environment found semantic errors that required debugging of the source code. Solutions to the errors found may have been included in the conversion process if time had not been a limiting factor. For example, JADE expects class methods that make assignments to have the method option 'updating' included in the method signature. To include the functionality necessary to implement adding the 'updating' option to each assigning method, would have required significant alteration to the converter logic along with an increase in investigation time. However, a manual insertion using text editor facilities achieved a satisfactory result. Such insertions are consistent and lend themselves to automation and were not regarded as significant.

During the JADE environment testing stage another error was discovered, relating to the use of 'for-loops' and array objects. The conversion of Figure 35 to Figure 36 results in a semantically and syntactically correct statement. However the logic behind the use of the statement to instantiate an array object is incorrect. An example of the completed conversion of a for-loop assigning values to the disks array is shown in Figure 41. Running the code with Figure 41 in the schema results in an 'array index out of bounds' error, due to the array index being set to zero. This is not allowed in JADE (*JADE online help*, 2001) as all JADE indices must be greater than zero. A difference between the original C++ code and the translated version is the maximum range to which each of the 'for-loops' will run.

In Figure 41, the converted for-loop would run from 'i' beginning at zero and running to MAXDISKS (which has been instantiated to 4), a total of 5 iterations. Whereas the original C++ for-loop, shown in Figure 42, would run from 'i', again at zero, whilst LESS THAN MAXDISKS, a total of 4 iterations before exiting the loop.

```
foreach i in 0 to MAXDISKS do
    disks[i] := MAXDISKS - i ;
endforeach;
```

Figure 41: Converted array assigning 'for loop'.

```
for (int i=0; i<MAXDISKS; i++) {
    disks[i] = MAXDISKS - i;
}
```

Figure 42: The original C++ 'for loop'.

Automating the instantiation of the arrays to one instead of zero, may have been achievable during the conversion; however, the process may have corrupted the assignment translation process by adding one to every assignment statement encountered, even in those statements not related to a for-loop. Again, such adjustment lends itself readily to automation but with time restraints was not regarded as significant.

5.5.1 The building inheritance conversion details

In the building inheritance translation, there was a significant rise in the number of LOC. This increase from 71 LOC to 292 LOC equals an increase of 221 LOC, which equates to an increase of over 311%. This is entirely due to the necessary inclusion of the rootSchema and is of no consequence to the executable.

Manual intervention was not required in the building inheritance conversion to realise a useable schema once the process had been tested in the JADE environment. This resulted in 100% of the converted schema being translated automatically. However, this figure still required time to modify the UML model in readiness for export to JADE and the modifications took a total of 20 minutes. Again, this might be automated with scripting language in Rose and does not detract from the overall automation of the process.

5.5.2 The Towers of Hanoi (Roeder, 20003) conversion details

As in section 5.5.1, an increase in the number of LOC from the original source application, 109 LOC, to the converted JADE equivalent application, 268 LOC, realised an increase of more than 145% in the number of LOC. The number of LOC requiring manual intervention, before, during or after the conversion, amounted to 19. The LOC requiring manual intervention, related to modification of:

- array assignments;
 - to not include zero;
- instantiation of objects to be used to assign a value to an array;
 - again zero not allowed;
- method options in those methods which update the value held by a variable;
 - append the option 'updating' to a method signature; and
- method signatures to include parameter object accessibility;
 - for example: the 'io' in Figure 43.

```
push(i : Integer io) updating;
```

Figure 43: Method signature alteration

The manual intervention required to modify the converted Towers of Hanoi schema amounted to 19 LOC, which represents a total of 92.9% of the converted schema being translated automatically. As mentioned in section 5.5.1, time was also required to modify the UML model before the conversion in preparation for the export of the model to a JADE schema. In the case of the Towers of Hanoi application, 15 minutes was required for the model to be altered in readiness for the export process to begin. As before, all manual intervention noted above lends itself readily to consistent automation and is of little negative significance to the study.

5.6 Evidence Found To Support the Research Questions

Section 5.6 restates and addresses each of the research sub-questions in turn, followed by the main research question.

5.6.1 Sub-question 1

Which properties, within a Rational Rose model file, are associated with the reverse engineered application's programming language?

A summary list of the Rose/UML model file properties associated with the reverse engineered Towers of Hanoi application's programming language follows:

1. Logical View *scripting* field;
2. Tower class *AppliedPattern* field;
3. *AfxSupportMacro* field;
4. tower's *inline* field;
5. addDisks' *inline* field;
6. pop's *inline* field;
7. push's *inline* field;
8. print's *inline* field;
9. test's *inline* field;
10. physical_model's unit reference list fields;
 - a. *ProjectFile*;
 - b. *WorkspaceFile*;
 - c. *Kind*; and
 - d. *ProjectName*;
11. *ForwardReferences* field;
12. *IndentType* field;

13. *NumTabsOrSpaces* field; and

14. *MaxCharsOfCommentLine* field.

Apart from the default language property nodes found in all model files, these seventeen properties are associated with the reverse engineered Tower of Hanoi application's programming language. In the case of the building inheritance application model files, the same nodes were repeated in relation to the source language, however, there were more references in number. The number of references to the source language in this converted model file numbered 29. This was due to the extra classes and the number of methods per class associated with the building inheritance application. Nine of the fields were repeated as in the Towers of Hanoi application. Fields 2 and 3 were repeated for each of the classes in the building inheritance application model file, an 'inline' field was repeated for each method in the classes included in the second application. Leaving fields 1 and 10 a, b, c, d, 11 through 14 repeated for the building inheritance application's model file.

5.6.2 Sub-question 2

Which components of a JADE schema file, produced by the RoseJADELink add-in, may be used to construct the static framework in preparation for code migration?

In answer to sub-question 2, all the components produced by the RoseJADELink add-in were included in the working schema, except for the externalFunctionDefinitions component discussed in detail in section 5.1.4. The components that were included in the converted schema file were:

1. schemaDefinitions;
2. constantDefinitions;
3. typeHeaders;
4. typeDefinitions;
5. databaseDefinitions;

6. schemaViewDefinitions;
7. _remapTableDefinitions;
8. externalFunctionSources; and
9. typeSources.

5.6.3 Sub-question 3

What improvement in the ratio of automatically to manually translated LOC in a legacy system may be achieved using the abstraction and re-implementation approach?

An answer to this question depends on the complexity of the application being converted, as shown by the results from each of the sample application conversions. The building inheritance application (Schildt, 2003, p. 280) provided 100% automatic conversion of the algorithmic code, without requiring manual intervention. This figure does not take into account the model manipulation mentioned in section 5.5.1 regarding the Rose/UML model, as this is in relation to the static structure abstraction and conversion.

The Towers of Hanoi achieved an improvement in the ratio of automatically to manually translated LOC of 2.9%, using the abstraction and re-implementation approach. An improvement of between 2.9% and 12.9% over the automatic translation results reported by Harsu (2000) and Terekhov (2001) respectively. This improvement translates into significant savings when applied to the figures described in section 2.2. On Terekhov's (2001) 1,500,000 LOC translation project, approximately 43,500 extra LOC may have been automatically converted, a saving of approximately \$US348, 000.

However, it is worth stating that the manual intervention noted in 5.5.2 lends itself readily to automation that may enable a projected 100% automated conversion.

5.6.4 The main question

If separation of static and algorithmic components of code for forward engineering of a legacy system is achieved, then may a reduction of manual intervention be realised in automated code conversion?

Evidence produced during this investigation proves that a reduction of manual intervention would be realised when translating applications of similar complexity using the abstraction and reimplementation approach. In the translation of legacy-system applications with an MCC rating of 3, a reduction of 2.9% in the number of LOC requiring manual intervention would be realised. With little modification, zero manual intervention may be achievable.

5.7 Chapter Summary

Details of the phases outlined in chapter 4 were presented. Implementation of the steps incorporating the phases of the investigation combined the needs outlined in the introduction and background, with the foundations provided by the studies in the literature review to develop the concepts presented in the project proposal. The chapter also stated and discussed the findings of this study, by showing excerpts of source and target model schemas and comparing and contrasting their contents to validate the findings. The study's findings have then been used to provide answers to the research questions as they were presented in section 2.4.

6 Conclusions

This investigation has detailed the phases involved in developing a programming language converter capable of using the static structure abstraction and transliteration method to translate a VC++ application to JADE. The concept presented by Waters in 1988 as more of a promise than a reality, is now achievable using today's tools and methods.

One of the objectives of this project has been to provide evidence that translating a legacy application via the static structure abstraction and transliteration method would result in a reduction of the amount of manual intervention required. This objective has been realised as shown by the findings in section 5.4. In describing the significance of this study, in section 2.2, the costs involved in translating manually from a legacy system's programming language were discussed briefly. In section 2.3, it was suggested that using the static structure abstraction and transliteration method to automate the conversion process would yield significant cost savings over the manual translation alternative. In answering the research questions in sections 2.4 and 2.4.1, the cost savings suggested by the author in section 5.6.3 are shown to be realistic and achievable.

From using the Towers of Hanoi sample application as a test case, the study's findings showed that an application with the same MCC rating would realise a reduction in manual intervention of 2.9% of the total LOC in the original application. In fact, cost savings would be realised if a reduction in manual intervention of this magnitude were applied to the best efforts of both Terekhov (2001) and Kontogiannis et al., (1998). Automation of the consistent alterations made manually may realise 100% automated code conversion.

As the study has been implemented, however, calculations from section 5.6.3 project a cost saving of approximately \$US348,000 would be realised over

Terekhov's (2001) best conversion efforts. A reduction in the number of LOC requiring manual intervention in the Kontogiannis et al., (1998) conversion would equate to approximately 8,700 LOC. Using the lower figure of \$US 8.00 per LOC (Cowley, 2003) for manual translation, a cost saving of around \$US 69,600 would be realised.

From the analysis of the data collected and correlated throughout the investigation, each of the research questions has been answered successfully. The goals of the project have been accomplished and the findings presented and discussed. Those findings revealed by this investigation advocate that significant savings in legacy-system translation costs are achievable using the static structure abstraction and reimplementing approach.

The test applications selected for translation were of levels of complexity representative of those that might be found in well-crafted application code and were not custom built for this study. These factors add to the veracity of the findings presented in the study.

Future studies include extending the translation mechanism to embrace the full C++ language and of incorporating OO source language similar to C++ e.g. Java, to extend evolution of legacy system modernisation while preserving valuable original system code aspects.

APPENDICES

Appendix A: Sample application – Towers of Hanoi.

