CMU-ITC-83-029

WHITE PAPER: The ICE Execution Environment

Document Number 003-001/002.01

DRAFT

Unclassified

June 21st, 1983

MH Conner, TC Peters, LK Raper

Information Technology Center
Carnegie-Mellon University
Schenley Park
Pittsburgh, PA 15213

ABSTRACT

The Integrated Computing Environment (ICE) to be developed by the ITC is a System within which numerous Workstations will function cooperatively across a communications network. The nodes of this network consist of Workstations and various 'Servers' that provide different kinds of communal services for the Workstations so connected. The software produced to support the individual Workstations, as well as the Servers, must minimally support communication amongst these nodes through a standard communications protocol. Beyond such minimal requirements, the ITC specifies an 'Execution Environment' within which that software can be produced in a facile fashion. This environment guarantees software portability and provides for modular extensibility.

The Execution Environment is an abstract model of an idealized base-machine and an idealized family of extended-machine services. This model provides the conceptual foundation upon which ITC software is designed and implemented. The degree to which an actual machine matches the architected features of the abstract model is the degree to which optimal performance is realized.

PREFACE

Although the 'Ideal' machine will never exist -- for there will never be agreement upon what such an ideal might be -- it is possible to specify, for a given set of circumstances, a 'friendly' set of features that would ease many of the burdens typically faced by software development personnel. Such features would eliminate or ameliorate these burdens without introducing unacceptable performance or resource costs; they would be simple to implement in state-of-the-art hardware; they would provide a conceptual framework within which experimentation and invention would not be constrained.

These features constitute the Execution Environment. In particular, the Execution Environment eliminates the usual 'system dependencies' that often inhibit or impede development of new programs. It provides safe haven for both matured and experimental software; it provides a vehicle that facilitates the introduction of novel hardware functions as well as improved software; it encourages commitment of effort to an evolutionary system.

Given the context within which the ITC is developing the ICE system -- namely, the rapid evolution of both software and hardware technology, as well as the evolving sophistication of the user community -- it is imperative that the ICE software must itself be highly portable, extensible, and robust. Lacking a well-defined and well-structured Execution Environment, this would be an impossible task. Thus, while the Execution Environment may be considered to be optional, it is nevertheless a fundamental condition for the success of the Project: a major role of the Execution Environment will be seen to be its use as the vehicle of Portability.

ACKNOWLEDGMENTS

Some of the Execution Environment's definition is taken from the Carnegie-Mellon SPICE Project, specifically the ACCENT Operating System Kernel. Additionally, due to the University environment, concepts and facilities of various versions of UNIX have been incorporated. Finally, various members of the Project have brought forward materials from their own practices.

CAVEATS

- The reader is advised to consult the ITC Glossary for precise definition of terms as employed by this White Paper.
- Much of the material in this White Paper is volatile.

CONTENTS

1.0 Introduction 1
1.1 Considerations 2
1.1.1 Distribution 2
1.1.2 Extensibility 2
1.1.3 Recovery 2
1.1.4 Exception Handling 2
1.1.5 Language Extensions 2
1.1.6 Protection 3
1.1.7 Integrity 3
1.1.8 Timeliness 3
1.1.9 Instrumentation 3
1.2 Objectives 4
1.2.1 Rapid Portability 4
1.2.2 Facile Extensibility 4
1.2.3 Topological Transparency 5
1.2.4 Fail-Soft Data Integrity 5
1.2.5 Secure Access 5
1.2.6 Optimal Performance 5
1.3 Requirements 6
1.4 Constraints 7
1.5 Plans 8
1.5.1 Group Structuring 8
1.5.2 Near Term 8
1.5.3 Long Term 9
1.5.4 Schedules 9

2.0 ICE Sub-System Component Overviews 11
2.1 The ICE Kernel 13
2.2 Transaction Management 14
2.3 Resource Management 15
2.4 Process Management 16
2.5 Program Management 17
2.6 Space Management 18
2.7 Time Management 19
2.8 Instrument Management 20
2.9 Data Structuring 21

LIST OF ILLUSTRATIONS

Figure 1. ICE EE Schedule 10
Figure 2. ICE EE Relationships 12

1.0 INTRODUCTION

The ITC Project is to develop a System that supports a very large community of users in an interactive fashion. The Product is to be known as the "Integrated Computing Environment" (ICE). The nature of this System Product is such that its development will entail research and development activities over a period of several years, during which time advances in software and hardware technology are expected to proliferate in abundance. Accordingly, it is essential that the Product be sufficiently robust to withstand far-reaching and unexpected upheavals in these areas. Of immediate concern is the need to ensure that design and implementation efforts throughout the Project's duration can be continued in an evolutionary fashion despite changes in the actual hardware employed for the Processor and its Peripherals, as well as possible changes in the Project's personnel.

