

Technical Report

**NIST SP 500-213
CMU/SEI-93-TR-23
November 1993**

Reference Model for Project Support Environments (Version 2.0)

(Jointly Published as CMU/SEI Technical Report 93-TR-23)



NGCR Project Support Environment Standards Working Group

edited by

Alan Brown, David Carney, Patricia Oberndorf,
Software Engineering Institute

Marvin Zelkowitz,
National Institute of Standards and Technology

Approved for public release.
Distribution unlimited.

National Institute of Standards and Technology
Gaithersburg, Maryland 20899
and

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Contents

Preface	ix
Acknowledgments	xi
1 BACKGROUND	1
1.1 Project Support Environment Standards Working Group	1
1.2 Approach	2
1.3 Domain of Interest	2
1.4 Scope of the Model	3
1.5 Types of Project Support	4
2 DESCRIPTION OF THE MODEL	5
2.1 Key Concepts and Terms	6
2.2 The Reference Model	7
2.2.1 Description of End-User Services	9
2.2.2 Description of Framework Services	10
2.3 Discussion of the Model	10
2.3.1 Conceptual Models vs. Actual Environments	10
2.3.2 Rationale for the Groupings in the Model	10
2.3.3 Place of the Target System in the Model	12
3 NOTES ON READING THE SERVICE DESCRIPTIONS	13

3.1	On the Relationships Dimension	14
3.2	On Other Aspects of PSE Services	15
4	TECHNICAL ENGINEERING SERVICES	17
4.1	System Engineering Services	18
4.1.1	System Requirements Engineering Service	18
4.1.2	System Design and Allocation Service	19
4.1.3	System Simulation and Modeling Service	20
4.1.4	System Static Analysis Service	21
4.1.5	System Testing Service	22
4.1.6	System Integration Service	23
4.1.7	System Re-engineering Service	24
4.1.8	Host-Target Connection Service	24
4.1.9	Target Monitoring Service	25
4.1.10	Traceability Service	26
4.2	Software Engineering Services	27
4.2.1	Software Requirements Engineering Service	27
4.2.2	Software Design Service	28
4.2.3	Software Simulation and Modeling Service	29
4.2.4	Software Verification Service	30
4.2.5	Software Generation Service	31
4.2.6	Compilation Service	32
4.2.7	Software Static Analysis Service	33
4.2.8	Debugging Service	34
4.2.9	Software Testing Service	35
4.2.10	Software Build Service	36
4.2.11	Software Reverse Engineering Service	37
4.2.12	Software Re-engineering Service	38

4.2.13	Software Traceability Service	39
4.3	Life-Cycle Process Engineering Services	40
4.3.1	Process Definition Service	41
4.3.2	Process Library Service	42
4.3.3	Process Exchange Service	42
4.3.4	Process Usage Service	43
5	TECHNICAL MANAGEMENT SERVICES	45
5.1	Configuration Management Service	45
5.2	Change Management Service	47
5.3	Information Management Service	48
5.4	Reuse Management Service	49
5.5	Metrics Service	50
6	PROJECT MANAGEMENT SERVICES	53
6.1	Planning Service	54
6.2	Estimation Service	55
6.3	Risk Analysis Service	55
6.4	Tracking Service	56
7	SUPPORT SERVICES	59
7.1	Common Support Services	60
7.1.1	Text Processing Service	60
7.1.2	Numeric Processing Service	61
7.1.3	Figure Processing Service	62
7.1.4	Audio and Video Processing Service	63
7.1.5	Calendar and Reminder Service	64
7.1.6	Annotation Service	65
7.2	Publishing Service	65

7.3	User Communication Services	68
7.3.1	Mail Service	68
7.3.2	Bulletin Board Service	69
7.3.3	Conferencing Service	70
7.4	PSE Administration Services	71
7.4.1	Tool Installation and Customization Service	71
7.4.2	PSE User and Role Management Service	72
7.4.3	PSE Resource Management Service	73
7.4.4	PSE Status Monitoring Service	74
7.4.5	PSE Diagnostic Service	74
7.4.6	PSE Interchange Service	75
7.4.7	PSE User Access Service	75
7.4.8	PSE Instruction Service	76
8	FRAMEWORK SERVICES	79
8.1	Object Management Services	80
8.2	Process Management Services	83
8.3	Communication Service	84
8.4	Operating System Services	85
8.5	User Interface Services	86
8.6	Policy Enforcement Services	87
A	EXTENDED DEFINITIONS OF KEY TERMS	89
B	COMMON PROJECT ACTIVITIES AND THEIR RELATION TO REFER- ENCE MODEL SERVICES	93
B.1	Management Activities	93
B.1.1	Acquisition Management	93
B.1.2	Project Management	94
B.1.3	Quality Assurance	95

