Use of an environment classification model

Marvin V. Zelkowitz*
Computer Systems Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899

Abstract

Various reference models have been proposed for the classification of features present in an integrated software engineering environment. In this paper, two such models are studied and a target system is mapped to the set of services present in these models. The results of this mapping and comments on the effectiveness of the models are given.

Keywords: Environment frameworks; Environment mappings; Integrated environments; Reference models

1 Introduction

Improvements in software technology depend upon improving the process used by professionals to design and construct software systems and improving the set of products available to aid in this activity. Aside from the more traditional tools, such as compilers and editors, there are probably thousands of other products sold to aid in developing requirements, designs, and validation of the resulting systems.

But to effectively use these CASE (Computer Aided Software Engineering) tools, three attributes must be true:

1. The tools must have a consistent internal interface to permit one tool to communicate effectively with another, to permit data to pass among them, and to allow for alternative tools to be substituted for a given tool relatively effortlessly.

2. The tools must have a consistent interface to permit users to move from tool to tool easily.

3. The tools must provide some useful functionality in the software development process.

Unfortunately, many (all?) of the current CASE tools fail this third attribute and there is little that can be automated to improve on this. It requires a creative effort to design effective tools. However, much progress has been achieved in addressing the first two attributes [8]. The role of an *integrated* software engineering environment (SEE) is to provide those consistent interfaces – both internal and external – and a common set of services (i.e., functions) so that tools which meet this third attribute can be integrated into an existing system and be easily used.

Today there is considerable interest in developing these integrated environments, Many vendors and standardization groups are working on such designs (e.g., DEC's Cohesion, IBM's AD/cycle, IDE's Software Through Pictures, HP's Softbench, CEARM, ECMA PCTE, CAIS-A, plus many more). However, there is still no common environment architecture that these models are designed for.

Because of this, there is a need for a general environment model for describing such efforts. Currently, we view an integrated environment as: (1) A framework providing a common set of services for all programs executing in the environment and providing consistent user and internal programmatic interfaces; and (2) A "populated" set of tools that provide the functionality needed for software development.

Within this structure, there have been several efforts at developing models of frameworks so that it is possible to describe two alternative environment architectures using consistent terminology and concepts. Once the software engineering field accepts such a common model, it will then be possible to evaluate various environment architectures and develop effective evaluation criteria for choosing among them.

Of the several reference models that have been proposed, three have received considerable attention:

1. The NIST/ECMA frameworks reference model, initially developed by the ECMA TC33 Task Group on the Reference Model (TGRM) [2] and extended by the NIST Integrated Environment Working Group (ISEE) [4]. This is a model describing some 48 different services (i.e., functions) which environment frameworks

*On leave from the Department of Computer Science, University of Maryland – College Park. This work was performed at NIST for the NGCR PSESWG program.

may provide.

2. The NGCR PSESWG reference model, which is a model giving the full functionality of a populated environment built on top of the NIST/ECMA framework [6]. The term *user services* are often given to the functionality described by this model, since such services are generally provided by tools invoked by the user.

3. The P1003.0 open system environment reference model, which is an environment model based upon the POSIX architecture for open systems [5].

Of considerable interest is whether the various models serve any practical purpose and what can be learned by using them. Towards this end, in this paper an existing system will be described using the first two of these complementary reference models (NIST/ECMA and PSESWG). In Section 2, I describe the two reference models and in Section 3 I describe the system to be modeled. Section 4 gives the results of this mapping and describes what is learned from the process. Section 5 gives some comments on the models themselves.

2 Reference Models

2.1 Framework reference model

A framework provides a common set of services needed to support application programs written for the domain of the environment. For a software engineering environment that means framework services support tools which aid in the design, development, deployment and management of the software development process. Although environment frameworks and data repositories are sometimes used synonymously, the data storage aspects of the repository are only one of the set of services needed to support software development activities.

In 1988 ECMA/TC33/TGRM began work on describing such a framework reference model. The initial version was published in 1990 and included heavy emphasis on the data repository aspects of the framework [2]. Beginning in 1989, NIST's Integrated Software Engineering Environment (ISEE) Working Group, working jointly with TGRM, extended the model to include a greater variety of services. The original model was revised in 1991 [4].