The Towers of Hanoi sample application was used in this investigation, courtesy of Roeder (2003), as it was found on his website. Code comments have neither been added nor removed; Roeder's (2003) source code is presented below.

```
#include <iostream>
using namespace std;

const int MAXDISKS=4;

class Tower {
private:
    int towerNumber;
    int disks[MAXDISKS];
    int numDisks;
    int I;
    int temp;
public:
    Tower(int n) {
        for (i=0; i<MAXDISKS; i++) {
            disks[i]=0;
        }
        numDisks=0;
        towerNumber=n;
    }

    void addDisks() {
        for (i=0; i<MAXDISKS; i++) {
            disks[i] = MAXDISKS - I;
        }
        numDisks=MAXDISKS;
    }

    int pop() {
        if (numDisks > 0) {
            temp = disks[numDisks-1];
        }
        disks[numDisks-1]=0;
        numDisks--;
        return temp;
    }

    void push(int i) {
        disks[numDisks] = I;
        numDisks++;
    }

    void print() {
        cout << towerNumber << " : " ;
        for (i=0; i<MAXDISKS; i++) {
            cout << disks[i] << " ";
        }
        cout << " " << numDisks << endl;
    }

    static void test() {
        Tower a(1);
        a.print();
        a.addDisks();
    }
};
```

```

        a.print();
        cout << "pop " << a.pop() << endl;
        a.print();
        a.push(99);
        a.print();
    }
};

void move(Tower &from, Tower &to_, Tower &use, int depth){
    if (depth==1) {
        from.print();
        to_.print();
        use.print();
        cout << "-----" << endl;
    }
    if (depth > 0) {
        move(from, use, to_, depth-1);
        to_.push(from.pop());
        move(use, to_, from, depth-1);
    }
    if (depth==1) {
        from.print();
        to_.print();
        use.print();
        cout << "-----" << endl;
    }
}

void hanoi() {
    Tower a(1);
    Tower b(2);
    Tower c(3);

    a.addDisks();

    a.print();
    b.print();
    c.print();

    cout << "-----" << endl;
    move(a, b, c, MAXDISKS);
    cout << "-----" << endl;

    a.print();
    b.print();
    c.print();
}

void main() {
    Tower::test();
    cout << "=====" << endl;
    hanoi();
}

```

Appendix B: The generated JADE Towers of Hanoi schema file.

```
jadeVersionNumber "6.0.08";
schemaDefinition
ConvertedTowersSample subschemaOf RootSchema partialDefinition, modelSchema;
constantDefinitions
    categoryDefinition ConvertedTOHmodified
    documentationText
    `This is the Application subclass.`
        MAXDISKS : Integer = 4;
    categoryDefinition Tower
typeHeaders
    ConvertedTOHmodified subclassOf RootSchemaApp;
    GConvertedTOHmodified subclassOf RootSchemaGlobal;
    SConvertedTOHmodified subclassOf RootSchemaSession;
    Tower subclassOf Object transient;
typeDefinitions
    ConvertedTOHmodified completeDefinition
    (
    documentationText
    `This is the Application subclass.`
    constantDefinitions
        MAXDISKS : Integer = 4;
    jadeMethodDefinitions
        move(
            from : Tower io;
            to_ : Tower io;
            use : Tower io;
            depth : Integer) updating;
        hanoi() updating;
        main() updating;
    )
    GConvertedTOHmodified completeDefinition
    (
    documentationText
    `This is the Global subclass.`
    )
    SConvertedTOHmodified completeDefinition
    (
    documentationText
    `This is the WebSession subclass.`
    )
    Tower completeDefinition
    (
    attributeDefinitions
        towerNumber: Integer protected;
        disks: IntegerArray protected;
        numDisks: Integer protected;
        i: Integer protected;
        temp: Integer protected;
    jadeMethodDefinitions
        tower(n : Integer) updating;
        addDisks() updating;
        pop() : Integer updating;
        push(I : Integer io) updating;
        print() updating;
        test() updating;
    )
    ConvertedTOHmodified completeDefinition
    (
    documentationText
    `This is the Application subclass.`
    constantDefinitions
        MAXDISKS : Integer = 4;
    jadeMethodDefinitions
        move(
            from : Tower io;
            to_ : Tower io;
            use : Tower io;
            depth : Integer) updating;
        hanoi() updating;
```

```

        main() updating;
    )
    Tower completeDefinition
    {
    jadeMethodDefinitions
        tower(n : Integer) updating;
        addDisks() updating;
        pop() : Integer updating;
        push(I : Integer io) updating;
        print() updating;
        test() updating;
    )
databaseDefinitions
    ConvertedTowersSampleDb
    {
    databaseFileDefinitions
        "ConvertedTowersSample";
    defaultFileDefinition "ConvertedTowersSample";
    classMapDefinitions
        ConvertedTOHmodified in "_usergui";
        ConvertedTOHmodified in "ConvertedTowersSample";
        GConvertedTOHmodified in "ConvertedTowersSample";
        SConvertedTOHmodified in "ConvertedTowersSample";
    )

```

Appendix C: Rational Rose model file grammar.

```

/*****
*This grammar has been developed to parse UML model files, specifically
*Rational Rose .mdl files.
*It has been tested on over 40 sample models created using
*Rose Enterprise Edition Version: 2002.05.20
*and parses all of them successfully.
*It has not been tested on Rose models created with
*earlier or later versions of Rational modelling tools.
*
*The grammar has been developed using the NorKen Technologies
**ProGrammar* tool and their Grammar Definition Language
*(GDL) Available at www.programmar.com
*
*AUTH:      Rob Chandler
*DATE:      20030921
*VERSION:    1.0.2
*****/

grammar Rose <Space=" \n\r\t",
            matchcase,
            hideliterals,
            showdelimiters,
            version="1.0.2">
{
    schema ::= [{Object}];           //Describes the model itself

    /* ***** LITERALS AND TERMINALS ***** */

    literal ::= boolean_literal | numeric_literal | string_literal ;
    boolean_literal ::= "TRUE" | "FALSE" ;
    numeric_literal <TERMINAL, TOKEN=NULL> ::= [sign] numeric [{"|" | "."}
numeric];
    sign <TERMINAL, BACKTRACK> ::= ("+" | "-") ;
    numeric ::= "[0-9]+" ;
    string_literal <TERMINAL, SPACE=""> ::= "\" *{"\""} "\" ;

    obj ::= "object" ; // term used often

    value ::= atValue
            | boolean_literal           //TRUE / FALSE
            | value_set                 //(111,111)
            | numeric_literal           //int or float
            | string_literal            //Any double quote delimited string
            | sub_property              //A literal followed by a value
            | comment_line              //Comment or documentation begins a line with

the |
            | "(" Text ")"             //type of comment
            | "uses\\"                 //irregular option
            | "extends\\"              //ditto
            | "Last name\\"            //more of the same

    atValue ::= "@" literal ;
    value Set ::= "(" {numeric_literal, ","} ")" ;
    sub_property ::= "(" literal value ")" ;
    Text ::= value_type comment_line ;
    value_type ::= "value cardinality" | "value Text" ;
    comment_line ::= "{"|" stuff } | literal ;
    stuff ::= *(comment_end) ;//regular expression
    comment_end ::= #BOL ("32" | "\t") ; //Beginning Of Line followed by
whitespace

    /* ***** OBJECT FORMAT ***** */

    Object ::= "(" obj Object_Name [{value}] [{Object_Properties}] ")" ;

    /* ***** OBJECTS NAMES ***** */

```



```

Object_Name ::= "action"
"ActionTime"
"ActivityDiagram"
"ActivityState"
"ActivityStateView"
"AssocAttachView"
"Association"
"AssociationViewNew"
"AttachView"
"Attribute"
"CategoryView"
"ClassAttribute"
"Class_Category"
"Class"
"Class_UTILITY"
"ClassDiagram"
"ClassView"
"Compartment"
"Connection_Relationship"
"ConnectionView"
"DataFlowView"
"Decision"
"DecisionView"
"defaults"
"Dependency_Relationship"
"Design"
"Device"
"DeviceView"
"Event"
"external_doc"
"Focus_Of_Control"
"Font"
"ImportView"
"Inheritance_Relationship"
"InheritTreeView"
"InheritView"
"Instantiated_Class"
"Instantiation_Relationship"
"InstantiateView"
"InteractionDiagram"
"InterfaceView"
"InterMessView"
"InterObjView"
"ItemLabel"
"Label"
"Link"
"LinkSelfView"
"LinkView"
"Mechanism"
"Message"
"MessView"
"Module_Diagram"
"module"
"Module"
"Module_Visibility_Relationship"
"ModView"
"ModVisView"
"NoteView"
"ObjectDiagram"
"Object"
"ObjectView"
"Operation"
"Parameter"
"Parameterized_Class"
"Partition"
"Petal"
"Process_Diagram"
"Process"
"Processes"
"Processor"
"ProcessorView"
"Properties"
"Realize_Relationship"
"RealizeView"
"Role"
"RoleView"
"SegLabel"

```

```

"SelfMessView"
"SelfTransView"
"sendEvent"
"State_Diagram"
"State_Machine"
"State"
"State_Transition"
"StateView"
"SubSystem"
"SubSysView"
"Swimlane"
"SynchronizationState"
"SynchronizationView"
"Tier_Diagram"
"TierView"
"TransView"
"UseCase"
"UseCaseDiagram"
"UseCaseView"
"UsesView"
"Uses_Relationship"
"Visibility_Relationship"

```

```

/* ***** OBJECT PROPERTIES ***** */

```

```

Object_Properties ::= Object_Key (value | Object | Object_List) ;

```

```

Object_Key ::= "abstract"

```

```

"action"
"actions"
>ActionTime"
"anchor"
"anchor_loc"
"annotation"
"AssociationClass"
"attributes"
"autoResize"
"bold"
"bottomMargin"
"cardinality"
"characteristics"
"charSet"
"class"
"class_attributes"
"client"
"client_cardinality"
"clipIconLabels"
"collaborators"
"color"
"compartment"
"compartmentItems"
"concurrency"
"condition"
"connections"
"const"
"Constraints"
"Containment"
"creation"
"creationObj"
>DataFlowView"
"defaultFont"
"defaults"
"default_color"
"derived"
"dir"
"documentation"
"drawSupplier"
"Event"
"exceptions"
"exportControl"
"external_docs"
"external_doc_path"
"external_doc_url"
"face"
"file_name"
"fill_color"

