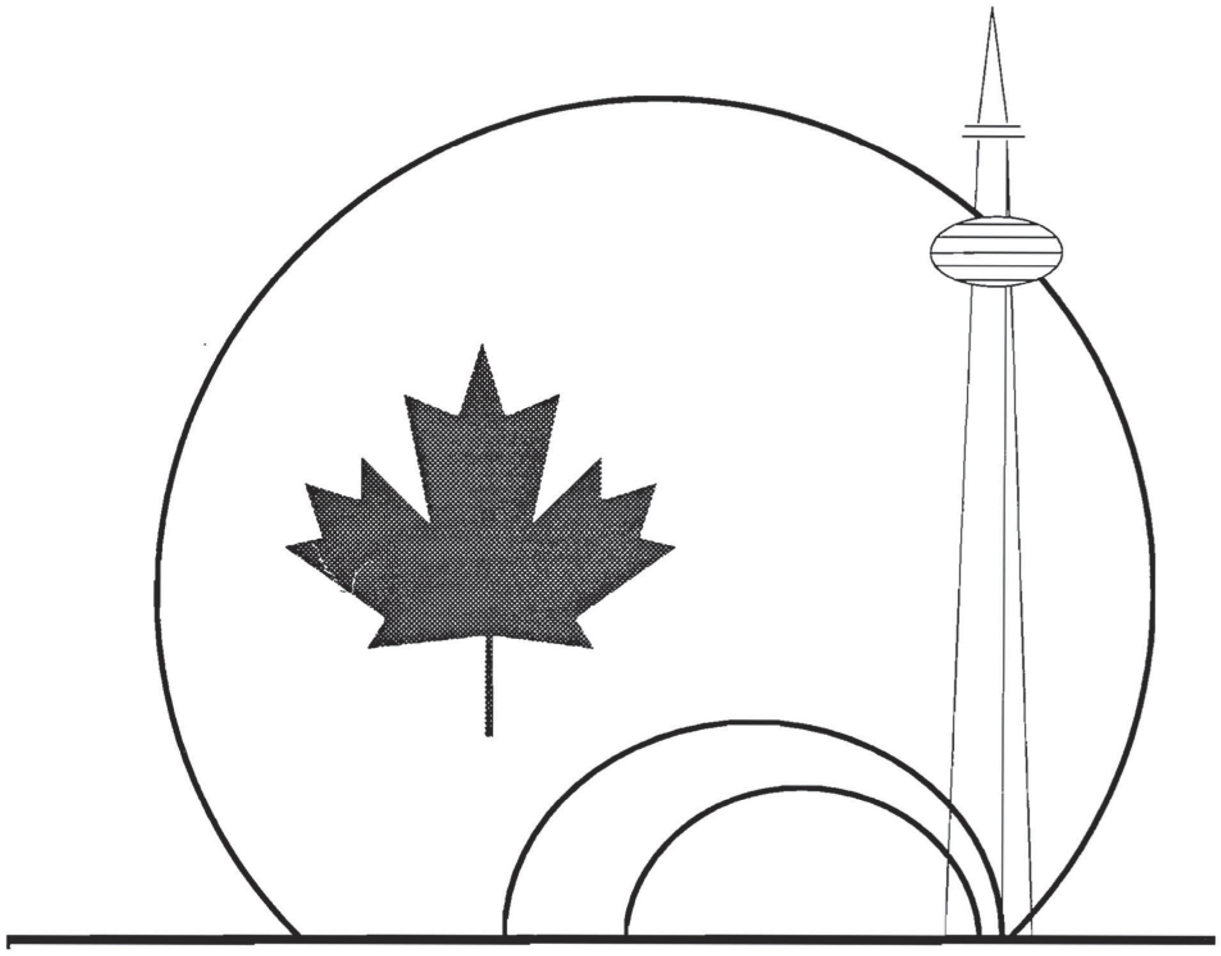
1996
ROCHESTER
FORTH
CONFERENCE

OPEN SYSTEMS



© N. Solnsteff

June 19-22, 1996
RYERSON POLYTECHNIC UNIVERSITY
Toronto, Ontario
Canada

ISBN# 0-914593-16-1

A publication of

The Institute for Applied Forth Research, Inc.
70 Elmwood Avenue
Rochester, NY 14611

Published in the United States of America
Printed in the United States of America

TABLE OF CONTENTS

* Abstracts only are included for these presentations.

