AFRL-IF-WP-TR-1999-1505

LEGACY SOFTWARE RE-ENGINEERING TECHNOLOGY (LSRET)

PETER G. CLARK
JOHN A. GILL

TASC
55 WALKERS BROOK DRIVE
READING, MASSACHUSETTS 01867

APRIL 1998

FINAL REPORT FOR PERIOD OF 20 MARCH 1996 – 20 FEBRUARY 1998

Approved for public release; distribution unlimited.

20010824 013

INFORMATION DIRECTORATE
AIR FORCE RESEARCH LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7334

NOTICE

USING GOVERNMENT DRAWINGS, SPECIFICATIONS, OR OTHER DATA INCLUDED IN THIS DOCUMENT FOR ANY PURPOSE OTHER THAN GOVERNMENT PROCUREMENT DOES NOT IN ANY WAY OBLIGATE THE US GOVERNMENT. THE FACT THAT THE GOVERNMENT FORMULATED OR SUPPLIED THE DRAWINGS, SPECIFICATIONS, OR OTHER DATA DOES NOT LICENSE THE HOLDER OR ANY OTHER PERSON OR CORPORATION; OR CONVEY ANY RIGHTS OR PERMISSION TO MANUFACTURE, USE, OR SELL ANY PATENTED INVENTION THAT MAY RELATE TO THEM.

THIS REPORT IS RELEASABLE TO THE NATIONAL TECHNICAL INFORMATION SERVICE (NTIS). AT NTIS, IT WILL BE AVAILABLE TO THE GENERAL PUBLIC, INCLUDING FOREIGN NATIONS.

THIS TECHNICAL REPORT HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION.

KENNETH LITTLEJOHN
Project Engineer

JAMES S. WILLIAMSON, Chief
Embedded Info Sys Engineering Branch
Information Technology Division
Information Directorate

WALTER B. HARTMAN
Acting Wright Site Coordinator
Information Directorate

Do not return copies of this report unless contractual obligations or notice on a specific document requires its return.

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	APRIL 1998	FINAL REPORT FOR 03/20/1996 - 02/20/1998

4. TITLE AND SUBTITLE
LEGACY SOFTWARE RE-ENGINEERING TECHNOLOGY (LSRET)

5. FUNDING NUMBERS
C F33615-92-D-1052
PE 78611
PR 3090
TA 01
WU 14

6. AUTHOR(S)
Peter G. Clark
John A. Gill

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
TASC
55 WALKERS BROOK DRIVE
READING, MASSACHUSETTS 01867

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
INFORMATION DIRECTORATE
AIR FORCE RESEARCH LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT-PATTERSON AFB, OH 45433-7334
POC: KENNETH LITTLEJOHN, AFRL/IFTA, 937-255-6548 EXT. 3587

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-IF-WP-TR-1999-1505

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE, DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This Final Report documents the state of the Legacy Software Reengineering Technology (LSRET) Prototype Project at the time it was halted. The technology developed by TASC under the LSRET program was to be used to assist in the reengineering of avionics software. The RET was designed to process FORTRAN and JOVIAL code, and generate Ada code. The RET was also designed to be extensible to other High Order Languages.

Much of the existing avionics software is poorly documented for maintenance purposes. LSRET was being developed to support engineers in transforming the legacy code into a more modern programming language and also to be used to (re)document either the legacy or the reengineered systems. This would improve the maintainability of the avionics software in two ways. LSRET could reengineer the software into a language that promotes better software engineering practices; and also makes it easier to find engineers who are trained in the use of the newer language. Second, LSRET could redocument the legacy and reengineered systems making them easier to maintain in the future.

14. SUBJECT TERMS	15. NUMBER OF PAGES	16. PRICE CODE
	38	

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR

Standard Form 298 (Rev. 2-89) (EG)
Prescribed by ANSI Std. 239.18
Designed using Perform Pro, WHS/DIOR, Oct 94

TABLE OF CONTENTS

Page

1. SCOPE 1-1

1.1 IDENTIFICATION 1-1
1.2 SYSTEM OVERVIEW 1-1
1.3 DOCUMENT OVERVIEW 1-1

2. REFERENCED DOCUMENTS 2-1

2.1 GOVERNMENT DOCUMENTS 2-1
2.2 NON-GOVERNMENT DOCUMENTS 2-1

3. TECHNICAL STATUS SUMMARY 3-1

3.1 SOFTWARE REENGINEERING STUDY 3-1
3.2 TECHNOLOGY TRANSITION 3-1
3.3 MULTIPLE LANGUAGE REENGINEERING PROCESS MODEL 3-3
3.4 EXISTING FORTRAN REENGINEERING TECHNOLOGY 3-4
3.5 LSRET PROTOTYPE 3-4
3.6 TESTING AND EVALUATION OF THE REENGINEERING TECHNOLOGY 3-5

4. SOFTWARE ASSETS DESCRIPTION 4-1

4.1 REASONING SYSTEMS SOFTWARE REFINERY 4-1
4.2 CUSTOM-DEVELOPED SOFTWARE 4-1
4.2.1 The ada Directory 4-2
4.2.2 The lsret Directory 4-2
4.2.3 The analyze Subdirectory 4-3
4.2.4 The casu Subdirectory 4-4
4.2.5 The casu-f Subdirectory 4-4
4.2.6 The cd Subdirectory 4-5
4.2.7 The ded Subdirectory 4-5
4.2.8 The dfd Subdirectory 4-6
4.2.9 The global Subdirectory 4-6
4.2.10 The hid Subdirectory 4-7
4.2.11 The jovial Subdirectory 4-8
4.2.12 The pak Subdirectory 4-8
4.2.13 The redoc-lite Subdirectory 4-9
4.2.14 The src Subdirectory 4-9
4.2.15 The uid Subdirectory 4-10
4.2.16 The refine Directory 4-10
4.2.17 The ret Directory 4-11
4.2.18 The spag Directory 4-14
4.3 EXTRACTING AND RUNNING THE PROTOTYPE 4-14
4.3.1 Extracting the RET from Tape 4-14
4.3.2 Starting and Stopping the RET 4-15

5. CONCLUSIONS 5-1

LIST OF FIGURES

Figure		Page
3-1	LSRET PROJECT PLANNING CHART	3-2
3-2	DIAGRAM OF THE LSRET PROCESS MODEL	3-4
4-1	LSRET SOFTWARE DIRECTORY STRUCTURE	4-2

