metaware.

the complete IT modernization solution

Refine® Technology Overview and Basics

A Metaware White Paper

EXECUTIVE SUMMARY

Drawing on 14 years of leading more than 180 successful Mainframe Modernization projects in 10 countries, and Refine®™, the most powerful reverse engineering technology commercially available, Metaware has built the Refine® Modernization Solutions to cover the entire lifecycle of mainframe modernization. Refine® solutions allow for *industrialized* modernization to guarantee the highest Return on Investment (ROI).

Metaware has built Refine® Solutions upon Refine®, the most powerful and scalable reverse engineering technology commercially available.

Refine® has emerged in the early 1990s and became the reengineering software of leading public and private IT laboratories such as AT&T Bell Labs, Boeing, hp, IBM, Lockheed Martin, NASA, Thomson-CSF, Stanford University, MIT, University of California at Berkeley, University of Illinois.

Thanks to Refine®, Metaware provides solutions for:

- Asset Analysis – Enabling automated discovery, analysis and measurement of the size, complexity and quality of mainframe source code.
- Language Translation and Database Conversion – to exit legacy or proprietary 3GL, 4GL, or databases such as IMS DB, IDMS, Adabas, VSAM files to more viable, popular, corporate standards.
- Replatforming – Consisting in reusing IBM z/OS, z/VM and z/VSE, Bull GCOS 7 and GCOS 8, and HP3000 by migrating applications "as is" (rehosting) onto Open Standards-based Architecture.
- Mass Change – Allowing for any systematic changes to source code to be achieved regardless of volumes for internationalization, standardization, or upgrade purposes.

This document presents Refine® technology and fundamentals.

1 REFINE® OVERVIEW

Refine® is a programming environment. It provides a programming language and a set of language processing tools (parser, compiler, etc.). The language is called the Refine® language, or just "Refine®", when there is no ambiguity as to whether we are discussing the language or the whole system. The most innovative features of the system, which distinguish it from other commercially available programming environments, are the following:

- The Refine® language the first programming language to provide an integrated treatment of set theory, logic, transformation rules, pattern matching, and procedure. Because the language is executable (as opposed to "paper" languages) and allows you to write programs at the specification level, the Refine® system supports programming with "executable specifications".
- Refine® provides a powerful object base – the Code Base Management System –that you can query and modify using the Refine® language. Refine® programs and other software-related objects (documents, test cases) are stored in the object base. This makes Refine® ideal for writing programs that manipulate software objects, such as program transformation systems and documentation systems. You can use the object base for representing the objects in your application domain, and all Refine® tools for manipulating the object base can be used by your application.
- The Refine® compiler is implemented as a program transformation system. Refine® programs are compiled into Lisp by successive application of program transformation rules. The rules themselves are written in Refine®; the Refine® compiler (and most of the rest of the system) is bootstrapped (i.e., written in its own language).
- Refine® provides facilities that allow you to design your own languages using a BNF-like notation to describe the grammar of your language. Given a grammar, Refine® compiles a lexical analyzer, parser, pattern matcher, pattern constructor and prettyprinter for your language. The syntax definition system is an optionally loaded system. It is described in a Refine® product documentation entitled "DIALECT User Guide".

2 DESIGN GOALS OF REFINE®

Refine® initial design requirements were:

- it should provide an **integrated environment that supports high-level programming**. This requires a language that is capable of expressing program specifications.
- it should be a tool for **analyzing and reformulating programs**. This requires that it provide for explicit models of programs and the other programming-related objects that go together to form a programming project. Tools must be provided that support the creation and the manipulation of models of programs.
- it should be **extensible**. You should be able to build domain-specific programming environments using the tools provided by Refine® as a foundation.
- it should allow you to **specify programs using whatever programming style** you find most appropriate to the problem at hand. This means that the Refine® language should support most common programming styles.
- it should **support the formal use of stepwise Refine®ment**—the process of converting specifications into target implementations by successive application of formal Refine®ment steps that embody distinct design decisions. The important point here is that these Refine®ments are explicit, that their rationale can be made explicit, and that there typically exist alternative Refine®ments of a given specification.