Introduction

The 1996 Rochester Forth Conference on Open Systems was held at Ryerson Polytechnic University, in Toronto, Ontario, in Canada June 19-22, 1996. This was the first Rochester Conference to be held outside the United States, and was a great success. Attendees came from Canada, the United States, England, the Netherlands, Australia, Austria, and Russia.

The Conference was hosted by the Institute for Applied Forth Research, Inc., in cooperation with the Southern Ontario chapter of the FORTH Interest Group. The Conference was sponsored by New Micros, Inc., with the continuing support of Mr. Randy Dumse; ITV Corporation, with the new support of Mr. Joe Zott; Ross Technology, Inc. and the support of Mr. Warren Bean and JWK International Corporation, with the continuing support of Dr. Jay Khim. Additional support was provided by Xela Associates, and Alexander Forsley, of Annandale, Virginia and Miller Microcomputer Systems, of Natick, MA.

During most Conferences the apparent theme appears, in contrast to the advertised theme. This year was an exception. Several papers dealt with different aspects of Open Systems. These include an overall view of Open Systems by Dr. Steve Benson, Forth as a scripting language by Dr. Everett Carter; and GENETIX by Bernard Hodson. An even larger view of Open Systems was described in a paper by Peter Johannes, Elizabeth Rather and Stephen Pelc regarding Europay and a background paper on the supporting software, EPIC, by Stephen Pelc. These efforts are all the more far reaching by supporting banking institutions in a variety of countries on a variety of hardware platforms. Finally, two remarkable papers: John Rible presented a small 32-bit RISC Core and Richard Wagner gave details of an airborne simulation.

The Conference Program Chair and Proceedings Editor was Dr. Nicholas Solnsteff and the Facility Chair was Dr. Bradford Rodriguez. I was delighted to have their strong support. In large measure they organized and orchestrated this Conference, resulting in a great success! My thanks goes to both of them, and I look forward to working with them on future Conferences.

No conference can be a success without a large supporting cast. We had very able assistance from: Elliott Chapin, Kirby Dumse, Alexander Forsley, Brenda Forsley, Beverly Galloway, Ken Kupisz, Ken McCracken, Rob McDonald, Wendy Rodriguez, Jon Verne and Robin Ziolkowski. I wish to complement and thank the very helpful Ryerson University Staff, most notably Patti Franklin, of the International Living Center and Peter Young, of the International Conference Center.

--Lawrence P. G. Forsley
Conference Chairman

The Europay Open Terminal Architecture
A Forth-based Token System for Payment Terminals

E. D. Rather,
FORTH, Inc.
111 N. Sepulveda Blvd., Suite 300
Manhattan Beach, CA 90266 USA

Peter Johannes,
Europay International
Chausée de Tervuren 198A
B-1410 Waterloo, Belgium

Stephen Pelc,
MicroProcessor Engineering, Ltd.
133 Hill Lane
Southampton, England

Abstract

Europay, the major European credit card organization (Eurocard, MasterCard in Europe, and other financial systems) is developing technology to support "smart" cards — Integrated Circuit Cards, or ICCs — in the credit cards of the future. This will require new software in all credit card terminals (ranging from 8051-based POS terminals to high-end ATMs). To facilitate this transition, they are designing a token-based system conceptually similar to Open Firmware or Java based on Forth. Using the "Open Terminal Architecture" (OTA) it will be possible for credit card issuers and acquirers to write application programs that will be completely platform independent, and run on all OTA-compliant kernels.

This is the second of three papers on OTA; it will discuss the overall architecture of OTA.

System Components

The purpose of an OTA system is to provide software to run in terminals used in payment applications. Conceptually, there are two hardware environments, and several classes of software. The hardware environments include the *development system*, which is based on a simple PC, and a *target* which is some form of payment terminal. The software includes *development software*, which runs on the PC; *kernels*, which include all platform-specific software in a terminal and other mandatory standard functions; *libraries*, which provide general functions to support terminal programs. and payment applications; *applications*, which are the functions specific to a particular payment product. and *terminal programs*, which perform general non-payment terminal functions and include high-level mechanisms for selecting and executing transactions and associated applications.