It is important to realize that the model is just a catalog of services that may be applicable to an environment framework. There was no implied architecture in describing any of the services. The current model consists of 48 services, grouped for convenience into 6 general classes. Each category is given below. Appendix A provides a brief summary of each service.

1. The 21 *object management services* provide for the storage, retrieval and management of a persisent object store.

2. The six *process management services* provide for developing process models of the development life cycle and the management of these processes within the environment.

3. The one *communication service* provides for communication among tools using communication paths like messages, RPCs and shared data storage.

4. The six *policy enforcement services* provide the confidentiality, security and integrity requirements on an environment.

5. The nine *user interface services* provide the interface between the executing program and the user who is interacting with the environment.

6. The five *framework administration services* provide for the installation and tailoring of tools, users and other objects into the framework.

2.2 Environment reference model

Since 1991, the U.S. Navy's Next Generation Computing Resources (NGCR) Project Support Environment Standards Working Group (PSESWG) has been developing a reference model of end-user services that would execute on top of the NIST/ECMA environment framework [6]. It is also a service-based model oriented around the set of activities that go into the development process. Appendix B lists the set of services in this model. The model consists of four major classes:

1. *Technical engineering services* are those services concerned with the building of projects on a project support environment. These include most of the technical tools in use in a software engineering context (e.g., requirements, design, coding and testing tools).

2. *Technical management services* are services related to software construction, but not of direct concern to most developers. Configuration management, reuse management, metrics and logistics support are the related services.

3. *Project management services* are those services needed to manage a project. These include those services for planning, estimating and tracking progress.

4. *Support services* are those services used by all environment users. These include tools like editors, desk-

top publishers, electronic mail and related services.

3 SUPPORT

Evaluation of environments relative to a reference model has been termed a *mapping*. Various modern integrated environments have been compared to various versions of the NIST/ECMA framework model [1] in order to determine how well each system meets the implied functionality of the framework model. Applying such a mapping to an older system, built before the current generation of concepts and jargon, should provide additional data as to the effectiveness of these models in describing systems, and should shed additional light on the efficacy of the models. Since the developers of several of these environments were also developers of the some of these models[1], mapping the model to their own system is somewhat circular – similar assumptions might have gone into both the environment and model. An independent mapping would be of value in order to check this assertion.

For most of the mid-1980s, I was involved in building an environment for the development, execution and testing of Pascal programs. This system, named SUPPORT, consisted of approximately 23,000 lines of Pascal source code in 31 modules and provided tools for developing Pascal programs using either a syntax-directed editor or a line-oriented editor. It included an interpreter for execution of programs and various testing tools for traced execution, variable inspection and other debugging activities. The system executed on both Unix 4.2 and IBM PC computers, and for several years the PC-based version was used in the introductory Computer Science course in the Department of Computer Science at the University of Maryland – College Park.

Since SUPPORT was built using strict[2] software engineering guidelines relevant to those days, the description of this system relative to the various environment reference models seemed like a worthwhile activity – both to describe SUPPORT in current environment framework terms and to help evaluate the effectiveness of the reference models themselves. In this paper, I will analyze the PC-based version of the system. Both systems share perhaps 95% of the same code with the PC version implementing more "framework" support to handle features that are "automatic" under Unix. Therefore, the PC version should be more indicative of its framework architecture.

The basic approach to this mapping is given by Figure 1. The grid represents the set of services in

[1] For example, the original TGRM framework model grew out of the ECMA development of the PCTE (A Portable Common Tool Environment) specification [7].

[2] Well, *almost* strict.