LIST OF TABLES

Table		Page
4-1	CONTENTS OF THE LSRET DIRECTORY	4-3
4-2	CONTENTS OF THE CASU SUBDIRECTORY	4-4
4-3	CONTENTS OF THE CASU-F SUBDIRECTORY	4-5
4-4	CONTENTS OF THE CD SUBDIRECTORY	4-5
4-5	CONTENTS OF THE DED SUBDIRECTORY	4-6
4-6	CONTENTS OF THE DFD SUBDIRECTORY	4-7
4-7	CONTENTS OF THE GLOBAL SUBDIRECTORY	4-7
4-8	CONTENTS OF THE HID SUBDIRECTORY	4-8
4-9	CONTENTS OF THE JOVIAL-PARSER SUBDIRECTORY	4-8
4-10	CONTENTS OF THE PAK SUBDIRECTORY	4-9
4-11	CONTENTS OF THE SRC SUBDIRECTORY	4-10
4-12	CONTENTS OF THE UID SUBDIRECTORY	4-10
4-13	CONTENTS OF THE REFINE DIRECTORY	4-11
4-14	CONTENTS OF THE RET DIRECTORY	4-12

1. SCOPE

1.1 IDENTIFICATION

This Final Report (FR) documents the state of the Legacy Software Reengineering Technology (LSRET) Prototype Project at the time it was halted.

1.2 SYSTEM OVERVIEW

The purpose of the RET Prototype System is to support Wright Laboratory (WL) and other software maintenance personnel who are responsible for reengineering legacy software in translating programs written in FORTRAN and JOVIAL into Ada. The RET planned to recognize FORTRAN 77 [1] and the J73 dialect of JOVIAL [2], and generate Ada code compatible with the Ada 83 dialect [3]. The RET will be designed to satisfy this specific objective and be extensible to other High Order Languages (HOLs).

From 1992 to 1995 we developed and implemented a RET capable of transforming FORTRAN to Ada under the Avionics Software Reengineering Technology (ASRET) project. The ASRET project included a software reengineering study, reengineering process model development, reengineering technology development, and testing and evaluation activities [4].

Under ASRET, we investigated existing reengineering and reverse engineering processes, techniques, and software tools. Based on this study, we developed a process model and environment for reengineering software from one language (FORTRAN) to another (Ada). The approach is to engineer an Ada program by reusing portions of the original FORTRAN design and implementation. We designed and implemented a RET prototype to assist the engineer in this process.

Under LSRET, the prototype executes on a Sun Sparc 10 workstation under Solaris at Wright Laboratory. We developed the RET on a Sparc 20 at TASC's Reading, Massachusetts facility.

1.3 DOCUMENT OVERVIEW

This report contains the status of the project and the entire RET Prototype System and is divided into the following sections:

- Section 1 — Scope
- Section 2 — Referenced Documents
- Section 3 — Technical Status Summary
- Section 4 — Software Assets Description
- Section 5 — Conclusions

Section 1 describes the purpose and structure of the RET Prototype System and the purpose of the FR. Section 2 identifies the Government and non-Government documents that are referenced in and used in the preparation of this SDD. Section 3 describes the status of all of the technical activities within the Project at the time it was halted. Section 4 describes the configuration and state of the commercial off-the-shelf (COTS) and custom-developed software assets acquired and built for the RET, again at the time the project was halted. Section 5 provides some brief concluding remarks regarding the potential for this type of tool had it been allowed to proceed to its planned completion.

2. REFERENCED DOCUMENTS

2.1 GOVERNMENT DOCUMENTS

1. MIL-STD-1753, FORTRAN DoD Supplement to ANSI X3.9-1978, 9 November 1978.
2. MIL-STD-1589C, JOVIAL (J73), 6 July 1984.
3. ANSI/MIL-STD-1815A, Ada Programming Language, 22 January 1983.
4. Avionics Software Reengineering Technology (ASRET) Project Final Report Volume I, Project Summary, Account, and Results, Technical Report WL-TR-95-1119, Avionics Directorate, Wright Laboratory, Air Force Systems Command, Wright Patterson AFB, Ohio, May 1995.
5. Avionics Software Reengineering Technology (ASRET) Project, Reengineering Tool (RET) User's Manual, Technical Report WL-TR-95-1118, Avionics Directorate, Wright Laboratory, Air Force Systems Command, Wright Patterson AFB, Ohio, May 1995.

2.2 NON-GOVERNMENT DOCUMENTS

6. Legacy Software Reengineering Technology (LSRET) Software Reengineering Study Report, Technical Report TR-08290-12-3, TASC, Reading, Massachusetts, 15 November 1997.
7. P.G. Clark, et al., Automated Reengineering for Legacy Weapon System Software, Proceedings of the 16th AIAA/IEEE Digital Avionics Systems Conference, 26-30 October 1997, Vol. I, pp. 1.1-35 - 1.1-42.
8. Software Requirements Specification for the Legacy Software Reengineering Technology (RET) Prototype System RET-SRS-01, Revision 1, Technical Report TR-08290-12-1a, TASC, Reading, Massachusetts, 15 August 1997.
9. Software Design Description for the Legacy Software Reengineering Technology (RET) Prototype System RET-SDD-01, Technical Report TR-08290-12-2, TASC, Reading, Massachusetts, 30 June 1997.
10. Software Refinery Training Manual, Reasoning Systems, Revised, June 1993.
11. GNU Emacs Manual, Seventh Edition, Version 19, Free Software Foundation, September 1992.

12.LSRET Source & Executables (Current & New RET), 8 mm data tape, TASC, Reading, Massachusetts, 13 February 1998.