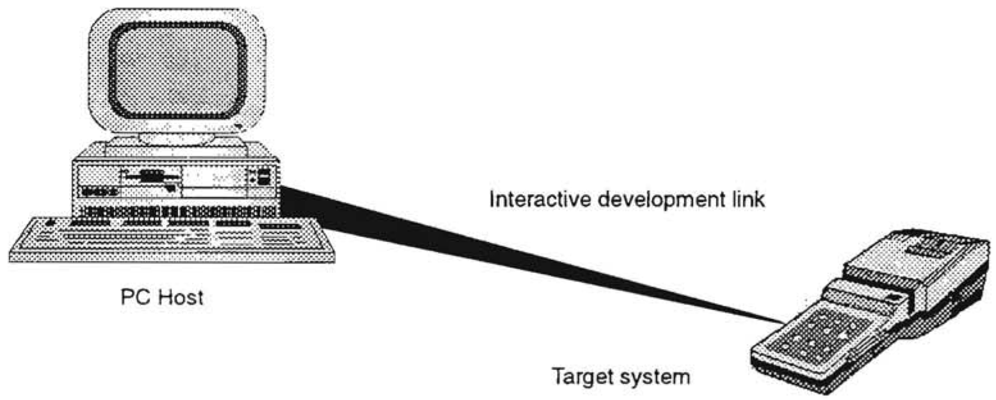


Figure 1

An OTA development environment for a terminal. A small program to support the terminal end of the Interactive Development Link (IDL) protocol is usually placed in the terminal ROM.

Development Environment

An OTA development system is used to develop terminal software, either low-level kernel software or high-level library or application software. Kernel development requires a target terminal to be connected, as the kernel is cross-compiled on the PC host and downloaded to the terminal across the Interactive Development Link. OTA libraries, terminal programs, and applications are also developed on a PC host. Because they are high-level code, they may also be executed on the host for preliminary testing, using the host's version of the standard kernel, and thus a target is not needed. For final testing using the terminal's own kernel and I/O, these programs would be tokenized (see discussion below) and downloaded to the target terminal.

As an example, a prototype development system consists of a PC running Windows 3.11, Windows-NT or Windows 95 and an IDL to a target terminal (see Figure 1). Development software on the PC is based on ProForth for Windows, a product of MicroProcessor Engineering Ltd.

Terminal Target Environments

The target system is any one of a large variety of payment terminals. Actual products range from small, hand-held devices with simple 8-bit microprocessors such as the 8031/51 family to 32-bit computers running operating systems such as Windows-NT or Unix. In order to simplify the production, certification and maintenance of software on such a wide variety of targets, OTA terminal code is based on a single *virtual machine* which is emulated on the actual devices. Prototype coding and testing has shown that this approach is feasible and provides good run-time performance, even on an 8051 CPU.

Each target contains a *kernel* consisting of a standardized set of functions whose CPU-specific implementation is optimized for that specific platform. This kernel supports standard *libraries* and *terminal programs* and *applications* which are written in high-level code for the virtual machine and will therefore run on any terminal with a standard OTA kernel.

Virtual Machine

The software in every OTA terminal is written in terms of a common virtual machine. This is a theoretical 32-bit microprocessor with standard characteristics defining addressing modes, stack usage, register usage, address space, etc. The kernel for each particular CPU type is written to make that processor emulate the virtual machine. The virtual machine concept makes a high degree of standardization possible across widely varying CPU types, and simplifies program portability, testing, and certification issues.

Virtual machine emulation may be accomplished by one of three methods: interpreting tokens representing VM instructions (like Java), translating these tokens into a directly executable "threaded code" form (like Open Firmware), or translating them into actual code for the target CPU. The latter two methods offer improved performance at a modest cost in added target complexity. Present systems employ interpretation.

Kernel

A kernel contains all functions whose implementation depends upon a particular platform (CPU and OS). It includes a selected subset of ANS Forth words, plus a number of specialized OTA functions such as terminal I/O support, token loader/interpreter support, and operations designed to support the particular needs of payment programs. Since it cannot itself be tokenized and downloaded, but must be physically installed (e.g. in PROM), the kernel is intended to be installed once, and not changed thereafter during the lifetime of the terminal. Therefore its functions are carefully designed to be very general in nature and as complete as possible, in order to support a wide range of present and future terminal programs and applications.

A kernel is normally provided by the manufacturer of the terminal in which it resides. It is developed and certified according to the OTA Kernel Specification. Standard kernel functions not appropriate to a particular terminal type (e.g., action of a non-existant device) may be coded for that terminal as null functions, so every kernel has an identical set of functions and the testing and certification process is simplified.

Libraries

OTA Libraries contain higher-level functions that support common features of terminal programs, such as language selection, and common features of applications, such as PIN verification. A terminal may contain several libraries, some accessible to all applications, and some restricted to particular applications or payment systems. Libraries are written and tokenized for the virtual machine, using functions provided in the kernel, and therefore can be run on any terminal.