3 REFINE® SYSTEM STRUCTURE

3.1 Specification-Level Programming

The Refine® language is a very-high-level programming language; it contains constructs that allow you to express your programs in a fashion that is easy to write, understand, and modify. These very high-level constructs, which include operators from standard set theory and first order logic, and also a powerful “transformation” operator, facilitate the writing of programs at the “specification” level. At this level almost all extraneous detail has been excluded, so that the program closely resembles the most abstract description of the problem to be solved. This range of very high-level language constructs is not found in other commercially available languages.

When the need arises to modify an existing Refine® program, the changes are made to the specification. Thus support and maintenance occur at a high level, where the decisions can be recorded and monitored, and policy enforcement and design review are more easily achieved.

3.2 Modelling Programs and Other Objects in the CBMS

Refine® provides a medium for storing descriptions of specifications and partially Refine®d programs. This medium is called the Code Base Management System (CBMS). With Refine®, you can model many types of software-related objects in the CBMS. Refine® provides an extensive set of object base tools that can be used to manipulate all of these kinds of objects. This leads to a highly integrated system where other software-related objects (e.g., documents) can be stored in the CBMS as well.

Since your specifications and partially Refine®d programs are written in the same language as the system itself, Refine® can use a single object base as the medium of the Refine®ment process; in its CBMS Refine® can store specifications of application programs, the Refine®ment rules that embody the programming knowledge needed to Refine® these specifications, and the intermediate forms of programs as they are Refine®d.

Since Refine® is intended to be used to model and analyze programs in many different languages and since you may want to be able to design new domain-specific programming languages, the Refine® system includes (as an option) an extensible parsing system. This allows you to describe the syntactic structure of your language by writing a grammar for your language. The Refine® system can then parse programs written in this language and construct an abstract syntax tree for the program in the CBMS. Further, Refine® has grammar-directed pattern matching, pattern construction and prettyprinting tools. The domain-specific grammar definition tools are part of an optionally loaded system. They are described in a separate document titled “DIALECT User Guide”.

3.3 Extensibility

The Refine® system is extensible. Object classes, types, functions and grammars are among the objects that you can define and manipulate. Object classes provide a particularly powerful mechanism for extending Refine®; you can define domain-specific languages to describe your application domain, complete with surface and abstract syntax, and all the features provided by Refine® for manipulating the CBMS.

3.4 Supporting Multi-Paradigm Programming

The Refine® language incorporates a number of programming paradigms. The specification level programming and logic programming aspects of the Refine® language were discussed above. Some other programming paradigms that are supported by Refine® are discussed here.

The Refine® language includes all of the concepts commonly found in procedural programming languages. Refine® includes many powerful object oriented programming features. Although much of the power of the Refine® language derives from its specification level constructs, Refine® provides for programming in many different styles because no one programming style is appropriate to all programming problems.

Because the Refine® language can be compiled and executed, it is an "executable specification language". You gain the freedom to experiment with different high-level program designs, since the task of refining a design into a low-level procedural implementation is performed automatically by the compiler. This encourages a style of programming called rapid prototyping, in which successive designs of a program are converted into prototype implementations, in order to provide quick feedback on the correctness and completeness of the designs. The "design-prototype-assess" process iterates until the design is well understood and debugged, preventing the common woe of uncovering costly design errors at a late stage in program development.

3.5 Program Refine®ment and Explicit Representation of Programming Knowledge

The Refine® language is capable of expressing very-high-level specifications of programs, low-level (or target-level) programs, and partially Refine®d specifications that contain a mixed range of constructs from very-high-level to low-level. For these reasons, Refine® is characterized as a "wide-spectrum" language.

The Refine® language is able to describe program Refine®ments. Refine®ments can be formalized in terms of "transformation rules" that embody a single design decision and rewrite one fragment of code into another. The ability to state transformation rules succinctly requires the ability to formally express "program schemas" or "patterns". For this reason, the Refine® language is designed to be able to conveniently express patterns and transformations.