```

"Focus_Entry"
"Focus_Of_Control"
"Focus_Src"
"font"
"frequency"
"friend"
"global"
"gridX"
"gridY"
"height"
"hidden"
"icon"
"icon_height"
"icon_style"
"icon_width"
"icon_y_offset"
"IncludeAttribute"
"IncludeOperation"
"inivt"
"instantiation_relationship"
"InterObjView"
"is_aggregate"
"is_loaded"
"is_navigable"
"is_principal"
"is_unit"
"italics"
"items"
"justify"
"keys"
"label"
"language"
"leftMargin"
"line_color"
"line_style"
"location"
"logical_models"
"logical_presentations"
"max_height"
"max_width"
"mechanism_ref"
"messages"
"MessView"
"module"
"multi"
"name"
"Nested"
"nestedClasses"
"nlines"
"nonclass"
"nonclassname"
"notation"
"object_arc"
"Operation"
"operations"
"opExportControl"
"ordinal"
"orientation"
"origin"
"origin_attachment"
"origin_x"
"origin_y"
"pageOverlap"
"parameters"
"Parent_View"
"partitions"
"path"
"pctDist"
"persistence"
"physical_models"
"physical_presentations"
"priority"
"process_structure"
"processes"
"ProcsNDevu"
"properties"
"protocol"

```

"quid"
"quidu"
"rank"
"realized_interfaces"
"result"
"rightMargin"
"roles"
"roleview_list"
"root_category"
"root_subsystem"
"root_usecase_package"
"scheduling"
"sendEvent"
"sequence"
"showClassOfObject"
"ShowCompartmentStereotypes"
"showMessageNum"
"ShowOperationSignature"
"size"
"snapToGrid"
"statediagram"
"statediagrams"
"statemachine"
"states"
"static"
"stereotype"
"strike"
"subsystem"
"subobjects"
"superclasses"
"supplier"
"supplier_cardinality"
"supplier_is_device"
"supplier_is_spec"
"supplier_is_subsystem"
"SuppressAttribute"
"SuppressOperation"
"sync_is_horizontal"
"synchronization"
"terminus"
"terminal_attachment"
"title"
"tool"
"topMargin"
"transitions"
"type"
"uid"
"underline"
"used_nodes"
"value"
"version"
"vertices"
"virtual"
"visible_categories"
"visible_modules"
"when"
"width"
"written"
"x_offset"
"y_coord"
"y_offset"
"zoom"
;

/* ***** LIST DEFINITIONS *****
*/

Object_List ::= "(" "list" [Object_List_Type] [{"Object | Object_Key value |
value}]] ")";

Object_List_Type ::= "action_list"
| "Attribute_Set"
| "class_attribute_list"
| "Compartment"
| "connection_list"
| "dependency_list"
| "diagram_item_list"

```

```
"external_doc_list"  
"inheritance_relationship_list"  
"link_list"  
"Messages"  
"nestedClasses"  
"Operations"  
"Parameters"  
"Partitions"  
"Points"  
"processes"  
"realize_rel_list"  
"role_list"  
"RoleViews"  
"StateDiagrams"  
"States"  
"transition_list"  
"unit_reference_list"  
"uses_relationship_list"  
"visibility_relationship_list"
```

};

Appendix D: A subset of C++ grammar.

```
//MyCPPsubset is a subset of the C++ language, focussing specifically on the
//statements contained within the methods of the Towers of Hanoi application
//used by this investigation.
//Permission for the use or alteration of this grammar, in full or in part
//is hereby given.
//CREATED BY:          Rob Chandler
//CREATED ON:          20031023
grammar myCPP <HIDELITERALS,
               HIDEREPEATERS,
               SPACE="\n\r\t\32",
               NOBACKTRACK>

{
    towers_of_hanoi ::= {{file_contents}};

    file_contents ::= {pre_processor_statements} [namespace_declaration]
                    {{global_variable_declarations}} {{class_declaration}}
                    {{application_methods}}
                    ;

    pre_processor_statements ::= pp_symbol "include" pp_object ;
    pp_symbol ::= "#" ;
    pp_object ::= (open_delimiter pp_subject close_delimiter) | string_literal ;
    open_delimiter ::= "<" ;
    close_delimiter ::= ">" ;
    pp_subject ::= "stdio.h" | "iostream" ;

    namespace_declaration ::= "using namespace std;" ;

    global_variable_declarations ::=
        {type_prefix} variable_declaration initializer ";" ;
    type_prefix ::= "const" ;

    variable_declaration ::= type ident {array_declaration} ;
    type ::= "int" ;

    array_declaration ::= "{" (expression | variable) "}" ;
    initializer ::= "=" (numeric | ident) ;

    variable ::= ident {array_declaration} ;

    class_declaration ::= "class" ident [base_class]"{" (class_contents) "}" ;
    base_class ::= ":" access_specifier ident ;
    class_contents ::= access_specifier ":" {{class_attributes}} {{class_method}} ;

    access_specifier ::= "public" | "protected" | "private" ;
    class_attributes ::= variable_declaration ";" ;
    class_method ::= operation ;
    application_methods ::= operation ;

    operation ::= [method_type][return_type] method_name parameters code_block ;
    method_type ::= "static" ;
    return_type ::= "void" | type ;
    method_name ::= ident ;
    parameters ::= "(" [parameter_list] ")" ;
    parameter_list ::= parameter [{","} parameter] ;
    parameter ::= (type | "Tower") [address_delimiter] ident ;
    address_delimiter ::= "&" ;
    code_block ::= "{" {{lines_of_code}} "}" ;

    lines_of_code ::= statement
                    | expression
                    | method_call [{";"}]
                    | output_call
                    | object_initializer
                    ;

    object_initializer ::= ident [class_specifier] (method_call | ident) ";" ;
    class_specifier ::= ":" ;

    expression ::= variable operator (variable | numeric);
```

```

operator ::= "-"
          | "+"
          | "<"
          | ">"
          | "="
          ;

method_call ::= (scoped_name | method_name) "(" [value_list] ")" ;
scoped_name ::= ident "." ident ;
value_list ::= value [{"," value}] ;
value ::= expression
        | method_call
        | variable
        | numeric
        | string_literal
        ;

output_call ::= "cout" [{output}] [flush] ";" ;
output ::= output_operators (method_call
                             | variable
                             | string_literal
                             | numeric)
        ;

output_operators ::= "<<" ;
flush ::= output_operators "endl" ;

statement ::= assignment_statement
           | for_statement
           | if_statement
           | inc_statement
           | dec_statement
           | return_statement
           ;

assignment_statement ::= variable "=" (expression | variable | numeric) ";" ;

for_statement ::= "for" "(" for_list ")" for_code_block ;
for_list ::= assignment_statement expression ";" inc_statement ;
for_code_block ::= code_block ;

if_statement ::= "if" "(" expression ")" if_code_block ;
if_code_block ::= code_block ;

inc_statement ::= variable "++" [";"];
dec_statement ::= variable "--" [";"];
return_statement ::= "return" (variable | numeric) ";" ;

string_literal <TERMINAL>
              SPACE=" " ::=
              { text_segment, [whitespace] } ;

text_segment ::=
["L"] "\"" text_elem [more_text_elems] "\"" ;

text_elem ::=
"(\\"";

more_text_elems ::=
"\\. Text_elem [more_text_elems] ;

ident <TERMINAL> ::=
identifier (? #VALUE ( ::= reserved_word );) ;

identifier ::=
[a-zA-Z_][a-zA-Z0-9_]* ;

reserved_word ::=
__asm__ | "else" | "operator"
| "auto" | "enum" | "typedef"
| "break" | "extern" | "private" | "union"
| "case" | "far" | "__far"
| "_huge" | "__huge"
| "protected" | "unsigned"
| "catch" | "float" | "public"
"virtual" | "cdecl" | "for" | "register" | "void"

```

```

"volatile" | "char" | "friend" | "return" |
| "class" | "goto" | "short" | "using"
| "const" | "if" | "signed" | "while"
| "continue" | "inline" | "sizeof"
| "namespace"
| "default" | "int" | "static" | "typename"
| "delete" | "long" | "struct"
"__uidof"
| "do" | "near" | "switch" | "try"
| "__try" | "throw"
| "double" | "new" | "template" | "finally"
| "__finally" | "except" | "__except" | "__leave"
| "__int8" | "__int16" | "__int32" | "__int64"
| "cdeclspec" | "__cdeclspec" | "__based"
| "__forceinline"
| "__virtual_inheritance" | "__multiple_inheritance"
"__single_inheritance"
| "explicit"
| "__export" | "__export"
// call modifiers
| "__cdecl" | "_cdecl" | "__fastcall"
| "__stdcall" | "_stdcall" | "__syscall" | "__oldcall"
| "__unaligned" | "pascal" | "_pascal" | "__pascal"
;

```

```

};