3. TECHNICAL STATUS SUMMARY

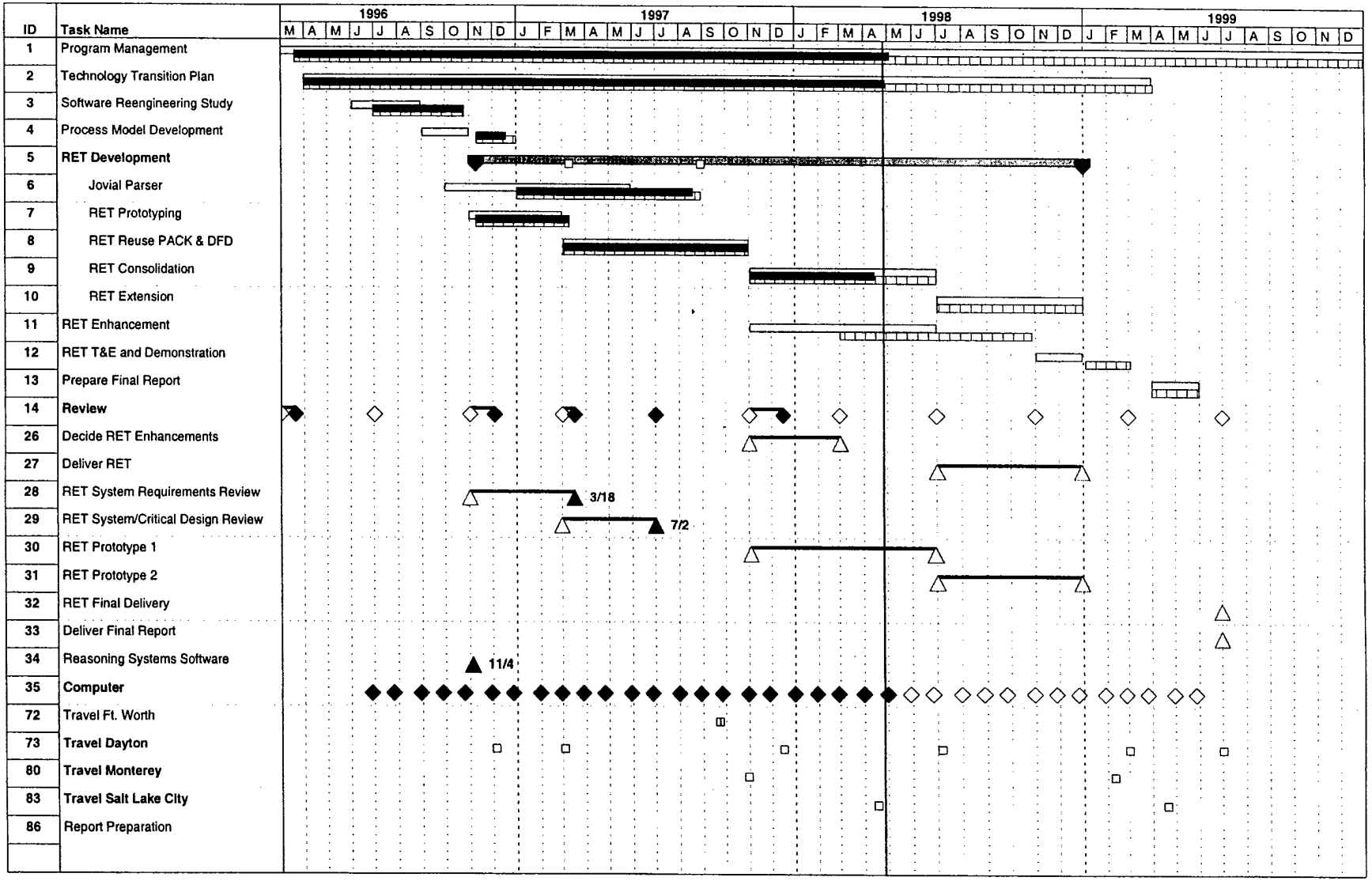
This section provides a status summary of the major technical tasks within the project at the time the work was halted. Figure 3-1 shows the planned schedule for the project and where we were on the schedule at termination.

3.1 SOFTWARE REENGINEERING STUDY

The Software Reengineering Study for LSRET had been completed and the draft was submitted to the Air Force [6]. The Study comprised a tool review, literature review, and a number of software reengineering experiments aimed at identifying better diagrams to enhance program understanding. The Study concluded that the Control Data Call Diagram and Control Dataflow Diagram, together with the theory on partitioning legacy embedded avionics software systems and the analysis process that was used, constitute a powerful and novel technology applicable to avionics software reengineering. TASC was awaiting Air Force comments before finalizing the Study.

3.2 TECHNOLOGY TRANSITION

This activity was to be ongoing throughout the duration of the project. It consisted of identifying potential technology transition candidates by whatever means we could and then contacting the candidates to arrange presentations and demonstrations with the idea of interesting them in using the LSRET Prototype when it became available.



CLIN 0001
Seq. No. 26
30 April 1998

To identify technology transition candidates we used a variety of methods that were both structured and unstructured. Structured methods included writing technical journal papers and making presentations and demonstrations at conferences. We had recently presented a paper at the Digital Avionics Systems Conference (DASC) [7] at which we spoke to a number of potentially interested candidates. We also used less structured approaches such as just calling technical contacts that were known through previous work relations and by piggybacking on another related project, Advanced Avionics Verification and Validation (AAV&V). We had made a presentation to the technical staff of three projects at Lockheed Martin in Ft. Worth, Texas at the direction of our Air Force customer. At the time LSRET was stopped, we were in the process of putting together a demonstration to show to the technical staff at Lockheed Martin and at the Software Technology Conference put on by Hill Air Force Base.

3.3 MULTIPLE LANGUAGE REENGINEERING PROCESS MODEL

The Process Model was something that was to be included with the user documentation delivered with LSRET at the end of the project. A high-level concept for the Process Model was beginning to emerge with the development of the demonstration. The current concept for the Process Model is best represented by the diagram shown in Figure 3-2. The process begins with the user parsing and analyzing the legacy source code to produce an abstract software representation. Then they would iterate between the software views to (re)document the legacy and target systems and update the software representation by redesigning and restructuring the legacy representation into the target representation. When the user is happy with the target software representation and target software views, they can generate the target source code. The Process Model would have been refined throughout the testing and evaluation of the LSRET Prototype.