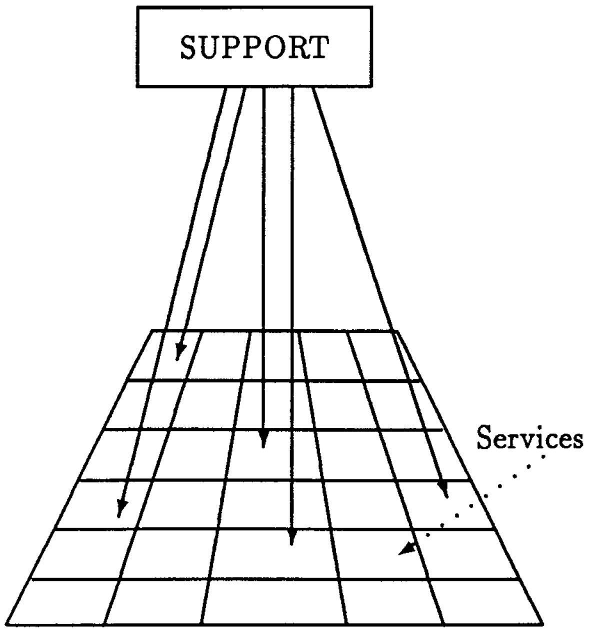

Figure 1: Describing frameworks

the framework and the SUPPORT functions will be mapped to those services. When completed we should have a good indication of how SUPPORT meets the service descriptions of the model.

4 Mapping

4.1 Structure of SUPPORT

Internally, SUPPORT consists of 31 modules that make up nine basic phases that execute as a single program under the MS-DOS operating system. Only those functions external to one module are considered in this analysis since these are the only functions that other phases (i.e., tools) have access to. Thus the set of externally defined procedures forms SUPPORT's framework interface.

The 10 phases of SUPPORT used in this analysis[3] can be separated into SUPPORT framework services and SUPPORT end-user services. These phases are:

Framework services. The five framework phases are:

Control. Control phase to process user input and direct execution to the appropriate phase.

Application Specific Utilities. Application specific utilities needed by various phases. This includes many supporting services, such as data type conversions, which are not part of the framework.

[3] I will consider the underlying MS-DOS operating system as well as the low-level Pascal read-write commands as one of the phases.

Phase	Mods	Lines	Stmts	Procs
Control	3	1662	732	10
Struct. Ed.	6	6176	2334	60
Line Editor	2	1835	793	4
Design	1	1030	494	1
Execution	3	2963	1164	20
Test	1	1141	465	5
Appl. Specific	5	3026	1134	57
User Interface	3	1711	748	36
Object Mgmt	4	1499	716	45
Data decls	3	1902	–	–
MS-DOS	–	–	–	7
Totals	**31**	**22,945**	**8,580**	**245**

Table 1: Basic SUPPORT data

User Interface. These create and manage windows and menus on the display.

Object management. Data repository services.

DOS. The MS-DOS operating system.

End-user services. The five end-user phases are:

Structured Editor. The syntax-editing phase that is the main program building component of SUPPORT.

Line Editor. An internal line editor for adding linear strings of ASCII text rather than using the syntax-editing paradigm.

Design. A graphical design tool for navigating and displaying program structure.

Execution. An interpreter for execution.

Test. An interactive tool for testing execution and investigation of run-time data structures.

SUPPORT was designed using fairly strict data abstraction principles. The user interface consisted of a set of logical windows independent from the actual screen real estate[4]. Internal program storage and external displays were kept separate. The Pascal program was stored as a parsed tree and was reformatted into ASCII in order to be displayed on the screen. Users were expected to know little of the internal format of their programs. Thus, for example, the Structured Editor phase continually translated the program between tree-format and linear strings. With user input (even for a good typist) being only 5-10 characters per second, a slow IBM PC-XT had plenty of time to perform these transformations in real-time in well under a second for a screen update.

4.2 Mapping to reference models

The basic data describing SUPPORT is given by Table 1. For each of the 245 externally defined procedures, its place within the framework was determined and all references to that procedure were identified. Table 2 presents a summary of this process. Each "*" represents a reference to that service within the indicated phase, and the numbers[5] indicate the number of functions within that phase that implement aspects of that service. Names in *italics* indicate services that are not present within SUPPORT.