```


Appendix E: The JADE language grammar

```
//*****
//*****
//Jade Grammar version 1.5
//Date created 20030520
//Rob Chandler
//Modified: 20031005: R Chandler, To include changes to JADE schema files
//targeting external functions sections.
//*****
//*****

grammar Jade <SPACE="\32\t\n\r",
              NOBACKTRACK>
{
//SCHEMA STRUCTURE

schema ::=
    [versionSection]
    [schemaDefinitionSection]
    [globalConstantSection]
    [localeSection]
    [translatableStringSection]
    [localeFormatSection]
    [librariesSection]
    [externalFunctionSection]
    [typeHeaderSection]
    [typeMembershipSection]
    [typeDefinitionSection]
    [extKeyDefinitionSection]
    [memKeyDefinitionSection]
    [inverseDefinitionSection]
    [databaseDefinitionSection]
    [dbServerSection]
    [schemaViewSection]
    [exposedListSection]
    [remapTableSection]
    [externalFunctionSourceSection]
    [typeSourceSection]
    ;

versionSection ::=
    ("jadeVersionNumber" | "jadePatchRelease") stringLiteral ";" ;
    stringLiteral <TERMINAL, SPACE=""> ::= "\" *("\"" "\"" "\"";
/*    stringLiteral <TERMINAL, SPACE=""> ::= { textSegment, [whitespace] } ;
    textSegment ::= ["L"] "\" textElement [textElements] "\" ;
    textElements ::= '\\.' textElement [textElements] ;
    textElement ::= *('\\\\')' ;
*/

    whitespace ::= '[\32\t\n\r]+' ;

schemaDefinitionSection ::=
    "schemaDefinition"
    schemaName ["subschemaOf" (schemaName | "null")} schemaOptionList ";" [textSection] ;
    schemaName ::= identifier ;
    identifier <TERMINAL> ::=
        ident (? #VALUE !::= reservedWord; );

    ident ::=
        '[a-zA-Z_][a-zA-Z0-9_$]*' ;

    reservedWord ::=
        "abortTransaction" | "and" | "Any"
        | "as" | "app" | "attributeDefinitions"
        | "begin" | "beginLoad" | "beginLock"
        | "beginTransaction" | "beginTransactionTransient"
        | "Binary" | "Boolean" | "break" | "call"
        | "categoryDefinition" | "Character"
        | "classMapDefinitions" | "_cloneOf"
        | "commitTransaction" | "commitTransientTransaction"
```

```

"constantDefinitions" | "constants"
"continue" | "create" | "currentSchema"
"currentSession" | "databaseDefinitions"
"databaseFileDefinitions" | "Date"
"dbServerDefinitions" | "defaultFileDefinition"
"Decimal" | "delete" | "div" | "do"
"documentationText" | "else" | "elseif"
"_encryptedSource" | "_endEncryptedSource"
"end" | "endforeach" | "endif" | "endLoad"
"endLock" | "endwhile" | "epilog"
"eventMethodMappings" | "exception"
"_exposedConstantDefinitions"
"_exposedMethodDefinitions"
"_exposedPropertyDefinitions"
"externalFunctionDefinitions"
"externalFunctionSources" | "externalKeyDefinitions"
"externalMethodDefinitions" | "externalMethodSources"
"false" | "foreach" | "global" | "if" | "in"
"Integer" | "inverseDefinitions" | "is"
"jadeMethodDefinitions" | "jadeMethodSources"
"jadePatchRelease" | "jadeVersionNumber"
"libraryDefinitions" | "localeDefinitions"
"localeFormatDefinitions" | "memberKeyDefinitions"
"membershipDefinitions" | "MemoryAddress"
"method" | "methodImplementations" | "mod"
"node" | "not" | "null" | "of" | "on"
"or" | "Point" | "parentOf" | "peerOf"
"primitive" | "process" | "raise" | "read"
"Real" | "referenceDefinitions"
"_remapTableDefinitions" | "return" | "reversed"
"rootSchema" | "schemaDefinition"
"schemaViewDefinitions"
"self" | "setModifiedTimeStam" | "step"
"String" | "subclassOf" | "subschemaOf"
"system" | "terminate" | "then" | "Time"
"TimeStamp" | "to" | "translatableStringDefinitions"
"true" | "typeDefinitions" | "typeHeaders"
"typeSources" | "vars" | "where" | "while"
"write" | "xor"

```

```

schemaOptionList ::= schemaOption [{"", " schemaOption"}] ;
schemaOption ::= completenessOption | ("patchVersion" "="
numericLiteral) | ("patchVersioningEnabled" "=" booleanLiteral) | schema_type;
completenessOption ::= "completeDefinition" |
"partialDefinition" ;
numericLiteral <TERMINAL, TOKEN=NULL> ::= /* {sign}*/ numeric
[{" ":" | "."} numeric] ;
sign ::= {"+" | "-"} ;
numeric ::= "[0-9]+" ;
booleanLiteral ::= "true" | "false" ;
schema_type ::= "modelSchema";

textSection ::= "documentationText" [textBlock] ;
textBlock ::= textBlockDelimiter *(textBlockDelimiter)
textBlockDelimiter ;
textBlockDelimiter ::= "" ;

globalConstantSection ::=
"constantDefinitions"
[{"categoryDefinition | constantDefinition"}] ;
categoryDefinition ::= "categoryDefinition" identifier [{"constantDefinition}]
;
constantDefinition <TERMINAL, TOKEN=NULL> ::= identifier [{"constantType} "="
constExpression [constantOptionList] ";" [{"documentationText" textBlock} timestamp] ;
constantType ::= fixedSizeType | "String" [ [{"identifier | literal
"}" ] | "Binary" [ [{"identifier | literal "}" ] | "Decimal" decimalDescriptor ;
fixedSizeType ::= "Integer" | "Character" | "Boolean" | "Real"
| "Date" | "Time" | "TimeStamp" | "Point" ;
literal ::= "null" | formLiteral | numericLiteral |
characterLiteral | booleanLiteral | stringLiteral ;
formLiteral ::= "" *{"'" " " ;
characterLiteral ::= [{"L"} "\\?{[0-9A-Za-z+|.]}\'" ;
constExpression ::= [{"#"} expression ;
constantOptionList ::= constantOption [{"", " constantOption"}] ;
constantOption ::= "subschemaHidden" ;

```

```

timestamp <BACKTRACK> ::= "setModifiedTimeStamp" alphaLiteral
[alphaLiteral] [numeric] dateTime ";" ;
alphaLiteral ::= characterLiteral | stringLiteral ;
dateTime ::= numeric [{"." | "."} numeric] ;

localeSection ::=
"localeDefinitions"
[{"numericLiteral [stringLiteral] ["_cloneOf" numericLiteral] "," }];

translatableStringSection ::=
"translatableStringDefinitions"
[{"localeTranslatableStrings"}] ;
localeTranslatableStrings ::= numericLiteral [stringLiteral] "("
{translatableStringDefinition ";" } ")" ;
translatableStringDefinition ::= identifier [{"("
[identifierList] ")" } "=" {transStringExpression | (stringLiteral
{transStringExpression})} ;
identifierList ::= identifier [{"," identifier}] ;
transStringExpression ::= ("% identifier) |

expressionList ;
expressionList ::= expression [{"(", " | ";"}
expression});
expression ::= [sign] [literal |
methodOrFunctionCall[{"callArgument"}] [typeExpression];
typeExpression ::= arithmeticExpression |
booleanExpression | relationExpression ;
arithmeticExpression ::= arithmeticOperator
[expression] ;
arithmeticOperator ::= "+" | "-" | "*" | "/" |
"mod" | "div" | "^" | "&";
booleanExpression ::= booleanOperator
[expression] ;
booleanOperator ::= "and" | "or" | "not" |
"xor";
relationExpression ::= relationOperator
[expression] ;
relationOperator ::= "=" | "<>" | "<" | ">" |
"<=" | ">=" ;

localeFormatSection ::=
"localeFormatDefinitions"
[{"localeFormatDefinition"}] ;
localeFormatDefinition ::= identifier ":" className "(" valueList ")" ";" ;
className ::= modifiedIdentifier ;
modifiedIdentifier ::= identifier [{"." identifier}];
valueList ::= literal [{"," literal] ;

librariesSection ::=
"libraryDefinitions"
["aLibrary"] /*{identifier}*/ ;

externalFunctionSection ::=
"externalFunctionDefinitions"
[{"externalFunctionHeader ["documentationText" textBlock] [timestamp]}] ;
externalFunctionHeader ::= functionName "(" [functionParamDeclList] ")"
[functionReturnType] externalLocation [functionOptionList] ";" ;
functionName ::= modifiedIdentifier ;
functionParamDeclList ::= functionParamDeclGroup [{";"
functionParamDeclGroup}] ;
functionParamDeclGroup <TERMINAL, TOKEN=NULL> ::= identifierList
":" externalType [paramOption] ;
externalType ::= "Integer" | "Character" | "Boolean" |
"Real" [literal] | "Point" | "String" [literal] | "Binary" [literal] |
"IntegerArray";
paramOption ::= "constant" | "input" | "output" | "io" ;
functionReturnType <TERMINAL, TOKEN=NULL> ::= ":" externalType ;
externalLocation ::= "is" [{"identifier | alphaLiteral} "in"
identifier ;
functionOptionList ::= functionOption [{";" functionOption}] ;
functionOption ::= "subschemaHidden" |
"presentationClientExecution" | "applicationServerExecution" ;

typeHeaderSection ::=
"typeHeaders"
[{"typeHeader"}] ;

```

```

typeHeader ::= typeName "subclassOf" ((className | "null") | "primitive")
[typeOptionList]" ;
    typeName ::= className | primitiveType ;
    primitiveType ::= fixedSizeType | "Any" | "Binary" | "Boolean"
| "Character" | "Date" | "Decimal" | "Integer" | "Point" | "Real" | "String" | "Time"
| "TimeStamp" ;
    typeOptionList ::= typeOption [{"," typeOption}] ;
    typeOption ::= typeOptionNumeric | typeOptionString ;
    typeOptionNumeric ::= ("highestSubId" "="
numericLiteral) | ("number" "=" numericLiteral) | ("maxBlockSize" "=" numericLiteral)
;
    typeOptionString ::= "abstract" | "transient" |
"protected" | "subschemaHidden" | "duplicatesAllowed" ;

typeMembershipSection ::=
"membershipDefinitions"
[{membershipDefinition}] ;
    membershipDefinition ::= className "of" typeSpecifier ";" ;
    typeSpecifier ::= dimensionedType | primitiveType | className ;
    dimensionedType ::= ("String" "[" literal "]" ) | fixedSizeType
| ("Binary" "[" literal "]" ) | ("Decimal" decimalDescriptor) | "Any" ;
    decimalDescriptor ::= "[" constExpression [","
constExpression]" ;

typeDefinitionSection ::=
"typeDefinitions"
[{typeDefinition}] ;
    typeDefinition ::=
    typeName [completenessOption] "(" {textSection} [timestamp]
{constantsSection} [attributesSection] [referencesSection] [jadeMethodsSection] [e
xternalMethodsSection] [eventMethodsSection]" ;
    constantsSection ::= "constantDefinitions"
[{constantDefinition}] ;
    attributesSection ::= "attributeDefinitions"
[{attributeDefinition}] ;
    attributeDefinition ::= identifier ":" typeSpecifier
[attributeOptionList] ";" ["documentationText" textBlock] [timestamp] ;
    attributeOptionList ::= attributeOption [{","
attributeOption}] ;
    attributeOption ::=
attributeOptionNumeric | attributeOptionString ;
    attributeOptionNumeric ::=
("subId" "=" numericLiteral) | ("number" "=" numericLiteral) ;
    attributeOptionString ::=
"readonly" | "protected" | "virtual" | "required" | "subschemaHidden" |
"implicitInverse" | "implicitMemberInverse" | "explicitInverse" |
"explicitEmbeddedInverse" | "transientToPersistentAllowed" ;
    referencesSection ::= "referenceDefinitions"
[{referenceDefinition}] ;
    referenceDefinition ::= identifier ":" typeSpecifier
[referenceOptionList]" ";" ["documentationText" textBlock] [timestamp] ;
    referenceOptionList ::= referenceOption [{","
referenceOption}] ;
    referenceOption ::=
referenceOptionNumeric | referenceOptionString ;
    referenceOptionNumeric
<TOKEN=NULL> ::= ("subId" "=" numericLiteral) | ("number" "=" numericLiteral) ;
    referenceOptionString ::=
"readonly" | "protected" | "virtual" | "required" | "subschemaHidden" |
"implicitInverse" | "implicitMemberInverse" | "explicitInverse" |
"explicitEmbeddedInverse" | "transientToPersistentAllowed" ;
    jadeMethodsSection ::= "jadeMethodDefinitions"
[{jadeMethodHeader ["documentationText" textBlock] [timestamp]]] ;
    JadeMethodHeader ::= methodName
 "(" [parameterList]" )" [returnType] [methodOptionList]" ;
    methodName ::= ["app" | "create" | "delete" |
"self"] [identifier] [{"." Identifier}] ;
    parameterList ::= parameter [{"," parameter}] ;
    parameter <TERMINAL, TOKEN=NULL> ::= identifier
["," identifier] ":" typeName [paramOption] ;
    returnType <TERMINAL, TOKEN=NULL> ::= ":"
typeName ;
    methodOptionList ::= methodOption [{","
methodOption}] ;
    methodOption ::= methodOptionString |
("number" "=" numericLiteral) ;