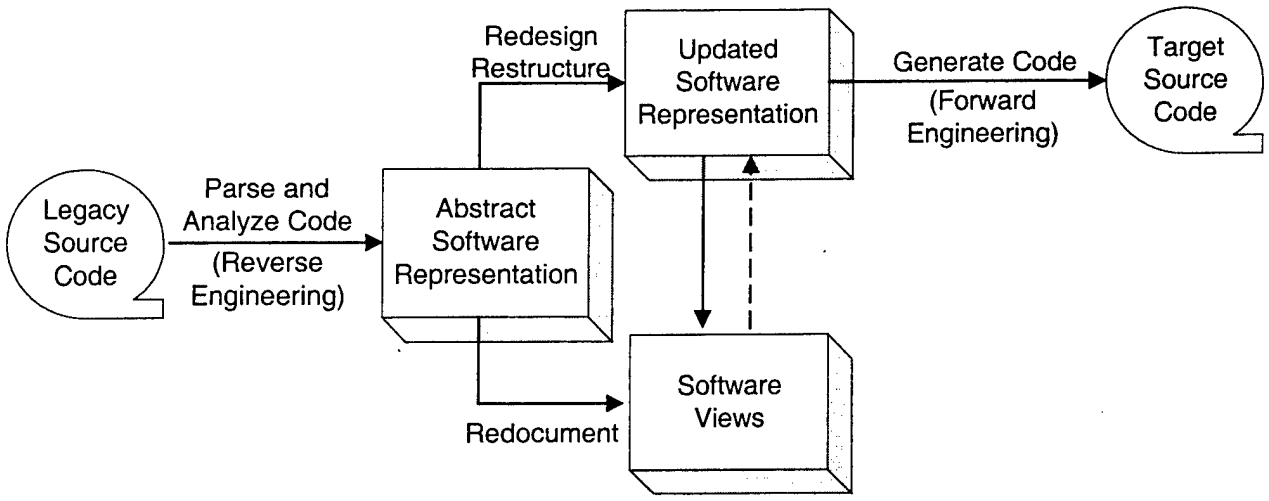


Figure 3-2 Diagram of the LSRET Process Model

3.4 EXISTING FORTRAN REENGINEERING TECHNOLOGY

The existing FORTRAN Reengineering Technology [4] has been enhanced so that the redocumentation features are implemented using the language-independent approach designed for LSRET. This was done in preparing an LSRET demonstration while waiting for the JOVIAL parser to be completed and integrated. Some of the transformation of the features of the old ASRET tool, such as FORTRAN arrays and input/output statements, were also identified for enhancement, but were put at a lower priority and thus were scheduled to be completed later in the contract.

3.5 LSRET PROTOTYPE

The LSRET Prototype was in the middle of the implementation phase when the project was stopped. The LSRET requirements had been drafted and revised [8] and the design had also been drafted [9]. Most of the redocumentation requirements (SDD [9] Section 5.6) had been implemented and unit-tested (see Section 4.2.13).

A JOVIAL parser/analyzer (SDD [9] Section 5.3) covering a substantial portion of the JOVIAL language [2] was completed and tested in collaboration with the AAV&V using the IRIS toolset. We were just about to start the integration of the

JOVIAL parser/analyzer into the LSRET Prototype when the project was stopped. The integration of the JOVIAL parser/analyzer into the LSRET Prototype entails building the abstract syntax graph and type graph domain models for the Refine Object Base (SDD [9] Sections 5.7.1 and 5.7.2) and then translating the outputs of the IRIS JOVIAL parser/analyzer to populate those domain models.

Much of the remaining implementation work is in the area of the transformation requirements (SDD [9] Section 5.4). Work on the language-independent version of the Packager was well along, but had not been tested with any JOVIAL code. Most of the remaining pieces of the JOVIAL version of the Transformer had not yet been implemented. The Code Generator module of the Transformer is commercial off-the-shelf (COTS) software being a part of the Refine/Ada tool that was being used and thus is already integrated into the LSRET Prototype.

3.6 TESTING AND EVALUATION OF THE REENGINEERING TECHNOLOGY

The task of testing and evaluation of the Reengineering Technology was scheduled as a later activity of the project to take place after the completion of a working LSRET Prototype (see Section 3.4). The plan was for the LSRET Prototype to be delivered to one or more technology transition partners (see Section 3.2) for trial use and evaluation.

4. SOFTWARE ASSETS DESCRIPTION

This section describes the software assets and their use and location of the RET Project at the time it was halted.

4.1 REASONING SYSTEMS SOFTWARE REFINERY

The LSRET Prototype is developed on top of the COTS tool suite called the Software Refinery [10], or Refine for short, sold by a company named Reasoning Systems. The tools that comprise the Software Refinery support software reengineering by providing various language parsing, analysis, and code generation capabilities. The following Software Refinery tools will be found in the directory /fs1/reasoning/ on the Air Force's lab computer and were purchased and used as part of the LSRET Prototype development:

- **Dialect** – language and tools used to build parsers, code generators, and pattern matchers for software languages from specifications
- **Intervista** – X-11 based toolkit for building windows-oriented user interfaces for Refine-based applications
- **Refine** – interactive programming environment providing object management and a high-level query/update language
- **Refine/Ada** – Ada language parser, analyzer, and code generator
- **Refine/FORTRAN** – FORTRAN language parser, analyzer, and code generator
- **Rerun** – Runtime environment for using Refine-based applications
- **Workbench** – application-programming interface (API) to the language-independent data structures and functions for building software reverse engineering and reengineering tools.

4.2 CUSTOM-DEVELOPED SOFTWARE

All of the custom-developed software for the RET was maintained on TASC's development computer in the /home/jagill/ directory as shown in Figure 4-1. The contents of each (sub)directory is described in this Section.

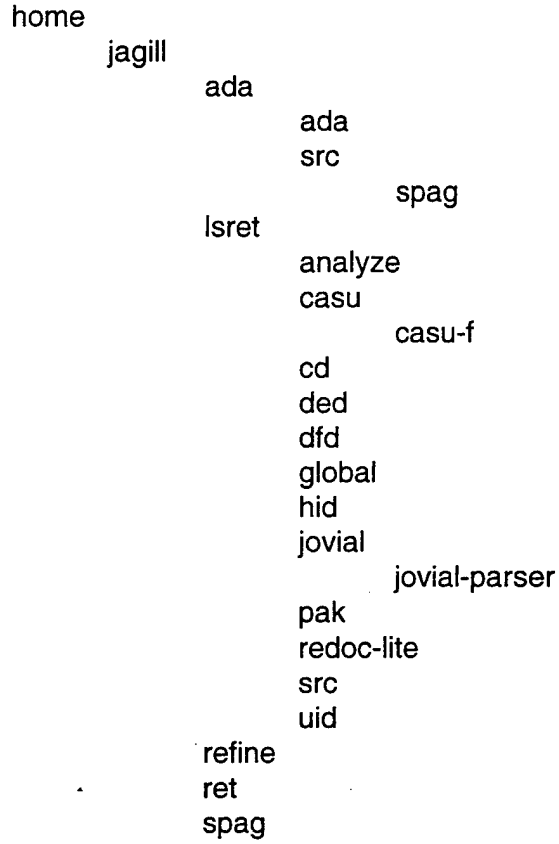


Figure 4-1 LSRET Software Directory Structure

4.2.1 The ada Directory

The ada directory contains the inputs and outputs for the old RET. The ada directory contains two subdirectories named ada and src. The ada subdirectory contains the Ada code generated from old RET using the Block 40 FORTRAN code as input. The src subdirectory contains the original Block 40 FORTRAN code given to TASC by the Air Force under the ASRET project. The src subdirectory also contains the subdirectory named spag. The spag subdirectory contains the structured Block 40 FORTRAN code generated using the SPAG on the original Block 40 FORTRAN code.