While the table provides valuable information, the numbers must be viewed carefully. Simply counting services does not provide much useful information. For example, some functions such as *digits*() may simply be a one line expression determining if a given character is between '0' and '9' while others like *edfcn*() are the entire implementation of the internal Line Editor. "Importance" of functionality does not follow from number of functions. The assignment of functions to various phases, however, does provide real information, which I will come back to later.

Service	Ctl	SE	LE	De	Ex	Tst	Utl
S/W Design	*						
S/W Comp.	*			*	*14	*5	*
S/W Debug	*	*		1	*		10
S/W Test	*						
Text Proc.	*	*	*1				
Rev. Eng.	*			1			
Status Mon.	*						1

Table 3: Distribution of User functions across phases

The full environment model provides less information. Since SUPPORT is a program development system, it implements only five of the 42 user services of the full environment reference model. Only these five service classifications are shown in Table 3, and all are involved in the details of program generation.

Since user commands are the principal interface between the user and the environment, the distribution of these commands according to phase invoked and service provided should be informative. Table 4 lists only those services invoked by direct input from the user. (That is, the user command is interpreted by the Control phase, which then calls the appropriate phase to perform the indicated service.)

4.3 Summary of mapping

An evaluation of the mapping data concerning SUPPORT reveals much about its structure. From Table 2, the underlying MS-DOS operating system becomes obvious. Security (as represented by policy enforcement services and the object management access

[4]Today, a system like X-windows would probably be used, but SUPPORT was designed before the acceptance of X-windows as an interface standard.

[5]The numbers sum to more than 245 since several functions perform more than one service. See Section 5 for a further explanation of this.

Service	**Ctl**	**SE**	**LE**	**De**	**Ex**	**Tst**	**Utl**	**UI**	**OMS**	**OS**
Object Mgmt										
Metadata	*	5	1							
Storage	*1	*15	*		*2		3	*	*19	
Relationship	*1	*17	*	*	*		*			
Name		*2								
Location	*	*	*				*		1	
Transaction										
Concurrency										
OS Process Support										
Archive	1	*			*			2		
Backup	*	2								
Derivation										
Replication	*	*1			*					
Access Control										
Function Attach										
Canonical Schema		*1								
Version										
Composite	*	*2	*							
Query	*	*4	*		1		*1			
Triggering										
Views										
Data Interchange	*	*	*2		*					
Process Mgmt										
State	*				1		*			
Enactment	*1									
All other services										
Communication										
Services										
User Interface										
Metadata		*1								
Session	*	*12			*		5			
Security										
Name										
Application Interface	*1	*1	*	*	*	*	*2	*17	*	
Dialog										
Presentation	*1	*3	*	*	*2	*	*6	*16	*2	
Internationalization	*	*		*		*	*	2		
User Assistance	*2	*	*		*	*	*1	*	*	
Policy Enforcement										
Services										
Framework Admin.										
Tool Registration	*2									
Resource Registration	*	1					*	1	*1	
Metrication	*				*		2		*	
User Administration										
Self-Configuration										
Application Specific	*3	*	*1	*1	*14	*5	*38	*1	*1	
Operating System	*	*1	*		*	*	*2	*	*16	7

Table 2: Distribution of framework functions across phases

Service	Ctl	SE	LE	De	Ex	Tst	Utl
Frame Svcs							
Name							1
Location							1
Backup		3					
Replication							1
Composite		2					
Query		1					
Data Int.		2					
UI Dialog	1	12					
UI User Assist							1
Res. reg.		1					
User Svcs							
S/W Design		1		1			
S/W Comp.					1		
S/W Debug	2					1	6
S/W Test					1		1
Text Proc.		3	2				
Rev. Eng.				1			
Status Mon.							1

Table 4: Distribution of user commands across phases

control service) is non-existent. There was only the physical security of the machine. Students would load their programs from floppy disks and use the entire machine until they completed their activities.

Similarly, SUPPORT executed each phase sequentially because of a lack of multitasking features in the operating system. This is reflected by a lack of concurrency services, as represented by omission of the concurrency, transaction, triggering and most of the process management services. The only process management operations were *syscmd*() that invoked the tool asked for by the user at the keyboard and the asynchronous interrupt operation *Break key* to permit the user to stop execution and provide an alternative input command.