```

```

methodOptionString ::=
"protected" | "updating" | "abstract" | "mapping" | "subschemaHidden" |
"clientExecution" | "serverExecution" | "lockReceiver";
externalMethodsSection ::= "externalMethodDefinitions"
[({externalMethodHeader [documentationText] textBlock [timestamp]})];
externalMethodHeader ::= methodName "(" [parameterList]
")" [returnType] [externalLocation] [methodOptionList] ";" ;
eventMethodsSection ::= "eventMethodMappings" [({identifier "="
identifier "of" typeName ";"})];

extKeyDefinitionSection ::=
"extKeyDefinitions"
[({classExternalKeys});
classExternalKeys ::= className [completenessOption] "("
[({externalKeyDefinition})] ")" ;
externalKeyDefinition <TERMINAL, TOKEN=NULL> ::= identifier ":"
typeSpecifier [keyOptionList] [sortOrder] ";" ;
keyOptionList ::= keyOption [{"." keyOption}];
keyOption ::= "descending" | "caseInsensitive" ;
sortOrder ::= numericLiteral ;

memKeyDefinitionSection ::=
"memberKeyDefinitions"
[({classMemberKeys});
classMemberKeys ::= className [completenessOption] "(" [({memberKeyDefinition})]
")" ;
memberKeyDefinition ::= keyPath [keyOptionList] [sortOrder] ";" ;
keyPath <TERMINAL, TOKEN=NULL> ::= identifier [{"." Identifier}]
;

inverseDefinitionSection ::=
"inverseDefinitions"
[({inverseDefinition});
inverseDefinition ::= referenceSpecifier referenceHierarchy referenceSpecifier
[({booleanOperator referenceSpecifier})] ";" ;
referenceSpecifier ::= identifier "of" className [inverseOption] ;
inverseOption ::= "manual" | "automatic" | "manualAutomatic" ;
referenceHierarchy ::= "peerOf" | "parentOf";

databaseDefinitionSection ::=
"databaseDefinitions"
[({databaseDefinition});
databaseDefinition ::= identifier "(" [databaseFilesSection]
[defaultFileSection] [classMapsSection] ")" ;
databaseFilesSection ::= "databaseFileDefinitions"
[({databaseFileDefinition});
databaseFileDefinition ::= alphaLiteral ["in" alphaLiteral]
[databaseFileOption] ";" ;
databaseFileOption ::= "number" "=" numericLiteral ;
defaultFileSection ::= "defaultFileDefinition" alphaLiteral ";" ;
classMapsSection ::= "classMapDefinitions" {className "in" (identifier
| alphaLiteral) [classMapOption] ";"};
classMapOption ::= "allInstances" | "subobjectInstances" |
"extend" ;

dbServerSection ::=
"dbServerDefinitions"
[({[identifier] "in" identifier [dbServerOptionList] ";"})];
dbServerOptionList ::= dbServerOption {"," dbServerOption} ;
dbServerOption ::= "remoteLocation" | "tcpipConnection" ;

schemaViewSection ::=
"schemaViewDefinitions"
[({schemaViewDefinition});
schemaViewDefinition ::= identifier "(" {className ";" } ")" ;

exposedListSection ::=
"exposedListDefinitions"
[({exposedListDefinition});
exposedListDefinition ::= identifier [exposedListOptionList] "("
[exposedClassDefinition] ")" ;
exposedListOptionList ::= exposedListOption {"," exposedListOption} ;
exposedListOption ::= "version" "=" numericLiteral |
"priorVersion" "=" numericLiteral | "registryId" "=" stringLiteral ;
exposedClassDefinition ::= className [exposedClassOptionList] "("
[exposedConstantsSection] [exposedPropertiesSection] [exposedMethodsSection] ")" ;

```

```

exposedClassOptionList ::= exposedClassOption {","
exposedClassOption} ;
    exposedClassOption ::= "autoAdded" ;
    exposedConstantsSection ::= "_exposedConstantDefinitions"
({exposedConstantDefinition} ;
    exposedConstantDefinition ::= identifier ";" ;
    exposedPropertiesSection ::= "_exposedPropertyDefinitions"
({exposedPropertyDefinition} ;
    exposedPropertyDefinition ::= identifier ";" ;
    exposedMethodsSection ::= "_exposedMethodDefinitions"
({exposedMethodDefinition} ;
    exposedMethodDefinition ::= methodName ";" ;

remapTableSection ::=
" remapTableDefinitions"
[{{remapTableDefinition}} ;
    remapTableDefinition ::= identifier [remapTableOptionList] "("
[{{remapFileDefinition}} ")" ;
    remapTableOptionList ::= remapTableOption[{" remapTableOption}] ;
    remapTableOption ::= "description" "=" stringLiteral ;
    remapFileDefinition ::= alphaLiteral "is" alphaLiteral ["in"
alphaLiteral] ";" ;

externalFunctionSourceSection ::=
"externalFunctionSources"
[{{functionName {" externalFunctionSource "}}} ;
    externalFunctionSource ::= externalFunctionHeader ;

typeSourceSection ::=
"typeSources"
[{{typeSource}} ;
    typeSource ::= typeName "(" [{{jadeMethodSourcesSection}}]
[{{externalMethodSourcesSection}}] ")" ;
    jadeMethodSourcesSection ::= "jadeMethodSources" [{{methodName {"
[{{comment}}] jadeMethodSource [{{comment}}]"}}}] ;
    jadeMethodSource ::= JadeMethodHeader [{{localConstsSection}}
[{{localVarsSection}} "begin" [{{instructions}}] [{{epilog}} instructions] "end" ";" ;
    localConstsSection ::= "constants"
[{{localConstDefinition}}] ;
    localConstDefinition ::= identifier [":"
constantType] "=" constExpression ";" | comment ;
    localVarsSection ::= "vars" [{{localVarsDefinition}}] ;
    localVarsDefinition ::= identifierList ":"
typeSpecifier ";" | comment ;
    instructions ::= comment | statementList |
methodOrFunctionCall [{{callArgument}}] [";" ;
    comment <TERMINAL> ::= "/" * ("*/") ~*/" |
commentCpp ;
    commentCpp ::= "/" ["\n\r"]+ ;
//*****statementList defined

below*****
methodOrFunctionCall ::= methodName [{"
[{{argList}} ")] | functionCall ;
    argList <TOKEN=NULL> ::= ["," arg [","
[arg] ;
    arg ::= argument | expression ;
    argument ::=
["exception"] [primitiveType] [methodOrFunctionCall] [literal] [expression] [{"&"
(methodOrFunctionCall | literal)}] [{{callArgument}}] ;
    callArgument ::=
[["."] methodName [{" [argList] ")}] ;
    functionCall ::= "call" identifier ;
    externalMethodSourcesSection ::= "externalMethodSources" [{{methodName
{" externalMethodSource "}}} ;
    externalMethodSource ::= externalMethodHeader ;

//*****
//*****
//
//
//
//
//*****
//*****

statementList ::= statement ";" [{{statement ";"}}] ;

```

```

statement ::= terminateStatement | transactionStatement | ioStatement |
ifStatement | whileStatement | foreachStatement | returnStatement | createStatement |
deleteStatement | breakOrContinueStatement | onExceptionStatement |
raiseExceptionStatement | assignmentStatement ;
terminateStatement ::= "terminate" ;
transactionStatement ::= "beginTransaction" | "commitTransaction" |
"abortTransaction" | "beginTransientTransaction" | "commitTransientTransaction" |
"beginLoad" | "endLoad" | "beginLock" | "endLock" ;
ioStatement ::= ("read" | "write") [arraylist] expression ;
breakOrContinueStatement ::= ("break" | "continue") [identifier] ;
returnStatement ::= "return" [booleanLiteral |
methodOrFunctionCall[{argument}]] ;
deleteStatement ::= "delete" [methodName] [{" " "} ] ;
whileStatement ::= "while" condition [{booleanOperator condition}] "do"
[":" identifier] [{"instructions}] "endwhile" [identifier] ;
condition ::= lhs [relationOperator rhs] ;
lhs ::= modifiedIdentifier[{callArgument}] [arraylist] |
Literal | methodOrFunctionCall[{callArgument}] ;
rhs ::= "null" | expression | modifiedIdentifier
[methodOrFunctionCall] | methodOrFunctionCall ;
foreachStatement ::= "foreach" identifier "in" [{callArgument}] [{"to"
expression} [{"step" expression} [{"reversed"} [{"where" expression} "do" ":"
identifier] [{"instructions}] "endforeach" [identifier] ;
createStatement ::= "create" identifier [{"as" expression}
[createOption] ;
createOption ::= "persistent" | "transient" | "sharedTransient"
;
onExceptionStatement ::= "on" expression "do" expression
[onExceptionOption | methodOrFunctionCall] ;
onExceptionOption ::= "global" ;
raiseExceptionStatement ::= "raise" expression [raiseExceptionOption]
;
raiseExceptionOption ::= "internal" | "precondition" ;
ifStatement ::= "if" condition [{booleanOperator condition}] "then"
[{"instructions}] [{"elseif" condition [{booleanOperator condition}] "then"
[{"instructions}]}] [{"else" [{"instructions}]}] "endif" ;
assignmentStatement ::= [arraylist] "!=" [booleanOperator]
[arraylist] [{"argument}][literal] ;
arrayList ::= [methodName] [{" "." modifiedIdentifier |
expression "\""} ] [{" "." modifiedIdentifier} ;
};