4.2.2 The lsret Directory

The LSRET directory contains the source and executables for the new LSRET Prototype. At this time, sessions can be run either in FORTRAN-to-Ada mode (currently loaded in the analyze directory) or in language-independent mode

(loaded in the redoc-lite directory). In the future, systems similar to the one loaded in the analyze directory would have been created for other source languages. Similarly, taking the Call and Set/Use (CASU) into a target language would have been accomplished using a similar RET session. Between the parsing and writing, the RET can be utilized in a language-independent fashion.

Table 4-1 Contents of the lsret Directory

FILENAME/DIRECTORY	DESCRIPTION
analyze	Loads a RET session which includes source Analysis and CASU production capabilities
casu	Call and Set/Use (CASU) package
cd	Call Diagram (CD) package
ded	Declaration Diagram (DED) package
dfd	Data Flow Diagram (DFD) package
global	RET global identifier package
hid	Hierarchical Interactive Display (HID) package
jovial	JOVIAL parser/analyzer provided by AAV&V
make-pkgs.re	Creates Refine packages for RET sessions
pak	Packager (PAK) package
rcs-header.re	Revision Control System (RCS) header for new source
redoc-lite	Loads a RET session in language-independent mode, allows Redocumentation and Packaging capabilities
refine-commands.re	Provides several Refine and Lisp functions that assist during a RET session
src	Source Code Listing (SRC) package
uid	User Interface (UID) package

4.2.3 The analyze Subdirectory

This directory, analyze, contains a single file, system.lisp, that initializes a RET session that enables Source Code Processing of FORTRAN code. This is a language-dependent RET session (Refine/FORTRAN and Refine/Ada is present).

The FORTRAN AST produced during analysis is present. A CASU Domain Model (DM) can be produced, and saved for later use. The system.lisp calls many other system.lisp files (the "make-system" command). These components are loaded in a specific order.

4.2.4 The casu Subdirectory

The CASU is a language-independent Domain Model Abstract Syntax Tree (AST) which represents a subset of an AST developed under analysis using the parser and semantic analyzer portions of the Source Code Processor (SCP).

CASU contains objects representing program entities that are used to develop Redocumentation views (DFD, DED, CD) and the initial Packager Diagram view. Because of its language-independence, the Redocumentation and Packager capabilities do not need to be reimplemented for each new source language that LSRET supports. Instead, a language-dependent extractor can pull out the necessary information and create a language-independent counterpart. This should make LSRET more extensible with only one function required for each new language added to LSRET.

At this time, FORTRAN is the only language that is extracted into CASU. Future plans included developing an Ada and JOVIAL extractor.

Table 4-2 Contents of the casu Subdirectory

FILENAME	DESCRIPTION
casu-intrinsic-map.re	Maps FORTRAN intrinsic functions to language-independent CASU counterparts
dm-casu.re	Call and Set/Use (CASU) Domain Model
export.lisp	CASU identifiers exported for use in other packages that "use" CASU package
system.lisp	Loads RET files
utility.lisp	General RET utilities

4.2.5 The casu-f Subdirectory

This package, specifically extract-casu-f, extracts the information from a FORTRAN AST, and creates a language-independent CASU domain model. Similar extractors would be created for other languages to be processed (both legacy and target languages).

This extraction takes place during a language-dependent RET session. The CASU DM can be saved as a Permanent Object Base (POB), and reloaded without reloading the FORTRAN AST or reloading this package.

Table 4-3 Contents of the casu-f Subdirectory

FILENAME	DESCRIPTION
dm-casu-f.re	Call and Set/Use Extractor FORTRAN-specific Domain Model
export.lisp	CASU FORTRAN identifiers exported for use in other packages that "use" casu-f package
extract-casu-f.re	Extracts AST information for the CASU DM
system.lisp	Loads RET Files

4.2.6 The cd Subdirectory

This version of the Call Diagram component is language independent. It is implemented using the CASU domain model. CD can generate a Call Diagram in any source language, as long as the CASU has been extracted from the AST.

The functionality is similar to that of the old CD component, with an added Detail window present which provides details on the set of unique subprogram calls made from one subprogram to another.

Table 4-4 Contents of the cd Subdirectory

FILENAME	DESCRIPTION
dm-cd.re	Call Diagram domain model
dm-global-cd.re	RET CD global domain model
rg-cd.re	Call Diagram (CD) representation generator
system.lisp	Loads the RET files
uid-cd-act.re	Call Diagram (CD) Mouse and Menu Actions
uid-cd-detail-view.re	Call Diagram Detail Window Draw Functions
uid-cd-hyp.re	Model-to-view and view-to-model functions for the CD
uid-cd-view.re	Call Diagram (CD) Draw Functions and Mouse Handler

4.2.7 The ded Subdirectory

This version of the Declaration Diagram component is language independent. It is implemented using the CASU domain model. DED can generate a Declaration Diagram in any source language, as long as the CASU has been extracted from the AST.

The functionality is similar to that of the old DED component.

Table 4-5 Contents of the ded Subdirectory

FILENAME	DESCRIPTION
dm-ded.re	Declaration Diagram domain model
dm-global-ded.re	RET Declaration Diagram global domain model
rg-ded.re	Declaration Diagram (DED) representation
system.lisp	Loads the RET files
uid-ded-act.re	Declaration Diagram (DED) Mouse and Menu Actions
uid-ded-hyp.re	Declaration Diagram (DED) Hyperlinks
uid-ded-view.re	Declaration Diagram (DED) Draw Functions

4.2.8 The dfd Subdirectory

This version of the Data Flow Diagram component is language independent. It is implemented using the CASU domain model. DFD can generate a Data Flow Diagram in any source language, as long as the CASU has been extracted from the AST.

The functionality is similar to that of the old DFD component. The file rg-dfd-sub is not provided in this implementation, as the DFD subprogram structure will be provided by the CASU domain model. In the old version, DFDs were only provided for the Right Hand Side (RHS). Now, DFDs can be created for both the RHS and Left Hand Side (LHS).