B.2 Engineering Activities	95
B.2.1 System Engineering	95
B.2.2 Software Engineering	96
B.2.3 Process Engineering	96
B.3 Supportability Activities	96
B.3.1 Logistics Support	96
B.3.2 Operation and Maintenance	97
C RATIONALE	99
D ABBREVIATIONS and ACRONYMS	103
E REFERENCES	107
INDEX	108
SUBMISSION OF COMMENTS	113

Comments on this document are welcome. See the last two pages of this document for information about submitting comments.

Certain commercial products are identified in this report. Such identification does not imply recommendation or endorsement by the National Institute of Standards and Technology and the Software Engineering Institute, nor does it imply that the product, publication, or service identified is necessarily the best available for the purpose.

Preface

The objective of the Next Generation Computer Resources (NGCR) program is to restructure the Navy's approach to acquisition of standard computing resources to take better advantage of commercial advances and investments. It is expected that this new approach will result in reduced production costs, reduced operation and maintenance costs, and more effective system integration. The program revolves around the selection of commercially-based interface standards in six areas: multi-system interconnects, multiprocessor interconnects, operating systems, database management systems, project support environments, and graphics standards.

The working group concentrating on project support environment standards is the Project Support Environment Standards Working Group (PSESWG, pronounced "peace-wig"). Like the other NGCR working groups, the goal of the PSESWG is to establish standards for interfaces; the particular domain of interest for the PSESWG is project support environment interfaces. As an initial step toward this goal the members of the working group have produced this Reference Model for a Project Support Environment (PSE). The first three chapters contain a general description of the model. The remaining chapters provide detail of individual aspects of the model.

In releasing this document, there is no intention of providing a model to which any environment might "conform." The reference model is a way of expressing an understanding of the functionality of a populated environment. It is not an architectural description to be used in implementing an environment.

Acknowledgments

This document has been developed with the help of a large group of people who participated in quarterly meetings of the NGCR PSESWG during 1991 and 1992. The co-chairs of this working group were LCDR Vinnie Squitieri and Patricia Oberndorf. Members of this working group included:

Carole Amos, Todd Barborek, Dennis Barney, Jerry Brookshire, Alan Brown, D. Bruce Macindoe, David Carney, Peter Clark, Geoff Clow, Douglas Cook, Charlotte Crawford, Hugh Davis, Anthony Earl, Michael Edwards, Bob Ekman, Peter Feiler, James Ferguson, Thomas Grobicki, Dick Grote, Stuart Jeans, George Hacken, Barbara Haleen, Bob Hanrahan, Hal Hart, Richard Hawkes, Henry Heffernan, Bob Hokanson, Stu Jeans, Judy Kerner, Tammy Kirkendall, Joe Lomax, Monte Luhr, Steve Lyda, Brad Lyon, Joyce Lyttle, Zyg Martynowicz, John McGregor, Charles McPherson, Jim Milligan, Les Mopps, Ed Morris, Bob Munck, Philip Nau, Patricia Oberndorf, Kathy O'Toole, Bob Page, Judi Peterson, Richard Randall, Jim Reed, Judy Ryerson, Michael Shapiro, Mike Snodgrass, Vinnie Squitieri, William Sudman, Linwood Sutton, Ramiro Valderama, Rosa Weber, Tom Wheeler, William Wong, and Marvin Zelkowitz

The original conceptual basis of the reference model came from Peter Feiler, who also participated in the earliest stages of PSESWG. The principal editors of this document were Alan Brown, David Carney, Patricia Oberndorf, and Marvin Zelkowitz, who are also responsible for the text of the first three chapters

The concept for the “prism” drawing was a particular contribution of Michael Shapiro.

Many valuable contributions to the body of the document were made by: Dennis Barney, Peter Clark, Geoff Clow, Barbara Cuthill, Bob Ekman, Peter Feiler, Hal Hart, Bob Hokanson, Jim Milligan, Carol Morgan, Bob Munck, Carl Schmiedekamp, Michael Shapiro, Bill Sudman, and Rosa Weber. The Reference Model has also benefitted from the valuable comments made by other reviewers, including: Ger van den Broek, Anthony Earl, Herm Fischer, Alex Lewin, Maria H. Penedo, Ian Simmonds, and Ian Thomas.