To these ends, the Execution Environment is partitioned into two functional levels: a repertoire of primitive, low-level facilities that constitute the **ICE Machine**, and families of sophisticated, high-level functions that constitute the **ICE Services**. The former require the speed and stability that direct encapsulation offers, the latter are more suited to the flexibility that is offered by software support.

1.1 CONSIDERATIONS

The architecture, design, and implementation of the ICE EE is not effected in a vacuum. It draws upon, and depends upon, existent or planned support at both hardware and software levels. In particular, it is to be developed upon a product-level UNIX System and subsequently ported to one that is currently being developed for an advanced function Workstation itself under development. That Workstation will itself provide a level of software support that masks various 'hardware dependent' features of the System environment. It is upon its level of 'Virtual Machine Interface' (VMI) that the ICE EE will ultimately operate.

As prelude to specific Objectives and Requirements, there are a number of 'considerations' that have and will continue to affect the definition of the ICE EE. These are identified and discussed below.

1.1.1 DISTRIBUTION

The Product will be a System whose elements are distributed across a vast number of Workstations and Server Machines. The problems of development are magnified both by the sheer numbers of such focal points and the fact that there does not even exist classic solutions to many problems even in the simpler context of a non-distributed System.

The development of the various Sub-Systems must be facilitated by an Execution Environment that both eases the explication of design and avoids unwarranted assumption of function best left in the domains of such Sub-Systems. The smallest possible suite of facilities must be provided, and they must make minimal assumption upon the structure of the System in which they are used.

1.1.2 EXTENSIBILITY

While the tendency will be to minimize the individual facilities of the EE, to avoid unforeseen constraints upon the other Sub-Systems, the need will arise to add or modify facilities both of the EE and of the other Sub-Systems. It is necessary that the Architecture of the System, and of the EE in particular, assist in such evolution. This is to be effected by architecting the construction of the System in a fashion that supports replacement of resources not as exceptions but as the norm.

1.1.3 RECOVERY

The System bears the burden that it be "as reliable as the electric service". It will contain hardware and software that does fail, that needs maintenance, that suffers extension. These activities must be possible without taking down the System. The Architecture of the System, and supporting facilities, must make this feasible. Because of this feature, there might well be some performance penalty.

1.1.4 EXCEPTION HANDLING

1.1.5 LANGUAGE EXTENSIONS

1.2 OBJECTIVES

Development of the ICE Execution Environment is no simple feat. It will consume human and product resources; it introduces a layer of potential complexity and performance degradation; it will require significant tuning and maintenance upon its deployment. Given these potential drawbacks, its creation and employment must be based upon significant benefits -- viz, the following Objectives.

1.2.1 RAPID PORTABILITY

Perhaps the major burden of the ICE Execution Environment is the need to support the movement of ICE Applications from one real machine to another, from the technology base of today to that of tomorrow. There are two major levels of program portability:

- Source Code
- Execution Environment

The former is easily handled by proper employment of HLL programming and modification of a compiler's Code Generation component. The later is more subtle, and not easily mechanized -- for it pervades the very design of programs. It is this problem that the ICE Execution Environment attacks by specifying an "Extended Target" to be employed during both the design and implementation efforts.

The Extended Target may be understood as a "generic" object, whose realizations in actual machines varies. Programs developed for such an object are usually produced through a High Level Language, which presents to its users an abstraction of the target machine for which object-code is generated. The Execution Environment is just such an abstraction -- both the ICE Machine and the ICE Services discussed below provide the abstraction that an HLL provides for the design and implementation of 'native' ICE Programs.

(Note that the present Project does not intend to develop any hardware to support such a machine -- its "realization" for the present Project will be entirely through **software** support.)

1.2.2 FACILE EXTENSIBILITY

A major consideration of the ICE Product is its customer base: a sophisticated University community that intends to use the Product as the vehicle for not only automating much of today's known procedures (accounting, course grading, etc), but more importantly to build advanced educational tools such as Tutoring Facilities and Experiment Simulators. Furthermore, there are groups of users who plan not to extend the Application Program base, but rather to experiment with System performance and capability through parametric modifications and wholesale replacement of individual modular units as well as full Sub-systems.

The needs of such users are, in fact, the very needs of the ITC development group. Accordingly, in addition to the classic techniques of program modularity (Units of Composition, Compilation, and Packaging), the Execution Environment must provide methods of modularity based upon both the possible multiplicity of processors and the existence of the ICE network. These forms of modularity should provide the means to 'intercept' functional requests and then interpose substitute support.