SUPPORT functionality is fairly well divided among functional phases, but functional phases are not as clearly separate as originally thought. For example, 35 user interface functions are defined in the User Interface phase; however, the Structured Editor phase implements a sizeable minority with a scattering of other functions in other phases.

On the other hand, the Design phase shows only one external function and the Line Editor only four. Understanding the development process clearly explains why. Most of SUPPORT was built by four graduate students working together. In hindsight it seems like the close cooperation of these individuals permitted them to trade information among themselves and allowed them to install various external functions that others needed. No one person was responsible for any

Service	Ctl	SE	LE	Ex	Utl	UI	OMS
Storage	*1	*15	*	*2	3	*	*19
By type							
trees	*	*7	*	*		*	*5
names	*1	*8		*			*8
act. recs.	*	*		*2			2
LALR		*					1
files	*			*	3		*3

Table 5: Fine-grained data storage

single phase.

The Design phase was a later addition to SUPPORT added by a fifth student. This student viewed the set of SUPPORT external functions as a framework definition and was able to build this product without violating any of the existing data separation among the phases. Only one external function was needed – to invoke the tool when requested by the user via a keyboard input command.

The Line Editor consisted of two modules – the editor that manipulated a window full of text and an LALR parser that performed the continual conversion between parsed trees and linear ASCII strings. Each of these were built by single individuals and required almost the minimum number of interfaces: (1) Invoke Line Editor; (2) Initialize LALR parsing tables; (3) Invoke LALR parser; and (4) One other parser interface.

The close cooperation among individuals certainly leads to a better understanding of their designs, but it probably leads to harder maintenance and enhancement tasks. On the other hand, working in isolation and having only the functional interface to work with leads to a more independent design. The costs of working with one of these approaches is not indicated by the available data, but the fact that such characteristics exist is indicated.

At first glance, Table 2 seems to indicate that object management services are distributed across most (7 out of 10) phases of SUPPORT. However, closer examination reveals that this is not necessarily so and may point out a weakness in the framework model. In Table 5 the Storage service is analyzed in some detail. The first line reproduces the corresponding line from Table 2. However, SUPPORT manages five basic data types: (1) Program storage in the form of syntax trees, (2) Program identifiers, called names, (3) Runtime activation records for the interpreter, (4) LALR parsing table data, and (5) Files of data (of various types). There is no special data storage for screen display items, since programs are stored as trees and continualy translated to screen objects upon demand.

When the storage functions are separated according to their data structure, as given in the rest of Table 5, a different pattern emerges.

Program *tree* storage is only handled by the Structured Editor and OMS phases, *names* are managed by the Control, Structured Editor and OMS phases, *activation records* are from the Execution and OMS phases, the *LALR parsing table* is only in the OMS phase and *files* are only in the utility and OMS phases. This is much more what you would expect in a modular design. The result of this indicates that the framework model may have a weakness in identifying fine-grained data objects. As discussed later, this is related to the weakness in the model in not yet really addressing the important issue of data integration.

It is not surprizing that only 10% of the external functions within SUPPORT are concerned with user level functionality, as represented by the services of Table 3, since manipulation of the user interface is more under the control of the user. This is represented by the table of user commands (Table 4) where over 40% (19 of 45) of the possible user commands reflect user level services. In addition, although there are no functions within SUPPPORT for User Interface Dialog services, about 25% of the user commands are concerned with display motion[6].

5 Comments on the reference model

In addition to describing SUPPORT, this mapping exercise provided additional feedback on the effectiveness of the two reference models used.

Since SUPPORT is a source code development system, it is not surprizing that it supports only seven of the services of the PSESWG model. These services were the Text Processing and the software development activities of design, compilation, testing and debugging. There were two commands relative to reengineering and environment status monitoring. The reengineering command builds the call tree of a program and was developed as part of a design tool, indicating the close relationship between design and reengineering. Status monitoring was added as an internal check to monitor the environment as it gobbled up memory unmercifully during its early development period.