4.2.9 The global Subdirectory

The global domain model is used by all other RET packages. The DM was placed in its own package, and the package was included in a use parameter during the creation of other RET packages. This allows these other packages to refer to global identifiers without the external global scoping.

It is possible that this package could be included in the CASU package, since the CASU would probably be loaded at all times. However, It seems appropriate to have global RET classes, maps and functions separated unto themselves.

Table 4-6 Contents of the dfd Subdirectory

FILENAME	DESCRIPTION
dfd-record-map.re	Maps variable names to records in FCR
dm-dfd.re	Dataflow Diagram domain model
dm-global-dfd.re	RET DFD global domain model
dm-uid-dfd-menu.re	Dataflow Diagram Menu Domain Model
fsi-bfi-dfd.re	File System Interface (Binary) for Dataflow Diagrams
rg-dfd-all.re	Data Flow Diagram (DFD) Generator
rg-dfd-data.re	Data Flow Diagram (DFD) Buffer and repository node generation
rg-dfd-gen.re	Data Flow Diagram (DFD) Generator
rg-dfd-util.re	Data Flow Diagram (DFD) generator utilities
system.lisp	Loads the RET files
uid-dfd-act.re	Data Flow Diagram (DFD) Mouse Handlers and Action Routines
uid-dfd-menu.re	Data Flow Diagram (DFD) Diagram Window Menu Action Routines
uid-dfd-view.re	Declaration Diagram (DED) Draw Functions
utility.lisp	General LISP support for RET

Table 4-7 Contents of the global Subdirectory

FILENAME	DESCRIPTION
dm-global.re	RET global domain model
export.lisp	Global identifiers exported for use in other packages that "use" global package
system.lisp	Loads RET files

4.2.10 The hid Subdirectory

The Hierarchical Interactive Diagram (HID) structure is a set of common functions that exists among several Redocumentation components. HID is an abstract representation of a text-based diagram. CD, DED, and SRC are specific types of HID diagrams. HID functions are required for any of these views, since the specific view generators call the abstract functions in HID to produce diagrams that exhibit a similar look and feel to one another.

Table 4-8 Contents of the hid Subdirectory

FILENAME	DESCRIPTION
dm-hid-dfnl.re	Hierarchical Interactive Display (HID) Draw Functional Domain Model
dm-hid-obj.re	Hierarchical Interactive Display (HID) Object Domain Model
system.lisp	Loads the RET files
uid-hid-dfnl.re	Hierarchical Interactive Display (HID) Draw Functions
uid-hid-sup.re	Hierarchical Interactive Display (HID) Support Functions

4.2.11 The jovial Subdirectory

The jovial subdirectory of lsret contains another subdirectory named jovial-parser. The jovial-parser subdirectory contains the JOVIAL parser/analyzer that was jointly developed with the AAV&V project for use on LSRET and some sample JOVIAL code for testing. It must be emphasized again, that the JOVIAL parser/analyzer had not yet been integrated into the LSRET Prototype at the time the project was stopped.

Table 4-9 Contents of the jovial-parser Subdirectory

FILENAME	DESCRIPTION
Hughes1.jovial	Sample JOVIAL code from Hughes
Hughes2.jovial	Sample JOVIAL code from Hughes
Hughes3.jovial	Sample JOVIAL code from Hughes
README	Instructions for using the JOVIAL parser/analyzer
iris	JOVIAL parser/analyzer that creates IRIS graphs
iris_and_resolve	Similar to iris with some variable nodes resolved to their declarations
iris_to_image	Produces a postscript file from the IRIS graphs generated by iris and iris_and_resolve
parser	JOVIAL parser
sample1	Small sample of JOVIAL code to test the parser/analyzer
sample1.ps	Postscript file generated from sample1
test.iris	Output file from iris run using test.jovial as input
test.jovial	Larger sample (than sample1) of JOVIAL code to test the parser/analyzer
test.jovial.ps	Postscript file generated from test.jovial

4.2.12 The pak Subdirectory

This version of the Packager component is language independent. It is implemented using the CASU domain model. PAK can generate a Package

Diagram in any source language, as long as the CASU has been extracted from the AST.

The functionality is similar to that of the old PAK component. This component would be used if the target language were Ada or any other where the concept of packages exists. Hence, for a legacy-to-C implementation, this component would not be utilized to exhibit the goal of macro program restructuring.

Table 4-10 Contents of the pak Subdirectory

FILENAME	DESCRIPTION
dm-global-pak.re	RET Packager global DM
dm-pak.re	Packager Domain Model
dm-uid-pak-menu.re	Packager Diagram Menu Domain Model
fsi-bfi-pak.re	File System Interface (Binary) for Packager Diagrams
pak-clu.lisp	LISP support for pak-rfg.re
pak-rfg.re	Packager (PAK) Resource Flow Graph (RFG)
rg-pak.re	Packager Representation Generator (of PAK-LIBRARY-NODE data structure)
system.lisp	Loads the RET files
uid-pak-act.re	Packager (PAK) Mouse Handlers and Action Routines
uid-pak-menu.re	Packager (PAK) Diagram Window Menu Action Routines
uid-pak-view.re	Packager (PAK) User Interface Creates Windows
utility.lisp	General LISP support for RET

4.2.13 The redoc-lite Subdirectory

This directory, redoc-lite, contains a single file, system.lisp, that initializes a RET session in a language-independent mode. A CASU DM can be loaded. The system.lisp calls many other system.lisp files (the "make-system" command). These components are loaded in a specific order.

4.2.14 The src Subdirectory

This version of the Source Code Listing component is language dependent. It is implemented using the source AST. In the future, this capability would have been developed using a language-independent approach, similar to the other Redocumentation capabilities.

Table 4-11 Contents of the src Subdirectory

FILENAME	DESCRIPTION
clu.lisp	Common Lisp Utilities for SRC
system.lisp	Loads the RET files
uid-src-act.re	Source Code Listing (SRC) Mouse and Menu Actions
uid-src-hyp.re	Model-to-view and view-to-model functions for the SRC
uid-src-view.re	Source Code Listing (SRC) Draw Functions and Mouse Handler

4.2.15 The uid Subdirectory

This directory contains all of the Main RET Window source and executables. The UID is almost identical to the old RET UID. Some pull-down menus are all that have changed.