Terminal Program

A terminal program is part of the Terminal Resident Services and consists of the high-level "personality" characteristic of this terminal type (POS, ATM, etc.). This includes the functions common to all transactions (e.g., card initialization and language selection) as well as the user interface required to select an application and process a transaction. The terminal program at the highest level is typically triggered by a card insertion. Like libraries, a terminal program is written for the virtual machine, based on both kernel and library functions, and supplied in token form. It can therefore be run on any terminal of the appropriate type (e.g., ATM), and is easily changed by downloading over a network at any time.

Applications

A terminal transaction will select an application as part of its processing flow. Applications fall into three general areas: cashless "purses" (Pay Before), debit cards (Pay Now) and credit cards (Pay Later); applications will generally vary in their method of processing a given transaction. Versions of these applications may be provided by different payment systems and further customized by individual issuers or acquirers. Applications are supplied in token form via the communications path, and (if security considerations permit) may be enhanced by token programs on an ICC.

Token Compilers and The Token Loader/Interpreter

Libraries, applications, and terminal programs are written in high-level code for the virtual machine. The OTA development system includes Forth and C compilers for this virtual machine, whose output consists of *tokens*. Tokens may be thought of as machine language instructions for the virtual machine. Tokens are either one or two bytes in length, and therefore represent the program in a form that is both CPU-independent and extremely compact (far more so, for example, than compressed source text).

Each kernel contains a *Token Loader/Interpreter* (TLI), which processes a stream of tokens into an executable form. Once the kernel is installed in a terminal, the libraries, applications, and terminal programs can be downloaded into the terminal in a variety of ways (direct connection to an OTA development host, acquirer hard-wired network, modem and dial-up telephone line, etc.). Program modifications and new applications are downloadable in the same manner whenever needed.

ICC Functions

One function of ICCs is to improve transaction security by incorporating and managing encrypted data and participating actively in the transaction validation process. It is conceptually possible to go beyond these functions and provide for ICCs that also contain program code to enhance a terminal's transaction processing, thereby providing new opportunities for payment products and services. To facilitate this, "sockets" are provided, that can be "plugged" by issuer-specific functions such as loyalty programs, which may be invoked at appropriate points in the transaction processing. The OTA does not currently propose that ICCs contain entire applications, but only plug behaviors that enhance existing terminal applications.

As far as security is concerned, the presumption is that if an ICC passes the decryption and data authentication tests, whatever functions and plug behaviors are on the card have been certified and are syntactically valid. The terminal decides to allow or disallow the card's proposed actions only as controlled by the terminal access security functions.

Open Terminal Architecture Features

The OTA token set provides program portability in systems with multiple CPU types by passing source code programs of various types through an intermediate compiler whose output is a string of tokens. Target terminals then process this code either by further compiling it to native code, translating it to a different form, or by interpreting it directly. Figure 2 illustrates this process.

The OTA token set covers three main areas. The first is the instruction set of a theoretical processor (virtual machine), and thus provides the instructions necessary for the efficient execution of programs. The second allows what are normally called "operating system functions" to be defined. In the embedded systems for which OTA is targeted, system functions cover not only OTA functions such as communications, TLVs, and library functions, but also access between program modules and the associated access control mechanism. The third token group includes tokens specific to OTA processing, including such functions as user message management, database management and handling of TLVs ("Tag, Length, Value" data formats, specified by ISO/IEC 8825, and used for communicating with the chip cards).

The OTA token set has been optimized for use on small terminals with ease of compilation, ease of interpretation, and good code density.