```

Appendix F: Sample application – Building Inheritance

The following source code has been extracted from Schildt's (2003, p. 280) classic text "C++ The Complete Reference", with only minor modifications. The modifications are made for brevity only, for example: class set and get methods were incorporated into the class declaration. The modifications did not include code commenting, as the application attribute names were considered self-explanatory as supplied.

```
#include <iostream>
using namespace std;

class Building {
private:
    int area;
    int rooms;
    int floors;
public:
    int get_area(){ return area; }
    int get_rooms(){ return rooms; }
    int get_floors(){ return floors; }
    void set_area(int value){ area = value; }
    void set_rooms(int value){ rooms = value; }
    void set_floors(int value){ floors = value; }
};

class House : public Building {
private:
    int bedrooms;
    int bathrooms;
public:
    int get_bedrooms(){ return bedrooms; }
    int get_bathrooms(){ return bathrooms; }
    void set_bedrooms(int value){ bedrooms = value; }
    void set_bathrooms(int value){ bathrooms = value; }
};

class School : public Building {
private:
    int offices;
    int classrooms;
public:
    int get_offices(){ return offices; }
    int get_classrooms(){ return classrooms; }
    void set_offices(int value){ offices = value; }
    void set_classrooms(int value){ classrooms = value; }
};

int main()
{
    House aHouse;
    School aSchool;

    aHouse.set_bathrooms(3);
    aHouse.set_bedrooms(5);
    aHouse.set_rooms(12);
    aHouse.set_floors(3);
    aHouse.set_area(500);

    aSchool.set_classrooms(200);
    aSchool.set_offices(10);
    aSchool.set_area(25000);
}
```



```
aSchool.set_floors(3);
aSchool.set_rooms(250);

cout << "The house has " << aHouse.get_bathrooms() << " bathrooms" << endl;
cout << "It also has " << aHouse.get_bedrooms() << " bedrooms" << endl;
cout << "It's area covers " << aHouse.get_area() << " units of area" << endl;
cout << "Over " << aHouse.get_floors() << " floors" << endl;

cout << "The school has " << aSchool.get_rooms() << " rooms " << endl;
cout << "covering " << aSchool.get_floors() << " floors, with a total " <<
endl;
cout << "of " << aSchool.get_area() << " units of area.\n" << endl;

return 0;
}
```

Appendix G: The generated JADE Building Inheritance schema file.

```
jadeVersionNumber "6.0.08";
schemaDefinition
TestSourceInheritanceSchema subschemaOf RootSchema partialDefinition, modelSchema;
constantDefinitions
    categoryDefinition Building
    categoryDefinition House
    categoryDefinition School
typeHeaders
    Building subclassOf Object transient;
    School subclassOf Building transient;
    House subclassOf Building transient;
    TestSourceInheritanceSchema subclassOf RootSchemaApp;
    GtestSourceInheritanceSchema subclassOf RootSchemaGlobal;
    StestSourceInheritanceSchema subclassOf RootSchemaSession;
typeDefinitions
    TestSourceInheritanceSchema completeDefinition
    (
        documentationText
        `This is the Application subclass.`
        jadeMethodDefinitions
            main();
    )
    GtestSourceInheritanceSchema completeDefinition
    (
        documentationText
        `This is the Global subclass.`
    )
    StestSourceInheritanceSchema completeDefinition
    (
        documentationText
        `This is the WebSession subclass.`
    )
    Building completeDefinition
    (
        attributeDefinitions
            area:          Integer protected;
            rooms:        Integer protected;
            floors:       Integer protected;
        jadeMethodDefinitions
            get_area() : Integer;
            get_rooms() : Integer;
            get_floors() : Integer;
            set_area(value : Integer);
            set_rooms(value : Integer);
            set_floors(value : Integer);
    )
    School completeDefinition
    (
        attributeDefinitions
            offices:      Integer protected;
            classrooms:   Integer protected;
        jadeMethodDefinitions
            get_offices() : Integer;
            get_classrooms() : Integer;
            set_offices(value : Integer);
            set_classrooms(value : Integer);
    )
    House completeDefinition
    (
        attributeDefinitions
            bedrooms:     Integer protected;
            bathrooms:    Integer protected;
        jadeMethodDefinitions
            get_bedrooms() : Integer;
            get_bathrooms() : Integer;
            set_bedrooms(value : Integer);
            set_bathrooms(value : Integer);
    )
    Building completeDefinition
```

```

    {
    jadeMethodDefinitions
        get_area() : Integer;
        get_rooms() : Integer;
        get_floors() : Integer;
        set_area(value : Integer);
        set_rooms(value : Integer);
        set_floors(value : Integer);
    }
    House CompleteDefinition
    {
    jadeMethodDefinitions
        get_bedrooms() : Integer;
        get_bathrooms() : Integer;
        set_bedrooms(value : Integer);
        set_bathrooms(value : Integer);
    }
    School CompleteDefinition
    {
    jadeMethodDefinitions
        get_offices() : Integer;
        get_classrooms() : Integer;
        set_offices(value : Integer);
        set_classrooms(value : Integer);
    }
    }
databaseDefinitions
    TestSourceInheritanceSchemaDb
    {
    databaseFileDefinitions
        "TestSourceInheritanceSchema";
    defaultFileDefinition "TestSourceInheritanceSchema";
    classMapDefinitions
        Building in "TestSourceInheritanceSchema";
        House in "TestSourceInheritanceSchema";
        School in "TestSourceInheritanceSchema";
        TestSourceInheritanceSchema in "_usergui";
        GtestSourceInheritanceSchema in "TestSourceInheritanceSchema";
        StestSourceInheritanceSchema in "TestSourceInheritanceSchema";
    }
    }

```

Appendix H: The converted Towers of Hanoi schema file

```
jadeVersionNumber "6.0.0B";
schemaDefinition
ConvertedTowersSample subschemaOf RootSchema partialDefinition, modelSchema;
constantDefinitions
    categoryDefinition ConvertedTOHmodified
    documentationText
`This is the Application subclass.`
    MAXDISKS : Integer = 4;
    categoryDefinition Tower
typeHeaders
    ConvertedTOHmodified subclassOf RootSchemaApp;
    GConvertedTOHmodified subclassOf RootSchemaGlobal;
    SConvertedTOHmodified subclassOf RootSchemaSession;
    Tower subclassOf Object transient;
typeDefinitions
    ConvertedTOHmodified completeDefinition
    {
        documentationText
`This is the Application subclass.`
        constantDefinitions
            MAXDISKS : Integer = 4;
        jadeMethodDefinitions
            move(
                from : Tower io;
                to_ : Tower io;
                use : Tower io;
                depth : Integer) updating;
            hanoi() updating;
            main() updating;
    }
    GConvertedTOHmodified completeDefinition
    {
        documentationText
`This is the Global subclass.`
    }
    SConvertedTOHmodified completeDefinition
    {
        documentationText
`This is the WebSession subclass.`
    }
    Tower completeDefinition
    {
        attributeDefinitions
            towerNumber: Integer protected;
            disks: IntegerArray protected;
            numDisks: Integer protected;
            i: Integer protected;
            temp: Integer protected;
        jadeMethodDefinitions
            tower(n : Integer) updating;
            addDisks() updating;
            pop() : Integer updating;
            push(I : Integer io) updating;
            print() updating;
            test() updating;
    }
    ConvertedTOHmodified completeDefinition
    {
        documentationText
`This is the Application subclass.`
        constantDefinitions
            MAXDISKS : Integer = 4;
        jadeMethodDefinitions
            move(
                from : Tower io;
                to_ : Tower io;
                use : Tower io;
                depth : Integer) updating;
            hanoi() updating;
```

```

        main() updating;
    }
Tower completeDefinition
(
jadeMethodDefinitions
    tower(n : Integer) updating;
    addDisks() updating;
    pop() : Integer updating;
    push(I : Integer io) updating;
    print() updating;
    test() updating;
)
databaseDefinitions
    ConvertedTowersSampleDb
    (
        databaseFileDefinitions
            "ConvertedTowersSample";
        defaultFileDefinition "ConvertedTowersSample";
        classMapDefinitions
            ConvertedTOHmodified in "_usergui";
            ConvertedTOHmodified in "ConvertedTowersSample";
            GConvertedTOHmodified in "ConvertedTowersSample";
            SConvertedTOHmodified in "ConvertedTowersSample";
    )
schemaViewDefinitions
_remapTableDefinitions
externalFunctionSources
typeSources
    ConvertedTOHmodified(
jadeMethodSources
move
(
move(

            from : Tower io;

            to_ : Tower io;

            use : Tower io;

            depth : Integer) updating;

vars
begin

    if depth =1 then
        from.print();
        to_.print();
        use.print();
    endif;

    if depth > 0 then
        move(from, use, to_, depth-1);
        to_.push(from.pop());
        move(use, to_, from, depth-1);
    endif;

    if depth =1 then
        from.print();
        to_.print();
        use.print();
    endif;

end;
)
hanoi
(
hanoi() updating;

vars
    a      : Tower;
    b      : Tower;
    c      : Tower;

begin

```

```

        create a transient;
        create b transient;
        create c transient;
        a.tower(1);
        b.tower(2);
        c.tower(3);

        a.addDisks();
        a.print();
        b.print();
        c.print();
        write "-----";
        move(a, b, c, MAXDISKS);
        write "-----";
        a.print();
        b.print();
        c.print();
end;
}

main
{
main() updating;

vars
    aTower : Tower;
begin

        create aTower transient;
        aTower.test();
        write "=====";
        hanoi();
end;
}

)

Tower(
jadeMethodSources

tower
{
    tower(n : Integer) updating;
vars
    I : Integer;
begin
        foreach I in 1 to MAXDISKS do
            disks[i]:=0;
        endforeach;
        numDisks:=0;
        towerNumber:=n;
end;
}

addDisks
{
addDisks() updating;

vars
    I : Integer;
begin
        foreach I in 1 to MAXDISKS do
            disks[i] := MAXDISKS - I;
        endforeach;
        numDisks:=MAXDISKS;
end;
}

pop
{
pop() : Integer updating;

vars
    temp : Integer;
begin
        if numDisks > 0 then
            temp := disks[numDisks-1];
        endif;

```

```

        disks[numDisks-1]:=0;
        numDisks := numDisks - 1;
        return temp;
    end;
}

push
{
push(I : Integer) updating;

vars

begin
    disks[numDisks] := I;
    numDisks := numDisks + 1;
end;
}

print
{
print();

vars
    I      : Integer;
begin
    write towerNumber.String & ":";
    foreach I in 1 to MAXDISKS do
        write disks[i].String & " ";
    endforeach;
    write " " & numDisks.String;
end;
}

test
{
test();

vars
    a      : Tower;
begin
    create a transient;
    a.tower(1);
    a.print();
    a.addDisks();
    a.print();
    write "pop " & a.pop().String;
    a.print();
    a.push(99);
    a.print();
end;
}

)