4 SUMMARY OF REFINE® TOOLS

The environment for Refine® program development includes several components, presented in two groups. The first group, language oriented components, includes the parser, printer, compiler, and a customized text editor. The second set of development tools are those that are not language specific; this group includes the command interface, the context mechanism, the online help system, the browser/editor for the CBMS, and the debugging system.

4.1 The Parser and Printer

The parser parses text and produces objects in the CBMS; one use of the parser is to parse Refine® programs from source text. The printer is the inverse of the parser; it takes objects in the CBMS and produces the corresponding text form. The parser and printer are extensible; you can define your own language, programming or otherwise, and customize the parser and printer to work with that language. This allows you to define a domain-specific language that you can process using all the same methods that Refine® uses on the Refine® language: object base representation, pattern matching, transformation and so on. The tools for defining your own domain-specific language are

part of an optionally loaded Refine® system. These tools are described in Refine® product documentation.

4.2 The Refine® Compiler

The Refine® compiler compiles Refine® specifications into Common Lisp by successive application of program transformation rules. Most of the transformations are specified using the pattern language; patterns are used as schemas to match against program subparts that are then modified to produce new subparts.

Refine® is a strongly typed language, but the burden of declarations and consistency is greatly relieved by the "type analyzer". The type analyzer provides more than passive testing. It combines the information from user declarations, the object base, and the use of data structures within the program in order to deduce the types of undeclared variables when enough contextual information is available to do so. The type analyzer also checks the consistency of declarations with actual usage, and reports all type errors in a program.

The compiler is incremental in two senses;

- First, separate definitions in a program can be compiled independently without need for recompiling other definitions or relinking.
- Second, when compiling a block of definitions, the compiler keeps track of which definitions have changed in a non-trivial way (i.e., more than just formatting), and will only recompile those that have changed.

The benefit of this is that file compilation goes very quickly when you have only changed a few definitions in a file, since the rest need not be recompiled.

4.3 The Customized Emacs Editor

Refine® uses a customized version of Emacs for text editing. The customizations are for two purposes; they make editing Refine® programs more convenient (for example, by indenting programs intelligently and making matching delimiters blink), and they allow you to invoke the Refine® compiler using editor commands.

4.4 The Refine® Command Interface

The Refine® command interface allows you to interact with Refine® by typing commands to an interpreter. Almost all the commands have menu equivalents, but some people prefer a command-style interaction. The command interface is an extension of a Common Lisp read-eval-print loop; this makes for a smooth integration of Refine® with Common Lisp. This smooth integration is preserved by other parts of the system; for example, Refine® can call Common Lisp (and vice versa) with no special foreign function calling interface.

4.5 The Context Mechanism

The context mechanism maintains a hierarchy of states of the CBMS. Previous states can be recreated. A variety of backtracking search algorithms can be easily implemented using this tool. Hence, under either program or user control, the context mechanism facilitates the use of multiple alternatives for whatever the CBMS is used to represent. For example, a scheduling application could use the context mechanism to store several alternative schedules and then run analysis programs to determine which is the optimal schedule.

4.6 The Online Help System

The online help system provides tools for displaying information about Refine® language constructs. There is also a facility for keyword-based access to documentation.

4.7 The CBMS Browser/Editor

The CBMS browser/editor allows you to display objects, move from an object to related objects, and edit the values of attributes of objects, all using a menu-based interface. The browser has special features for viewing programs stored in the CBMS.

4.8 The Debugging System

The Refine® debugging tools let you monitor run-time events (such as function calls and modifications to the CBMS) using either a menu-based, command-oriented, or programmatic interface.

metaware.
the complete IT modernization solution

www.Metaware.fr
Tel. +33 1 30 15 60 00
Fax. +33 1 30 15 06 71

Global Headquarter:

1. Parc des Grillons
60, route de Sartrouville
78230 Le Pecq Cedex – FRANCE