Some detailed comments on the NIST/ECMA frameworks model obtained during this exercise are: Some of these are:

Uses of model. The following were all viewed as positive characteristics of this model:

[6]It is known that the NIST/ECMA model is weak in the area of user interfaces, so the UI service descriptions may not be accurate.

1. The reference model provides a consistent mechanism for describing software products and permits a common notation for describing the functionality that is implemented. This mapping exercise was effective in describing SUPPORT and, as shown in Section 4.3, in indicating attributes about SUPPORT that may not have been obvious beforehand.

2. Little seemed to be missing in the set of framework services. Services not present in the model were all grouped under the term "Application specific," and all seemed to be services that are outside of an environment framework. The functionality of these services was involved directly with the implementation of the SUPPORT features themselves.

3. Characterizing the set of functions within SUPPORT turned out to be a valuable documentation aid. It provided a consistent notation for describing features in a system to those not intimately involved in its design or construction.

Issues in model development. The NIST ISEE and ECMA TGRM committees are still undergoing the process of revising the framework model. The following are all issues that still need further clarification:

1. The set of services are somewhat imprecise. For example, the SUPPORT function *symbind*() looks up a name in a symbol table and sets the scope of the variable so defined. In this current mapping it was considered to be a Relationship service since it established the relationship between a name object (the variable being defined) and a tree object (the location in the parse tree containing the procedure with the name's declaration). On the other hand, it could have also been classified as a Query service (Is the name already defined in the symbol table?), a Name service (Translate a user's name into an internal tree pointer to its declaration), a Location service (Where is the name defined?) or a Storage service (Create appropriate symbol table information). In a few instances, multiple definitions for a given service were indicated, but were not done for most of the SUPPORT functions. For most systems the set of supported functions may be orthogonal to the set of service classes of the reference model.

As long as the model and the system being mapped come from the same general design model, the framework services and environment services should be similar. This was not particularly true in SUPPORT's case. However, as use of standardized reference models grow, this will orient thinking of developers towards certain common directions[7] and this orthogonality issue will gradually disappear. This has already happened with the seven layer OSI communications model

[7]Which is the critical reason we better get the model right!

which started out as simply a set of services and now is viewed as a network architecture [3]. Similar results will probably occur in software engineering.

2. Implementation of a service does not mean complete functionality of that service. This has already been described in Section 4.2. Also, as already explained, the number of functions implementing a service does not indicate a completeness level for covering that service.

3. Some services appear to have *virtual* implementations. For example, Table 2 indicates no communication service functions. However, SUPPORT certainly communicates among its phases. One method of communication is shared data using some of the object management services. In SUPPORT's case, communication is often a byproduct of any data storage or access operation or can be a side effect of parameter passing among external procedures.

4. Object management services seem to be scattered among all phases of the system. However, the OMS phase implements coarse-grained functionality dealing with allocation and storage of data (mostly the Storage service). As previously shown, most of the other OMS framework services are functions dealing with specific data types (e.g., name objects, tree objects, runtime objects). A decision was made in this mapping to include these as OMS services for the SUPPORT framework rather than application specific utility services needed by SUPPORT alone.

It is my belief that this will be true of other products so mapped. The OMS repository seems well suited for storage of large objects. There is a need to handle fine grained-data and direct interaction of data among system phases. Some of the issues of data integration, or the passing of information among environment components, is not handled by this model, and the need to include concepts like ASIS, Diana, Polylith or other data transfer agents should be added.

6 Conclusion

In this paper I have presented the results of applying the NIST/ECMA framework reference model, and, to a lesser extent, the PSESWG environment reference model, to the description of a 23,000 line Pascal system. The models were an effective and interesting way to describe this system. All the services provided by SUPPORT are developed within one or the other of the models. I expect as these models become better known they will form a set of informal requirements upon which tool vendors will base future products. While the model was never meant to be such a requirements document, its existence may lead towards the standardizing of many future tools and thus possibly allow for greater interoperability among different vendor products.