Table 4-12 Contents of the uid Subdirectory

FILENAME	DESCRIPTION
dm-global-main.re	RET Main Window global Domain Model
dm-main.re	RET Main Window Domain Model
dm-pir-global.re	PIR global Domain Model
dm-pir.re	Primary Internal Representation (PIR) Domain Model
fsi-bfi.re	File System Interface (Binary) for File PIRs
system.lisp	Loads the RET files
uid-main.re	Main RET Window

4.2.16 The refine Directory

The refine directory contains the Refine environment configurations, called "worlds," used to run Refine-based applications. These "worlds" can be loaded to start a Refine session in an emacs [11] buffer.

Table 4-13 Contents of the refine Directory

FILENAME/WORLD	DESCRIPTION
README.refine	Instructions for using saved "worlds"
all	World with all Refine extensions (Ada and FORTRAN)
bare-bones	World with no Refine extensions (language-independent)
fretless-base	World with the Ada Refine extension
un-ada-ed	World with the FORTRAN Refine extension

4.2.17 The ret Directory

The ret directory contains the source and executable files for the old RET. The file system.lisp lists the order of compiling and/or loading of files for the RET.

The source files written in Refine have an .re extension. When compiled, the executables have an .lfasls2 extension. LISP source files have a .lisp extension, and compiled files have an .fasls2 extension.

RCS was used for revision control. RCS is a GNU product, available through the Free Software Foundation. The default header is provided in rcs-header.re.

Use the first lisp function in refine-commands.re to load the ret. Start the RET at the Refine prompt with the "(ret)" command. Follow the instructions in the ASRET Users Manual [5].

The directory structure is different on the development machine than on the customer machine. Refine and many global LSRET files are located in /fs1/reasoning/*. RET source and compiled files are stored on individual user directories. The directory ~/ret/ includes the RET virtually intact from ASRET (with some modifications). The language independent RET files are all in subdirectories of ~/lsret/. The ~/refine/ directory contains Refine saved-worlds for several situations.

Table 4-14 Contents of the ret Directory

FILENAME	DESCRIPTION
dfd-record-map.re	Maps variable names to records in FCR
dm-cd.re	Call Diagram domain model
dm-ded.re	Declaration Diagram domain model
dm-dfd.re	Dataflow Diagram domain model
dm-global	RET global domain model
dm-hid-dfnl.re	Hierarchical Interactive Display (HID) Draw Functional Domain Model
dm-hid-obj.re	Hierarchical Interactive Display (HID) Object Domain Model
dm-main.re	RET Main Window Domain Model
dm-pak.re	Packager Domain Model
dm-pir.re	Primary Internal Representation (PIR) Domain Model
dm-rg-ada.re	Ada Code Generator Domain Model
dm-scp.re	Source Code Processor Domain Model
dm-uid-dfd-menu.re	Dataflow Diagram Menu Domain Model
dm-uid-pak-menu.re	Packager Diagram Menu Domain Model
fsi-bfi-dfd.re	File System Interface (Binary) for Dataflow Diagrams
fsi-bfi-pak.re	File System Interface (Binary) for Packager Diagrams
fsi-bfi.re	File System Interface (Binary) for File PIRs
Implicit-fns.re	Replaced by rg-signature.re
make-package.re	Creates the ASRET-SCP package
make-pkg.re	Creates the ASRET-SCP package
pack-data.re	Routines to split common blocks into packages with the routines to which they correspond
pak-rfg.re	Packager (PAK) Resource Flow Graph (RFG)
refine-commands.re	Helpful user commands for use during RET session
rg-ada-type.re	Ada Code Generator - Deduce data types
rg-ada.re	Ada Code Generator
rg-cd.re	Call Diagram (CD) representation generator
rg-ded.re	Declaration Diagram (DED) representation
rg-dfd-all.re	Data Flow Diagram (DFD) Generator
rg-dfd-data.re	Data Flow Diagram (DFD) Buffer and repository node generation
rg-dfd-gen.re	Data Flow Diagram (DFD) Generator
rg-dfd-sub.re	Data Flow Diagram (DFD) subprogram structure initialization
rg-dfd-util.re	Data Flow Diagram (DFD) generator utilities

Table 4-14 Contents of the ret Directory (Cont'd)

FILENAME	DESCRIPTION
rg-implicits.re	Replaced by rg-subprogram.re
rg-package.re	Locates External and Implicit packages in Packager
rg-pak.re	Packager Representation Generator (of PAK-LIBRARY-NODE data structure)
rg-signature.re	Subprogram signatures for implicit functions and external subroutines
rg-subprogram.re	Generates packages for implicit functions and external subroutines
scp-anal.re	Source Code Processor (SCP) Analyze
scp-comment.re	Transform Comments
tran-act.re	Transform Comments
tran-bits.re	Transformations for objects which are used in logical expressions
tran-rules.re	Transform FORTRAN statements
tran-subprogram.re	Transform implicit function calls and external subroutine calls
uid-cd-act.re	Call Diagram (CD) Mouse and Menu Actions
uid-cd-hyp.re	Model-to-view and view-to-model functions for the CD
uid-cd-view.re	Call Diagram (CD) Draw Functions and Mouse Handler
uid-data-act.re	Data Item Mouse and Menu Actions
uid-data-view.re	Data Items View
uid-ded-act.re	Declaration Diagram (DED) Mouse and Menu Actions
uid-ded-hyp.re	Declaration Diagram (DED) Hyperlinks
uid-ded-view.re	Declaration Diagram (DED) Draw Functions
uid-dfd-act.re	Data Flow Diagram (DFD) Mouse Handlers and Action Routines
uid-dfd-menu.re	Data Flow Diagram (DFD) Diagram Window Menu Action Routines
uid-dfd-view.re	Data Flow Diagram (DFD) User Interface Creates Windows
uid-hid-dfnl.re	Hierarchical Interactive Display (HID) Draw Functions
uid-hid-sup.re	Hierarchical Interactive Display (HID) Support Functions
uid-main.re	Main RET Window
uid-pak-act.re	Packager (PAK) Mouse Handlers and Action Routines
uid-pak-menu.re	Packager (PAK) Diagram Window Menu Action Routines
uid-pak-view	Packager (PAK) User Interface Creates Windows
uid-src-act.re	SRC (Source Code View) Menus and Actions
uid-src-mouse.re	SRC (Source Code View) Mouse Handlers
uid-src-view.re	Source Code View
util.re	Debugging Utility Functions
util2.re	General-purpose utility functions

Table 4-14 Contents of the ret Directory (Cont'd)

FILENAME	DESCRIPTION
initialize-session.lisp	Make the scp package and set various options
load.lisp	RET Prototype System loader
pak-clu.lisp	LISP support for pak-rfg.re
system.lisp	Loads the RET files
utility.lisp	General LISP support for RET

4.2.18 The spag Directory

The spag directory contains the SPAG FORTRAN control flow restructuring tool that was acquired and used by the old RET Prototype.