```

Appendix I: The converted Building Inheritance schema file

```
jadeVersionNumber "6.0.08";
schemaDefinition
TestSourceInheritanceSchema subschemaOf RootSchema partialDefinition, modelSchema;
constantDefinitions
    categoryDefinition Building
    categoryDefinition House
    categoryDefinition School
typeHeaders
    Building subclassOf Object transient;
    School subclassOf Building transient;
    House subclassOf Building transient;
    TestSourceInheritanceSchema subclassOf RootSchemaApp;
    GtestSourceInheritanceSchema subclassOf RootSchemaGlobal;
    StestSourceInheritanceSchema subclassOf RootSchemaSession;
typeDefinitions
    TestSourceInheritanceSchema completeDefinition
    {
        documentationText
        `This is the Application subclass.`
        jadeMethodDefinitions
            main();
        }
    GtestSourceInheritanceSchema completeDefinition
    {
        documentationText
        `This is the Global subclass.`
        }
    StestSourceInheritanceSchema completeDefinition
    {
        documentationText
        `This is the WebSession subclass.`
        }
    Building completeDefinition
    {
        attributeDefinitions
            area:          Integer protected;
            rooms:        Integer protected;
            floors:       Integer protected;
        jadeMethodDefinitions
            get_area() : Integer;
            get_rooms() : Integer;
            get_floors() : Integer;
            set_area(value : Integer);
            set_rooms(value : Integer);
            set_floors(value : Integer);
        }
    School completeDefinition
    {
        attributeDefinitions
            offices:      Integer protected;
            classrooms:   Integer protected;
        jadeMethodDefinitions
            get_offices() : Integer;
            get_classrooms() : Integer;
            set_offices(value : Integer);
            set_classrooms(value : Integer);
        }
    House completeDefinition
    {
        attributeDefinitions
            bedrooms:     Integer protected;
            bathrooms:    Integer protected;
        jadeMethodDefinitions
            get_bedrooms() : Integer;
            get_bathrooms() : Integer;
            set_bedrooms(value : Integer);
            set_bathrooms(value : Integer);
        }
    Building completeDefinition
```



```

    {
    jadeMethodDefinitions
        get_area() : Integer;
        get_rooms() : Integer;
        get_floors() : Integer;
        set_area(value : Integer);
        set_rooms(value : Integer);
        set_floors(value : Integer);
    }
    House completeDefinition
    {
    jadeMethodDefinitions
        get_bedrooms() : Integer;
        get_bathrooms() : Integer;
        set_bedrooms(value : Integer);
        set_bathrooms(value : Integer);
    }
    School completeDefinition
    {
    jadeMethodDefinitions
        get_offices() : Integer;
        get_classrooms() : Integer;
        set_offices(value : Integer);
        set_classrooms(value : Integer);
    }
    }
    databaseDefinitions
    TestSourceInheritanceSchemaDb
    {
    databaseFileDefinitions
        "TestSourceInheritanceSchema";
    defaultFileDefinition "TestSourceInheritanceSchema";
    classMapDefinitions
        Building in "TestSourceInheritanceSchema";
        House in "TestSourceInheritanceSchema";
        School in "TestSourceInheritanceSchema";
        TestSourceInheritanceSchema in "_usergui";
        GtestSourceInheritanceSchema in "TestSourceInheritanceSchema";
        StestSourceInheritanceSchema in "TestSourceInheritanceSchema";
    }
    }
    schemaViewDefinitions
    _remapTableDefinitions
    externalFunctionSources
    typeSources
        TestSourceInheritanceSchema {
        jadeMethodSources
    main
    {
    main();

    vars
        aHouse : House;
        aSchool : School;

    begin

        create aHouse transient;
        create aSchool transient;

        aHouse.set_bathrooms(3);
        aHouse.set_bedrooms(5);
        aHouse.set_rooms(12);
        aHouse.set_floors(3);
        aHouse.set_area(500);
        aSchool.set_classrooms(200);
        aSchool.set_offices(10);
        aSchool.set_area(25000);
        aSchool.set_floors(3);
        aSchool.set_rooms(250);
        write "The house has " & aHouse.get_bathrooms().String & " bathrooms";
        write "It also has " & aHouse.get_bedrooms().String & " bedrooms";
        write "It's area covers " & aHouse.get_area().String & " units of area";
        write "Over " & aHouse.get_floors().String & " floors";
        write "The school has " & aSchool.get_rooms().String & " rooms ";
        write "covering " & aSchool.get_floors().String & " floors, with a total ";
        write "of " & aSchool.get_area().String & " units of area.";

    end;
    }
}

```

```

)
    Building(
        jadeMethodSources

get_area
{
get_area() : Integer;

vars

begin
    return area;
end;
}

get_rooms
{
get_rooms() : Integer;

vars

begin
    return rooms;
end;
}

get_floors
{
get_floors() : Integer;

vars

begin
    return floors;
end;
}

set_area
{
set_area(value : Integer)updating;

vars

begin
    area := value;
end;
}

set_rooms
{
set_rooms(value : Integer)updating;

vars

begin
    rooms := value;
end;
}

set_floors
{
set_floors(value : Integer)updating;

vars

begin
    floors := value;
end;
}

```

```

)
    House(
        jadeMethodSources

get_bedrooms
{
get_bedrooms() : Integer;

vars

begin
    return bedrooms;
end;
}

get_bathrooms
{
get_bathrooms() : Integer;

vars

begin
    return bathrooms;
end;
}

set_bedrooms
{
set_bedrooms(value : Integer)updating;

vars

begin
    bedrooms := value;
end;
}

set_bathrooms
{
set_bathrooms(value : Integer)updating;

vars

begin
    bathrooms := value;
end;
}

)
    School(
        jadeMethodSources

get_offices
{
get_offices() : Integer;

vars

begin
    return offices;
end;
}

get_classrooms
{
get_classrooms() : Integer;

vars

begin

```

```
        return classrooms;
    end;
}

set_offices
{
    set_offices(value : Integer)updating;

    vars

    begin
        offices := value;
    end;
}

set_classrooms
{
    set_classrooms(value : Integer)updating;

    vars

    begin
        classrooms := value;
    end;
}

)
```

Appendix J: Glossary of terms

TERM	DESCRIPTION	SOURCE
Algorithm	A systematic problem-solving procedure, especially an established, recursive computational procedure for solving a problem in a finite number of steps.	(Howe, 2003a)
API	Application Programmer Interface: The interface (calling conventions) by which an application program accesses operating system and other services.	(Dictionary.com, 2003)
Application	A program that gives a computer instructions that provide the user with tools to accomplish a task.	(Dictionary.com, 2003)
Architecture	The manner in which the components of a computer or computer system are organised and integrated	(Merriam-Webster, 2003a)
Attribute	A quality or characteristic inherent in or ascribed to someone or something. A named value or relationship that exists for some or all instances of some entity and is directly associated with that instance.	(Howe, 2003b)
BPR	Business Process Re-engineering. An initiative to modify and improve the step-wise processes within an organisation.	(Maylor, 2003)
CASE	Computer Aided Software Engineering	
Class	A set of objects that share the same attributes, operations, relationships and semantics	(Booch et al., 1999)
Code bloat	Software growth without obvious benefit is the very definition of "code bloat."	(Langa, 2001)

TERM	DESCRIPTION	SOURCE
Construct	A 'type' for example: unsigned int; OR a 'statement', for example: condition statement, which may be considered a native structure in a programming language.	The author of this document
Converter	The tool used to perform the translation process	The author of this document
dll	dynamic link library: A library which is linked to application programs when they are loaded or run rather than as the final phase of compilation.	(Dictionary.com, 2003)
Forward engineer	Forward engineering is the process of moving from a high-level abstraction and logical implementation-independent design, to the physical implementation of that design.	(Chikofsky & Cross, 1990, p. 14)
Grammar	A mechanism used to describe the syntax of a language	(Sebesta, 1999)
GUI	Graphical User Interface: An interface for issuing commands to a computer utilizing a pointing device, such as a mouse, that manipulates and activates graphical images on a monitor.	(Dictionary.com, 2003)
HLPL	High-Level Programming Language	
HPS	High Productivity System	
HTML	Hyper-Text Mark-up Language: A markup language used to structure text and multimedia documents and to set up hypertext links between documents, used extensively on the World Wide Web.	(Dictionary.com, 2003)
IS	Information System: the network of all communication channels used within an organization	(Dictionary.com, 2003)
Legacy system	Any software application based on older technologies and hardware that may still provide core services to an organisation.	(Good, 2002)

TERM	DESCRIPTION	SOURCE
LOC	Lines Of Code	
MCC	McCabe's Cyclomatic Complexity	
MDA	Metamodel Driven Architecture	(OMG, 2003)
meta	A prefix meaning one level of description higher. If X is some concept then meta-X is data about, or processes operating on, X.	(Dictionary.com, 2003)
Metamodel	"A metamodel is in effect an abstract language for some kind of metadata".	(OMG, 2002, p. 15)
MOF	Meta_Object Facility	(OMG, 2002)
Method	In object-oriented programming, a method is a programmed procedure that is defined as part of a class and included in any object of that class. A class (and thus an object) can have more than one method. A method in an object can only have access to the data known to that object, which ensures data integrity among the set of objects in an application. A method can be re-used in multiple objects.	(TechTarget, 1999)
MFC	Microsoft Foundation Classes	
Monolithic system	Consisting of or constituting a single unit – relating to the development style used to implement a technical system, usually in an imperative language.	(Merriam-Webster, 2003b)
OMG	Object Management Group	
OO	Object Oriented: Of, related to, or being a language or system that can use and support objects	(Dictionary.com, 2003)
parse tree	A hierarchical, linked set of nodes representing the input stream.	(Aho et al., 2003)