However, certain features of the model prevented a precise description of the set of services relative to the mapped system. The various committees involved in these models are in the process of revising their models, and it is hoped that mappings, such as this one, can be used to affect future changes.

Going back to our original goal of looking at models of *integrated* SEEs, the current set of models provides little help. Integration is often viewed through three integrational attributes of presentation (e.g., user interface), data (e.g., repository) and control (e.g., process management) [9]. Use of a repository allows a common schema to be shared among tools, but it does not force such a sharing. As already stated, there is a need for specifications like ASIS or Diana as mechanisms to insure the passing of common data among the tools of the environment. Similarly, user interface characteristics need more than a common window system in order to create the common "look and feel" of the integrated interface. Such technology is currently outside of the models and is currently under investigation.

Thus one must be careful in using the results of any one mapping. They provide valuable data in comparing systems or in determining the functionality provided by a system currently in use. However, the models are still in their infancy and requires additional development before they fully characterize an integrated software engineering environment.

References

[1] Brown A. W., A. N. Earl, and J. McDermid, *Software Engineering Environments*, McGraw Hill International (1992).

[2] ECMA, A Reference Model for Frameworks of Computer Assisted Software Engineering Environments, ECMA TR/55 (December, 1990).

[3] OSI, OSI: Connection Oriented Transport Protocol Specification, TC97/SC6, IS 8073.

[4] NIST, Reference Model for Frameworks of Software Engineering Environments, Special Publication 500-201, Natl. Inst. of Stnds and Tech., (December, 1991) (Also ECMA TR/55, Edition 2).

[5] Guide to the POSIX Open System Environment, POSIX P1003.0, Draft 15 (June, 1992).

[6] Reference Model for Project Support Environments, NGCR PSESWG Draft version 0.9 (February, 1993).

[7] Thomas I., PCTE Interfaces: Supporting Tools in Software Engineering Environments, *IEEE Software* 6(6):15-23 (1989).

[8] Thomas I. and B. Nejmeh, Definitions of Tool Integrations for Environments, *IEEE Software* 9(2):29-35 (1992).

[9] Wasserman A. I., Tool Integration in Software Engineering Environments, in *Software Engineering Environments,* F. Long (Ed.), *Lecture Notes in Computer Science*, 467, Springer Verlag, Berlin (1989) 137-149.

[10] Zelkowitz M. V., L. Herman, D. Itkin and B. Kowalchack, A Tool for Understanding Program Execution, *Journal of Pascal, Ada and Modula-2* (May, 1989) 12-20.

[11] Zelkowitz M. V., B. Kowalchack, D. Itkin and L. Herman, Experiences Building a Syntax-directed Editor, *Software Engineering Journal* (November, 1989) 294-300.

A Framework reference model

The framework reference model [4] consists of 48 service groups in six broad categories:

1. Object Management Services

These services define the definition, storage, maintenance, management, and access of object entities and the relationships among them.

Metadata. Definition and maintenance of metadata (e.g., schemas) according to a supported data model.

Data Storage and Persistence. The definition, control and maintenance of data.

Relationship. Defining and maintaining relationships among objects.

Name. The relationships between external names and internal object identifiers.

Distribution and Location. Management and access of distributed objects.

Data Transaction. Defining and enacting transactions.

Concurrency. Insuring concurrent access to the object management system.

Operating System (OS) Process Support. Defining OS processes as objects.

Archive. Transferring information to off-line media and vice-versa.

Backup. This restores the development environment to a consistent state after any media failure.

Derivation. Definition and enactment of derivation rules among objects.

Replication. Explicit replication of objects in a distributed environment.

Access Control. Rules by which access to SEE objects may be granted to or withheld from.

Function Attachment. Attachment of functions to object types.

Common Schema. This provides a canonical schema of the objects and process descriptions.

Version. Create, access, and relate versions of objects and configurations.

Composite Object. This manages composite objects.

Query. This is an extension to the data storage service's read operation.

State Monitoring and Triggering. This enables the definition, specification and enaction of database states and state transformations.

Sub-Environment. This enables the definition, access, and manipulation of a subset of the object management model.