This list of contributors was compiled from various sources; if any names of contributors have been accidentally omitted, the oversight is deeply regretted.

Chapter 1

BACKGROUND

The U.S. Navy has embarked on the Next Generation Computer Resources (NGCR) program to fulfill its need for standard computing resources. The program revolves around the selection of interface standards in six areas. The interface standards will be based on existing industry standards with multi-vendor support. The objective is to restructure the Navy's approach to take better advantage of commercial advances and to reduce cost and duplication of computer resources. This document is part of the NGCR program.

1.1 Project Support Environment Standards Working Group

One of the areas chosen by NGCR for interface standardization is that of project support environments (PSEs). The initial focus for the PSE Standards Working Group (PSESWG) is to identify areas in support environments that are in need of standardization and for which industry accepted standards may be available within the NGCR's timeframe. The primary goal of the PSESWG is to provide an interface standard that can be used by project managers as an aid in procuring or assembling a Project Support Environment (PSE) for a particular project or organization. This standard will itself consist of several interface standards that have been chosen for their compatibility and consistency and their ability to support a wide range of project support environment needs. This standard will use industry standards where possible, promoting use of commercial off-the-shelf (COTS) and government off-the-shelf (GOTS) products.

The first step towards this goal for PSESWG is the establishment of a reference model that describes the full scope of functionality that is expected of a PSE. This reference model will provide the basis for:

- Determination of and examination of interfaces for which standards might be included in

the final PSESWG standard.

- Identification of requirements for interfaces which might be beneficial to standardize but for which no industry standardization activity can be identified.
- Consensus throughout the environments community.

While there are several other reference model activities that are relevant to this goal, none individually has the scope that is required nor provides a definition of the concepts at a suitable level of abstraction. Thus the reference model presented in this document is new, although it builds on those other reference models. The PSESWG activity is being coordinated with those other activities whenever possible.

1.2 Approach

Prior to developing this reference model, a large collection of existing environment efforts and models was inspected. This included (but was not limited to) the Software Technology for Adaptable, Reliable Systems (STARS) program, the National Institute of Standards and Technology (NIST) Integrated Software Engineering Environment (ISEE) working group, the European Computer Manufacturers Association (ECMA) TC33 Task Group on the Reference Model, the Ada Joint Program Office Evaluation and Validation Team, the Air Force Software Life Cycle Support Environment (SLCSE) project, Honeywell's Engineering Information Systems (EIS) program, the Conceptual Environment Architecture Reference Model (CEARM) effort, and the standardization committees within IEEE and ANSI for POSIX and for CASE Tool Integration Models (CTIM). The products of those efforts have been analyzed and many valuable aspects have been combined and abstracted.

1.3 Domain of Interest

The approach of this model is most directly comparable to the approach evidenced in the POSIX Open Systems Environment and the NIST/ECMA Reference Model for Frameworks of Software Engineering Environments. However, while both of these have a similar approach, they have different domains of interest. For POSIX that domain is Open Systems Environments,¹ and for the NIST/ECMA reference model it is the domain of PSE frameworks that support software engineering. The domain of the PSESWG reference model encompasses both the POSIX and NIST/ECMA domains. Because their approaches are so similar, PSESWG has made direct use of both models as components of our reference model. This approach led to the realization that since both efforts developed relatively independently of each other, there are numerous small (but at times critical) inconsistencies between them that must be addressed. Members of NGCR PSESWG have been active in helping the two communities to resolve these inconsistencies.

¹Note that this use of the term "environment" is quite different from the sense in which it is used in the phrase "software engineering environment."

The domain of the PSESWG reference model also encompasses domains of interest that are not addressed in the work of either POSIX or NIST ISEE. Numerous specifications and technical reports, describing actual or proposed products, tools and standards, were examined. While some of these provided valuable ideas for the writing of this document, PSESWG's need for greater breadth and scope required the development of a different model for a complete, populated PSE. Thus the majority of the ideas presented in this model are original and are not derived from any earlier efforts.

Finally, the approach is explicitly aimed at establishing a conceptual basis for an environment, not at standardizing any particular environment product; our model must be viewed in this light. This approach is in contrast to many current users of a given environment who, if confronted with the question "What is your environment like?", would reply by listing the available tools. The basic premise of the NGCR program is to standardize on interfaces rather than products. Thus, while tools can help to understand the interfaces on which they depend, they are not central to this reference model, and there is no part of the intended results of the PSESWG activities that involves choosing a standard toolset.