1.2.3 TOPOLOGICAL TRANSPARENCY

While each Workstation will enjoy its own unique identity, as each user will be uniquely identifiable, it must be possible for any user to use any Workstation without regard to its identity. Thus, although obvious constraints such as the lack of a particular kind of graphic display unit prevent certain programs from performing on a given Workstation, the user must be able to perform the majority of tasks without concern for whether certain support mechanisms or data reside locally upon the individual Workstation or remotely somewhere within the network.

This 'transparency of resources' is ultimately supported by the network functions, but is directly supported by the Execution Environment's support of well-isolated Processes. Amongst such Processes exist functions such as File Servers, Name Servers, Network Servers, etc. These functions take advantage of the Execution Environment's ability to handle 'distributed' functional decomposition: a File Server, for example, will exist not only within those Workstations designated as its 'Cluster Nodes' and its 'Central Node', but also partially within each Workstation that employs it. That the set of Processes constituting the File Server is distributed throughout numerous Workstations is transparent to that function (this implies, of course, that its development might even be effected within a single Workstation -- freeing it from dependencies upon a Network Server).

1.2.4 FAIL-SOFT DATA INTEGRITY

The trauma of data loss, whether through misadventure or mischief, is frequently major. Such losses can occur through hardware failure at any moment. To facilitate mechanisms that perform "staging," the use of the Process mechanism to support "Daemons" must be feasible.

1.2.5 SECURE ACCESS

???

1.2.6 OPTIMAL PERFORMANCE

???

1.3 REQUIREMENTS

???

Given the above Objectives, the following Requirements are implied.

1.4 CONSTRAINTS

Although it might be possible to develop a System that meets all the above Objectives, the present effort must face certain real-world limitations -- viz, the following Constraints:

- Cost

  Each Workstation, with the exception of those employed as Servers, must be available at retail for $3000 or less. Various models of the Workstation could be more or less expensive, but the 'standard' model must meet this price constraint.

- Feasibility

  The Project's duration is such that the Workstation must be in production within 3 years.

1.5 PLANS

To meet the stated Objectives within the stated Constraints, the Project must needs stage the development of the Workstation -- specifically, it must stage the introduction of the Execution Environment.

1.5.1 GROUP STRUCTURING

The small band of ICEEE devotees operates with the following divisions of responsibility. (It is emphasized that listing of an item under an individual's name does **not** indicate ownership, but rather that the individual is responsible to the group for the care and feeding -- development and documentation -- of that item.)

MHC

1. General
   a. NIL Consultant
2. Research & Liaison
   a. At large
3. Components
   a. Program Mgt

TCP

1. General
   a. Chief Whip
   b. Programming Conventions
2. Research & Liaison
   a. VMI
3. Components
   a. Process Mgt
   b. Space Mgt
   c. Time Mgt
   d. Data Structuring

LKR

1. General
   a. PM Consultant
2. Research & Liaison
   a. UNIX
   b. SNA/LU-6.2
3. Components
   a. Transaction Mgt
   b. Resource Mgt
   c. Instrument Mgt

1.5.2 NEAR TERM

Since it will take some time before even a prototype of the Execution Environment is available, an evolutionary Project development practice must exist immediately. Given that the Execution Environment is meant to support numerous Applications, and that even an Operating System can be seen as an Application, and that a major Objective of the Execution Environment is the facile porting of Applications employing it -- given this, it is possible to begin development of the various Server functions and certain Fundamental Tools and Applications upon such a supported Operating System.

To be sure, such a temporary substructure will not provide some of the features that the ICE Execution Environment offers. It should, however, be possible to develop prototypes of these Programs even lacking these features. When the ICE Execution Environment does become available, these Programs will continue to operate upon their Operating System base -- for that base will itself be supported by ICE. Subsequently, these Programs can be upgraded to employ directly those ICE Execution Environments that will make them immediately portable to various target machines, in support of various now superior Operating Systems.

It happens that a proven Operating System is presently available, with the necessary support tools (compilers, linkers, etc) -- UNIX (TM) exists incarnate upon the IBM PC, SUN MicroSystem, and the IBM 370.

The planned sequence of near-term activities is:

1. Specification of the prototype Execution Environment
2. Design and Implementation of the prototype upon the IBM PC
3. Porting of the prototype from the IBM PC to the SUN MicroSystem
4. Porting of the prototype from the IBM PC to the IBM 370

The prototype will be developed in the generic C language, utilizing the existent generic facilities of Unix.

1.5.3 LONG TERM

The employment of XENIX (TM) and its C-language compiler provides both the time required to develop the ICE Execution Environment and the opportunity to learn from the experiences of those developing ICE Sub-Systems, Fundamental Tools, and Fundamental Applications.