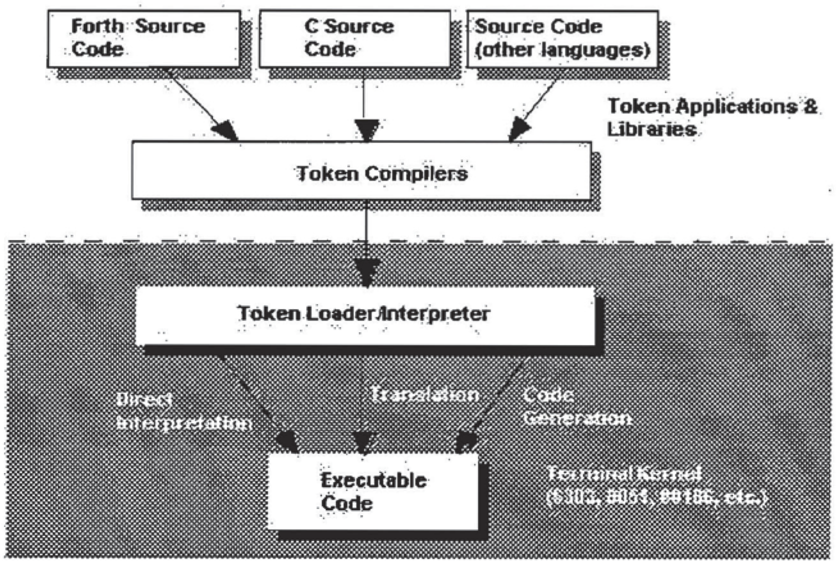


Figure 2

Tokens in the Interpreter ConceptThey may be generated from a variety of source code formats, downloaded to a terminal, and interpreted or executed.

Virtual Machine CPU Features

The OTA virtual machine is based on a two-stack architecture, as seen in Forth. This architecture has been further modified for portability, code density, ease of compilation, and for use with other programming languages. For example, it contains provisions for local variables and frame pointers used in C. Thus, OTA token compilers can be written not only for Forth but also C and other languages. It is a byte-addressed, 32-bit machine, with 32-bit registers and stack elements.

OTA defines a single address space for programs. This address space is accessible for data storage only. Programs may not assume that executable code is in this address space. Depending on the actual CPU and kernel implementation strategies the executable code may be in a different address space, or may be under the control of a memory management unit. In any case, programs are not permitted to access their own program memory directly, and any attempt to do so will be flagged during the program certification procedure.

"External memory" may be used to store databases and program modules; however this space is controlled entirely by the kernel, and access to data therein is provided to client programs only indirectly. This memory may be in mapped pages, flash ROM, disk, or simply more main memory — this is completely at the discretion of the kernel implementor and is transparent to the programs.

Programs and Tokens

The instruction set of the virtual machine is coded as a byte stream of tokens. The most common functions, including most Forth primitives, are expressed in one-byte or "primary" tokens. Some system functions and most OTA-specific functions are two-byte or "secondary" tokens. Some tokens also have associated values, for such things as literal values and branch offsets.

The token compiler compiles source code as a string of tokens which may be downloaded to the target terminal. After downloading, the terminal uses a token loader/interpreter to execute the tokens.

Program and Library Management

The software on an OTA terminal is organized as a set of separate *modules*. A module is a collection of definitions passed through the token compiler as a single unit. The main Terminal Program, each application, and each library are examples of modules.

Modules need to communicate to perform overall terminal functions. A program module, for example a library, contains a number of definitions. Some of these are used only internally, and others must be accessible externally. Following all the definitions in a module there is a special section where definitions to be referred to externally are "exported" or listed in a way that constucts a table of entry points published for access by higher-level modules.

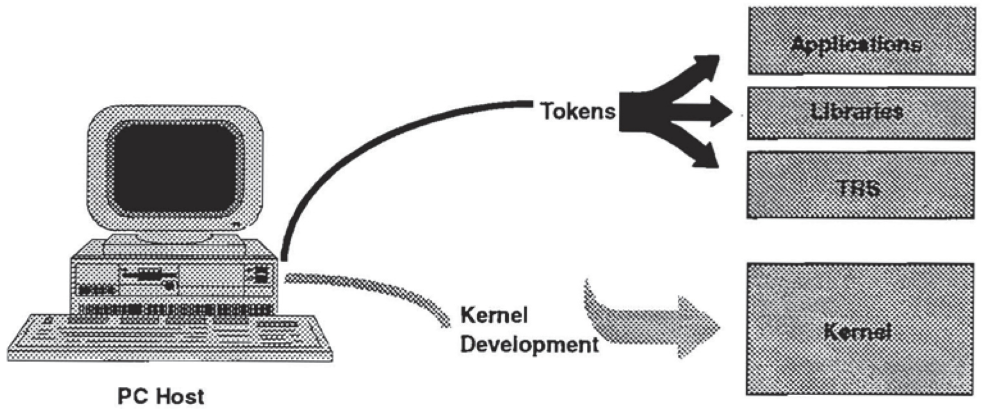


Figure 3

Architecture Summary and Status

The basic architecture of OTA was successfully validated in the prototypes exhibited at the Europay Members' Meeting in Seville. Issues to be revisited include placement of certain functions in the kernel vs. external token programs, minimum and maximum size requirements, and further definition of standard libraries and application functions. In addition, intensive efforts will be directed in the next few months toward development of salable software products, including development kits and validation suites for both kernels and token modules.