Data Interchange. The two-way translation between data repositories in different SEEs.

2. Process Management Services

The unambiguous definition of software development activities across total software lifecycles. The services here are:

Process Definition. The creation of a library (repository) of process assets, each of which may be a complete process, a (sub)process (or process element), or a process architecture.

Process Enactment. The enactment by process agents that may be users of the SEE or machines.

Process Visibility and Scoping. Several enacting (sub)processes may cooperate to achieve the goal of a higher level process or process system.

Process State. Certain changes in the enactment state of a process may be defined as events and may act as conditions or constraints affecting other processes.

Process Control. A process being enacted may be managed.

Process Resource Management. Process agents may be assigned to enact various processes and process elements.

3. Communication Service

Communication Service needs to provide two-way communication among the components of an SEE.

4. User Interface Service

The User Interface services provide the connection between the user and programs executing in the environment.

User Interface Metadata. This defines the schemas needed to support the user interface.

Session. Initiation of a session between the user and the environment.

Security. This mediates between user views and actions and the security policies.

User Interface Name and Location. This permits the framework to manage multi-user and multi-platform environments.

Application Interface. Data transfer capabilities into and out of the tools and environment to the end user.

Dialog. Integrity constraints between the user and the framework.

Presentation. Low-level manipulation of display devices by the user interface.

Internationalization. Alternative formats depending upon local conditions (local character codes, time formats, etc.).

User Assistance. Consistent feedback from various tools to the user for help and error reporting.

5. Policy Enforcement Services

The reference model uses the term "policy enforcement" to cover the similar functionality of security enforcement, integrity monitoring, and various object management functions such as configuration management. The set of services is:

Mandatory Confidentiality. These are policies established by an administrator concerning access to the information contained in an object.

Discretionary Confidentiality. These are policies established by a user concerning access to the information contained in an object.

Mandatory Integrity. This is defined as the protection of objects from unauthorized or unconstrained modification.

Discretionary Integrity. Discretionary integrity controls are implemented by the user for all functions defined for discretionary access controls.

Mandatory Conformity. These are the result of automation of operational models.

Discretionary Conformity. Individual users would use conformity enforcement to structure their own work environment.

6. Framework Administration Services

An SEE framework needs to be administered because its precise configuration may be constantly changing to meet the changing needs of the software development enterprise.

Tool Registration. This provides a means for incorporating new tools into an environment.

Resource Registration and Mapping. This is the service necessary for the management, modelling, and control of the physical resources of the environment.

Metrication. This provide the means to determine the productivity, reliability, and effectiveness of a framework and of the environment built on it.

User Administration. This provides the ability to add users to an environment.

Self-Configuration Management. This supports the existence of many simultaneous resident configurations of a framework implementation.

B SEE reference model

The PSESWG reference model [6] consists of the following 54 service classifications:

1. Technical Engineering Services

These services include:

Life-cycle process engineering services. Process Definition, Process Library, Process Exchange, Process Usage.

System engineering services. System Requirements Engineering, System Design and Allocation, System Simulation and Modeling, System Static Analysis, System Testing, System Integration, System Requirements Validation, System Re-engineering, Host-Target Connection, Target Monitoring, Traceability.

Software engineering services. Software Requirements Analysis, Software Design, Software Simulation and Modeling, Code Verification, Software Generation, Compilation, Debugging, Software Testing, Software Static Analysis, Software Build, Software Reverse Engineering, Software Re-engineering, Software Traceability.

2. Technical Management Services

These services include:

Configuration management services. Version management, Change management.

Reuse management services.

Metrics services.

3. Project Management Services

These services include:

Project management services. Scheduling, Estimating, Risk analysis, Tracking.

4. Support Services

Common support services. Text processing, Numeric processing, Figure processing, Audio and Video processing.

Publishing service.

Presentation preparation service.

User communication service. Mail, Bulletin Board, Conferencing, Calendar, Annotation.

Administration services. Tool installation, PSE user and role management, PSE resource management, PSE status monitoring, PSE diagnostic, PSE interchange, PSE user access.