TERM	DESCRIPTION	SOURCE
pdf	Portable Document Format: PDF is the file format for representing documents in a manner that is independent of the original application software, hardware, and operating system used to create those documents.	(Dictionary.com, 2003)
Reverse engineer	Reverse engineering is the process of analysing a subject system to: identify the system's components and their inter-relationships create representations of the system in another form or at a higher level of abstraction.	(Chikofsky & Cross, 1990, p. 15)
Rose/UML	Rational Rose implementation of the UML	
Simulated construct	A construct devised to simulate the properties or actions of a structure not otherwise available in a programming language.	The author of this document
syntactically correct	According to the rules of syntax. The structure rules.	The author of this document
Translate	In this context, to migrate the code in one programming language to another programming language, while essentially maintaining the same functionality.	The author of this document
Transliterate	To transcribe (a word, etc., in one alphabet) into corresponding letters of another alphabet.	(W. Collins, 1988)
UML	Unified Modelling Language: A non-proprietary, third generation modelling language. The Unified Modelling Language is an open method used to specify, visualise, construct and document the artefacts of an object-oriented software-intensive system under development.	(Dictionary.com, 2003)
VC++	Microsoft Visual C++	
XMI	XML Metadata Interchange	

TERM	DESCRIPTION	SOURCE
XML	eXtensible Mark-up Language: A metalanguage written in SGML that allows one to design a mark-up language, used to allow for the easy interchange of documents on the World Wide Web	(Dictionary.com, 2003)

REFERENCES

- Aho, A., Sethi, R., & Ullman, J. (2003). *Compilers: principles, techniques and tools*. New Jersey: Prentice Hall.
- Boggs, W., & Boggs, M. (2002). *Mastering UML with Rational Rose 2002*. Alameda, California: Sybex.
- Booch, G., Rumbaugh, J., & Jacobson, I. (1999). *The Unified Modeling Language user guide*. Upper Saddle River: Addison-Wesley.
- Chidamber, S.R., & Kemerer, C. F. (1991, October). Towards a metrics suite for object oriented design, In A. Paepcke, (ed.) Object oriented programming systems, languages and applications (OOPSLA'91). *SIGPLAN notices*, 26(11), 197-211.
- Chikofsky, E., & Cross, J. (1990). Reverse engineering and design recovery: a taxonomy. *IEEE Software*, 7(1), 13, 17.
- Church, J. *StoryBook*. [Computer software]. (2003). Perth, Western Australia: Edith Cowan University.
- Collins, M. (1993). *An Ada-like language to facilitate reliable coding of low cost embedded systems*. Unpublished thesis, Edith Cowan University, Perth.
- Collins English dictionary*. (1988). London: William Collins & Sons.
- Cowley, S. (2003). *Vendors convene to examine legacy apps modernization*. Retrieved April 17, 2003 from <http://www.computerworld.com.au/pp.php?id=991088706&taxid=975794159>
- Demeyer, S., Rieger, M., & Tichelaar, S. (1998). *Three reverse engineering patterns*. Retrieved November 12, 2003 from <http://www.iam.unibe.ch/~famoos/Deme98p/threerevpat.2pgup.pdf>
- Dictionary.com. (2003). *On-line dictionary*. Retrieved November 13, 2003 from <http://dictionary.reference.com/>
- Ducasse, S. E. (2001). *Reengineering object-oriented applications*. Unpublished thesis, Universite Pierre et Marie Curie, Paris.
- Good, D. (2002). *Legacy transformation*. San Jose: Technology Research Club.
- Gutschmidt, T. (2003). *Open source high-level languages in your neighborhood*. Retrieved November 11, 2003 from <http://www.developer.com/lang/other/article.php/1581881>

- Harsu, M. (2000). *Re-engineering legacy software through language conversion*. Unpublished thesis, University of Tampere, Dept. of Computer Science, Tampere, Finland.
- Hill, S. (1995). *Towers of Hanoi*. Retrieved September 15, 2003 from <http://www.ecs.umass.edu/ece/hill/ece242.dir/hanoi.c>
- Howe, D. (2003a). *The free on-line dictionary of computing. [algorithm]*. Retrieved July 1, 2003 from <http://dictionary.reference.com/search?q=algorithm>
- Howe, D. (2003b). *The free on-line dictionary of computing. [attribute]*. Retrieved July 1, 2003 from <http://dictionary.reference.com/search?q=attribute>
- JADE. (2003). *JADE developer's reference*. Retrieved November 13, 2003 from <http://www.jadeworld.com/downloads/jade6manuals/devref.pdf>
- JADE online help*. [Computer software]. (2001). Christchurch, NZ: Jade Software Corporation Ltd.
- Kazman, R., O'Brien, L., & Verhoef, C. (2002). *Architecture reconstruction guidelines*. (2nd ed.). (ESC-TR-2002-034). Pittsburgh: Carnegie Mellon University, Software Engineering Institute.
- Kontogiannis, K., Martin, J., Wong, K., Gregory, R., Muller, H., & Mylopoulos, J. (1998). Code migration through transformations: an experience report. *CASCON-98 IBM Conference*, Toronto, Ontario, November 30-December 3, 1998. Retrieved on November 12, 2003 from <http://www.notamusica.de/home/jmartin/resume/cascon98.pdf>
- Krishnamoorthy, S. (2003). *RE: [SR#167559387] -- How can I get Rose 2001A to capture and retain functional code*. Personal communication, March 10, 2003.
- Langa, F. (2001). *Rethinking "Software bloat"*. Retrieved July 9, 2003 from <http://www.informationweek.com/story/IWK20011212S0003>
- Levine, J., Mason, T., & Brown, D. (1995). *Lex & Yacc*. Cambridge: O'Reilly.
- Liberty, J. (2001). *Teach yourself C++ in 21 days*. (4th ed.). Indianapolis: Sams Publishing.
- Markosian, L., Newcomb, P., Brand, R., Burson, S., & Kitzmiller, T. (1994). Using an enabling technology to reengineer legacy systems. *Association for Computer Machinery. Communications of the ACM*, 37(5), 58-70.
- Maylor, H. (2003). *Project management*. Essex: Pearson Education.
- McCabe, T., & Butler, C. (1989). Design complexity measurement and testing. *Association for Computer Machinery. Communications of the ACM*, 32(12), 1415-1425.

- Merriam-Webster. (2003a). Merriam-Webster online dictionary - architecture. Retrieved July 10, 2003 from <http://www.m-w.com/cgi-bin/dictionary?book=Dictionary&va=architecture>
- Merriam-Webster. (2003b). Merriam-Webster online dictionary - monolithic. Retrieved July 9, 2003 from <http://www.m-w.com/cgi-bin/dictionary?book=Dictionary&va=monolithic>
- Moynihan, V., & Wallis, P. (1991). The design and implementation of a high-level language converter. *Software - Practice and Experience*, 21(4), 391-400.
- NorKen. (2003). *ProGrammar - parser development toolkit*. Retrieved April 24, 2003 from <http://www.programmar.com/grammars.htm>
- Object Management Group. (2002). *Meta Object Facility (MOF) specification*. Retrieved November 13, 2003 from <http://www.omg.org/cgi-bin/apps/doc?formal/02-04-03.pdf>
- Object Management Group. (2003). *Why do we need legacy transformation standards?* Retrieved November 12, 2003 from http://www.omg.org/registration/Legacy_Transformation-whitepaper_06.pdf
- O'Sullivan, J. (2000). JADE in action. *Offline*, 34(3), 6-8.
- Pressman, R. (2001). *Software engineering: a practitioners approach* (5th ed.). New York: McGraw-Hill.
- Quatrani, T. (2000). *Visual modeling with Rational Rose 2000 and UML*. Upper Saddle River: Addison-Wesley.
- Rational Rose help*. [Computer software]. (2001). Rational Software Corporation.
- Roeder, C. (2003). *The Towers of Hanoi in Three Styles of C, and C++*. Retrieved October 21, 2003 from <http://www.croeder.com/notes/hanoi2.cpp>
- Schildt, H. (2003). *The complete reference C++*. (4th ed.). Berkeley, California: McGraw-Hill Osborne.
- Seacord, R., Comella-Dorda, S., Lewis, G., Place, P., & Plakosh, D. (2001). *Legacy system modernization strategies*. Pittsburgh, Pa.: Carnegie Mellon University, Software Engineering Institute.
- Seacord, R. C., Plakosh, D., & Lewis, G. A. (2003). *Modernizing legacy systems : software technologies, engineering processes, and business practices*. Reading, Mass. : Addison-Wesley.
- Sebesta, R. (1999). *Concepts of programming language*. (4th ed.). Reading, Mass: Addison Wesley Longman.
- Skarmstad, T., Khan, K., & Rashid, A. (1999). Constructing commercial off-the-shelf from legacy systems: a conceptual framework. *Proceedings of the 10th Australasian Conference on Information Systems, 1999*, pp. 798-805.

- [Electronic version]. Retrieved October 30, 2003 from <http://www.vuw.ac.nz/acis99/Papers/PaperSkramstad-092.pdf>
- Suh, E., & Allain, A. (2003). *Code journal*. Retrieved September 15, 2003 from <http://www.cprogramming.com/codej/issue3.html>
- Sultanoglu, S. (1998). *Complexity metrics and models*. Retrieved October 24, 2003, from <http://yunus.hun.edu.tr/~sencer/complexity.html>
- TechTarget. (1999). *Whatis.com tech search*. Retrieved July 1, 2003 from http://whatis.techtarget.com/definition/0,,sid9_gci212559,00.html
- Terekhov, A.A. (2001). Automating language conversion: a case study. *Proceedings of the IEEE International Conference on Software Maintenance*, Florence, Italy, 7-9 November 2001. pp. 654-658.
- Terekhov, A., & Verhoef, C. (2000). The realities of language conversions. *IEEE Software*, 17(6), 111-124.
- Tieman, P. (2001). *Cyclomatic complexity metric*. Retrieved November 13, 2003 from http://www.delphifaq.com/software/sc_help/cyclomatic.htm
- Verbruggen, R. (2003). *Depth of inheritance tree*. Retrieved September 17, 2003 from <http://www.compapp.dcu.ie/~renaat/ca421/OOmetrics.ppt>
- Waters, R. (1988). Program translation via abstraction and reimplementaion. *IEEE Transactions on Software Engineering*, 14(8), 1207-1228.
- Watson, A. H., & McCabe, T. J. (1996). *Structured testing: a testing methodology using the cyclomatic complexity metric*. Retrieved October 24, 2003 from <http://www.mccabe.com/nist/chapter2.php#446018>
- Weiderman, N. H., Bergey, J., Smith, D., & Tilley, S. (1997). *Approaches to legacy system evolution*. Pittsburgh, Pa.: Carnegie Mellon University, Software Engineering Institute.
- White, D., Scribner, K., & Olafsen, E. (1999). *MFC programming with Visual C++ 6*. Washington: Sams Publishing.
- Wikipedia. (2003). *Cfront*. Retrieved November 11, 2003 from <http://en2.wikipedia.org/wiki/Cfront>