1.4 Scope of the Model

The purpose of the reference model is to describe environments that support projects that engineer, develop and maintain computer-based systems. There are many varieties of such projects. They can comprise the work of several dozens of people or can be a solitary effort. They can be geographically dispersed or concentrated. They also can be institutionally dispersed, sharing people and facilities of several organizations, or concentrated within a single organization. Projects can have widely divergent degrees of automated support. Lastly, the nature of projects may be essentially exploratory, developmental, or maintenance, or may encompass all of these. Yet common to projects of interest to PSESWG is a set of important characteristics:

- Their province is the exploration, engineering, development, or enhancement of a computer-based system.
- They require some mature form of management.
- There is computer-based support for the project.
- There is computer-based support for communication during the project's execution.
- There are several stages within the life of the project, often encompassing various engineering activities.

These characteristics do not uniquely apply to software engineering projects, but include projects involving hardware and firmware, systems engineering, etc. These characteristics are also not peculiar to Navy or to DoD projects, but are typical of engineering projects in general.

Although a project support environment can be either automated or manual, the scope of this reference model is a computer-based support environment. This scope can be further articulated

by distinguishing between different aspects of the automation. For instance, various project support capabilities alluded to above can be provided on PCs, on workstations, on mainframe computers, or on networks involving these. The scope of this reference model encompasses all these. To the greatest extent possible the concentration has been on capabilities that are common to all, not applicable to only one.

1.5 Types of Project Support

Projects require many types of support. Examination of the processes that projects use provides important information on the PSE support that may be required. The functions of projects that can be supported by PSEs can be grouped within four major categories:

- technical engineering functions (e.g., system design, simulation)
- technical management functions (e.g., reuse management, configuration management)
- project management functions (e.g., resource scheduling, project tracking)
- support functions (e.g., editing, maintenance of the support facility)

While details about these categories might be debated,² there is probable agreement that they represent the types of support functions that projects may require. And given the extent and complexity of this area, it is probably impossible to find any set of categories that will find universal agreement. The Working Group has therefore chosen a set of categories that will be most useful as a means toward its principal goal, namely, selecting interface areas for standardization.

²For example, some might believe that the “management functions” extend beyond project budgeting to such accounting functions as payroll; others will disagree. Post-deployment logistics presents a more difficult example. The purpose of this list is to convey the general scope; the services described in chapters 4 through 8 provide more detail.

Chapter 2

DESCRIPTION OF THE MODEL

This chapter first establishes the basic premise of the reference model, then describes the model itself, and lastly discusses several concepts central to an understanding of it.

The reference model is a conceptual description of the functionality provided by a project support environment. This description is general and is bounded neither by a particular application domain nor by any specific lifecycle paradigm for a development project. This is in contrast to an actual implemented environment that is constructed of particular components (i.e., software and hardware) and that typically does reflect a chosen lifecycle paradigm, at least implicitly.

The distinction between conceptual and actual is of fundamental importance. The conceptual viewpoint that governs this reference model provides an abstract description of the functionality expected in a PSE. An actual viewpoint would describe a particular realization of the conceptual view in terms of a PSE architecture with specific tools and standards. There is a mutually reflective relationship between the conceptual and the actual views, i.e., between this PSE reference model and existing environments: one may either consider the model to be abstracted from many environments or regard a particular environment as a realization of the model.

Figure 2.1 illustrates this distinction. The left-pointing arrow illustrates the activity of studying several existing environments to derive information for the model. The right-pointing arrow shows how a particular environment could be a realization of the model. One benefit of this approach is that it provides a common means of describing environments (e.g., “How is SLCSE

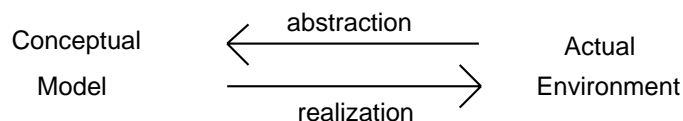


Figure 2.1: Conceptual and Actual Distinction.

different from EAST?”).³ This further provides an ongoing validation of the model; it is a necessary attribute that the reference model provides an accurate reflection of technology that exists.

2.1 Key Concepts and Terms

There are several key concepts and terms used in the Reference Model. This section provides an overview of them and their interrelationships. These terms are more fully described and defined in Appendix A. These key terms are indicated below by italics. It should be noted that some of these concepts are not amenable to simple definition, either because the term is broadly applicable, forcing description rather than definition, or because the term currently has conflicting meanings in the environments community. It is hoped that this section of the Reference Model may help resolve some of this confusion.