1.5.4 SCHEDULES

The plans are to be realized in time as shown in Figure 1 on page 10.

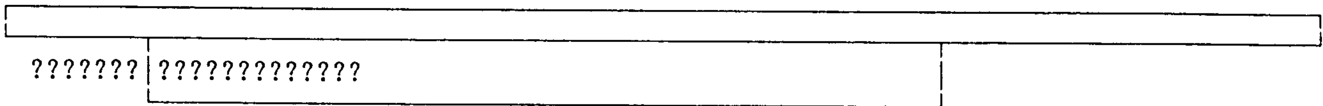

Figure 1. ICE EE Schedule

2.0 ICE SUB-SYSTEM COMPONENT OVERVIEWS

The ICE Execution Environment is but one of the Sub-Systems that comprise the System. It is related to its own environment as depicted in Figure 2 on page 12

As seen, the Execution Environment of both the ICE EE as well as that of all other Sub-Systems is partially that provided by a UNIX-interface (termed the 'UI'). The ICE EE itself may be considered to be an extension of the facilities offered by such an interface. During early development, Sub-Systems, including the EE itself, must be content with the facilities supported by the UI. As development progress, two changes will occur:

1. Additional Facilities unique the ICE EE will appear;

2. Portions of the UI implementation will be replaced by ICE EE implementations.

At one point, it will become mute whether a EE facility is ICE or UNIX: they will have been so merged as to become a new version of the UI support.

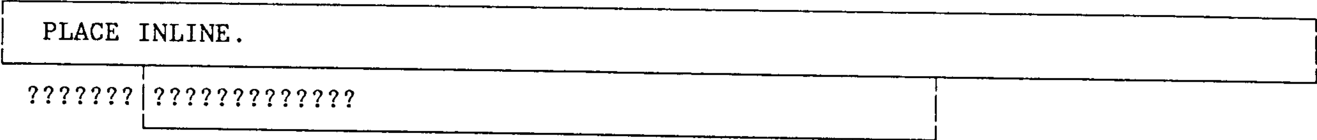

Figure 2. ICE EE Relationships

2.1 THE ICE KERNEL

This Component is the closest to the Processor itself. It is the minimal set of functions that need to be atomic within the Sub-System, as dictated by requirements of both the individual Workstations and the System at large.

2.2 TRANSACTION MANAGEMENT

This Component supports the integrity of logical 'units of work', where such units are defined by the various Sub-Systems. It provides the means to specify dynamically the bounds and properties of Transactions, and the means to control their activities.

2.3 RESOURCE MANAGEMENT

This Component is integral to all other EE managers, for it is the single function that is able to bring resources into existence. Without passage through the Resource Manager, a potential resource cannot be accessed. It provides System-wide unique identifiers for all resources. These identifiers are termed 'handles'.

As a resource is brought forward, it is associated with a handle-value. This value is unique -- it has never been used before the association, it will never be used again.

2.4 PROCESS MANAGEMENT

This Component supports both 'light weight' and 'heavy weight' Processes. The former are termed 'Local Processes' (ie, within the same address space), the latter 'Remote Processes' (ie, within different address spaces). Local Processes provide more optimal execution, while Remote Processes provide topological transparency.

The Process concept has many facets. Chief amongst them is its use as a mechanism of modularity.

2.5 PROGRAM MANAGEMENT

This Component supports the coordinated execution of Programs within Processes. It is itself oblivious to the Process boundary; it deals only with the Program text and the actions required to activate the Programs.

2.6 SPACE MANAGEMENT

This Component manages the allocation and deallocation of main-memory to Programs. Requests for its services are either implicit (eg, Automatic storage) or explicit (eg, Controlled storage). It coordinates the usage of main-memory as a Workstation resource, and is accordingly limited to the domain of a single Workstation.

2.7 TIME MANAGEMENT

This Component provides traditional Date and Time information, as provides primitives for Software-timers used either for measurement of elapsed time or as alarms.

2.8 INSTRUMENT MANAGEMENT

This Component provides various facilities to standardize and coordinate performance measurement and dynamic tracking of 'exceptional' conditions. It supports Counters, Logs, and Traces. Such objects may be defined and manipulated dynamically; they may be local or remote, unique to a single Program or Process, or known throughout the System.

2.9 DATA STRUCTURING

This Component centralizes the design and implementation of the several classical data structuring techniques. It supports the manipulation of Chains, Rings, Stacks, Queues, Trees, etc. It implements each in the pragmatic fashion dictated by its user. Use of this Component's facilities is expected to have significant impact upon the standardization and subsequent defect-reduction of most Sub-Systems.

T H E   E N D