4.3 EXTRACTING AND RUNNING THE PROTOTYPE

This section contains the instructions for extracting the RET from the archive tape that was created in closing down the project and running the RET. ***It must be noted that the project was stopped with much work still in progress so many functions are incomplete and there may be defects in some of the functions.***

4.3.1 Extracting the RET from Tape

In the process of closing down the LSRET project, an archive tape [12] of the files described previously in this Chapter was created. This section contains the instructions for extracting the files making up the RET. This procedure should be done with the help of your System Administrator since it involves mounting tapes and creating new directories to which you will need access in order to run the RET.

1. Log on to your UNIX account.
2. Change directories (UNIX command "cd") to the directory that you want to be the root for the LSRET Prototype. If the directory needs to be created first consult with your System Administrator or use the UNIX command "mkdir."
3. Load the archive tape [12] into a tape drive and mount the tape drive (UNIX command "mount").

4. Extract the files from the tape using the UNIX command "tar xvf." This function will create the same directory structure as previously described in this Chapter and copy the files listed into the proper directories.
5. Rewind the tape using the UNIX command "mt rewind," unmount the tape drive (UNIX command "umount"), unload the archive tape from the tape drive, and store it.

4.3.2 Starting and Stopping the RET

This section contains the instructions for starting and stopping the RET. The procedure assumes that all of the needed files have been extracted from the archive tape (see Section 4.3.1) and are on a working UNIX platform in the directories described previously in this Chapter. This section is not meant to be a User Manual for the RET, but is included to help a user launch the RET and to describe how to exit from it. After starting the RET, the user is free to explore all of the functions in the RET found in the drop-down menus. To start the RET:

1. Log on to your UNIX account. If your account does not default to an X-windows environment, then get into X-windows.
2. Change directories (UNIX command "cd") to the directory from which you want to run the RET and start an emacs buffer (type "emacs&" at the command prompt).
3. Follow the instructions in the file "README.refine" found in the refine (sub)directory (see Section 4.2.16) to load the proper saved "world" that you need to run the RET.

4. There are two main variants of the RET: the old RET or ASRET [4]; and the new RET or LSRET [8 and 9]. The old RET is self-contained. To invoke the old RET: 1) load the file "refine-commands.re" found in the lsret (sub)directory (see Section 4.2.2) in the emacs buffer; 2) find the first "(progn" in the refine-commands.re file and copy or type the entire "(progn" down to the enclosing ")" into the command line of the empty emacs buffer and press the "return" (or "enter") key; 3) after the "(progn" has completed, type "(ret)" into the command line of the same emacs buffer as in step #2 and press the "return" (or "enter") key, this will launch the old RET. The new RET comes in two separate modules, the analysis module (see Section 4.2.3) and the redocumentation module (see Section 4.2.13). To invoke the new RET modules: 1) in the command line type "(make-system "<path>/<module>")" where <path> is the full directory path to the lsret directory (see Section 4.2.2), where the LSRET files are stored and the <module> is either "analyze" for the analysis module or "redoc-lite" for the redocumentation module, and press the "return" (or "enter") key; 2) after the "(make-system" has completed, type "(uid::ret)" into the command line of the emacs buffer and press the "return" (or "enter") key, this will launch the new RET module. For the redocumentation module to produce any meaningful outputs: the analysis module must have been run earlier and the user must have completed the "Save CASU" operation found in the drop-down menus; and after starting the redocumentation module the user must complete the "Load CASU" operation found in the drop-down menus using the same CASU name as in the analysis module.

To stop any RET session:

1. Position the mouse pointer over the "▼" symbol in the upper left corner of the RET window and press the right mouse button. This will cause a menu of commands to appear on which you should select the "Quit" command causing the RET window to be closed.
2. In the emacs buffer, drop down the "Files" menu and select the "Exit Emacs" command. You will see a dialogue box asking whether or not to kill the active processes (in this case Refine) to which you should click on the "Yes" button. This will close emacs.

5. CONCLUSIONS

The technology developed by TASC under the LSRET program was to be used to assist in the reengineering of avionics software. Much of the existing avionics software is written in either the JOVIAL or FORTRAN programming languages and is poorly documented for maintenance purposes. LSRET was being developed to support engineers in transforming the JOVIAL or FORTRAN code into the more modern Ada programming language and also to be used to (re)document either the legacy or the reengineered systems. This would improve the maintainability of the avionics software in two ways. LSRET could reengineer the software into a language that is more modern and promotes better software engineering practices; and also makes it easier to find engineers who are trained in the use of the new language. Second, LSRET could redocument the legacy and reengineered systems making them easier to maintain in the future.

In the area of software language-to-language translation, LSRET would provide a semi-automated approach to reengineering. Manual translation of software from one source code language to another is slow and very error prone, but does allow redesign and utilization of more modern features of the target language. This ultimately results in a much more maintainable system but forces a long testing and operational period before the translated software can achieve the same level of reliability as the legacy software. Purely automated translation of software, while fast and fairly accurate, merely moves the syntactic and semantic limitations of the legacy language into the reengineered software. This creates a reengineered system that only utilizes the equivalent features in the target language and may well result in software that is as hard, or harder, to maintain than the original software. LSRET is unusual in that it tried to combine the best aspects of both approaches by involving a "user-in-the-loop" to redesign and making use of some of the modern features of the Ada language while providing faster, more accurate automated translation of the low-level algorithmic code. This would result in reengineered systems being fielded faster and more reliable than with manual translation and more maintainable than with purely automated translation.

In the area of specific language support, LSRET would provide some important capabilities that are either non-existent or rare. For FORTRAN-to-Ada reengineering tools, some do exist, but fall mainly in the area of purely automated tools generating what the industry has dubbed as "AdaTRAN." In the

realm of JOVIAL-to-Ada reengineering, few if any tools exist with little hope of new tools emerging. Also, LSRET has been designed to be language-independent to the extent possible. We have demonstrated the capability to support multiple legacy languages and adding another target language, such as C, should also prove to be not very difficult.