An *Environment* is a collection of software and hardware⁴ components; there is typically some degree of compatibility that renders these components harmonious. One can describe an environment using the contrasting viewpoints of conceptual vs. actual; or in a slightly different way, one can describe an environment in terms of the way it supports human activities.

When described from the conceptual point of view, an environment’s capabilities are referred to as *Services*, which are abstract descriptions of the work done. Some of these services are of direct interest to an end-user (e.g., an engineer, manager, or secretary directly participating in the execution of a project) while others comprise an underlying infrastructure, or *Framework*, comprised of relatively fixed capabilities that support processes, objects, and user interfaces.

When described from the opposite, or actual view, i.e., when a realized environment is considered, the components that directly support an end-user are generally called *Tools*. Although no single definition for “tool” will suffice, that of the IEEE Glossary⁵ is useful: a computer program used to help develop, test, analyze, or maintain another computer program or its documentation. As in the conceptual view, the components that comprise an actual infrastructure are referred to as the *Framework*. The same term, framework, is thus used in both a conceptual and an actual sense, and its precise meaning depends on the context.

Finally, when an Environment is considered from the vantage point of how it supports human activities, then either the environment will provide a *Service* to a human user or a human user will perform some *Task* with the aid of the environment. For instance, one can speak of the *task* of testing software, or of using a software testing *service*.

These different views of an environment result in subtle differences in the meanings of key terms. In particular, there is a slightly different meaning for service when it is contrasted with tool and when it is contrasted with task. In the first case, a tool is an actual realization of one or more conceptual services. While there is no strict correlation between tool and service (because one

³Explanations of all acronyms are provided in Appendix D.

⁴For the purposes of this document, PSESWG concentrates on the software components of an environment.

⁵*IEEE Standard Glossary of Software Engineering Terminology*, IEEE Std 610.12-1990.

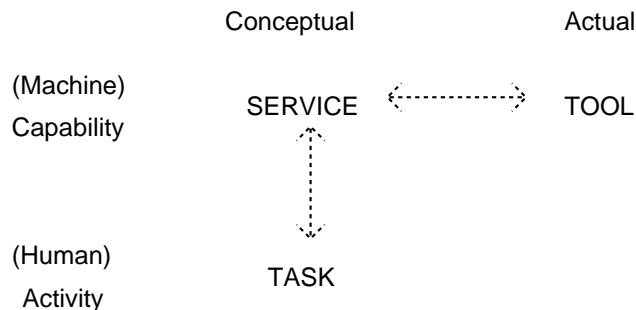


Figure 2.2: Relationship between Service, Tool, and Task.

tool may realize many services, or one service may be realized by many tools), there are relatively straightforward correlations between tools’ functionalities and service descriptions. In the second case, a task and a service provide complementary views of the same activity. For instance, one might consider that the environment provides some capability (e.g., an environment’s testing service); or one might consider that a human user performs some task using the environment (e.g., the human activity of testing). Whichever view one takes, both refer to the same notion, e.g., a human using a piece of software to test the output of an engineering process.

In brief, services are the capabilities of the environment, tasks make use of and provide context for those capabilities, and tools are the actual executable software components. Figure 2.2 illustrates the distinction between these concepts. *Service* can be contrasted with *Tool* along an axis of Conceptual vs. Actual, or it can be contrasted with *Task* along an axis of Capability vs. Activity.

2.2 The Reference Model

The PSE reference model is a catalog of service descriptions spanning the functionality of a populated environment. The service descriptions are grouped by several different categories, including degrees of abstraction, granularity, or functionality. The highest-level division classifies services either as end-user or framework services. When realized (i.e., when implemented by tools), the end-user category includes services that relate to direct support for the execution of a project, e.g., that would be needed by engineers, managers, and secretaries. By contrast, framework services (when realized) more often pertain to users who facilitate, maintain, or improve the operation of the computer system itself (e.g., a human user performing such tasks as tool installation) or are used directly by other services in the environment. As a heuristic for understanding the differences between framework and end-user services, framework services in actual environments tend to be understood by studying “program logic manuals” or by reading the source program and looking for procedure invocations. On the other hand, end-user services are recognized by reading user manuals for the actual environment. However, these rules are only implicit and not part of the model itself. It is certainly reasonable for a user command to

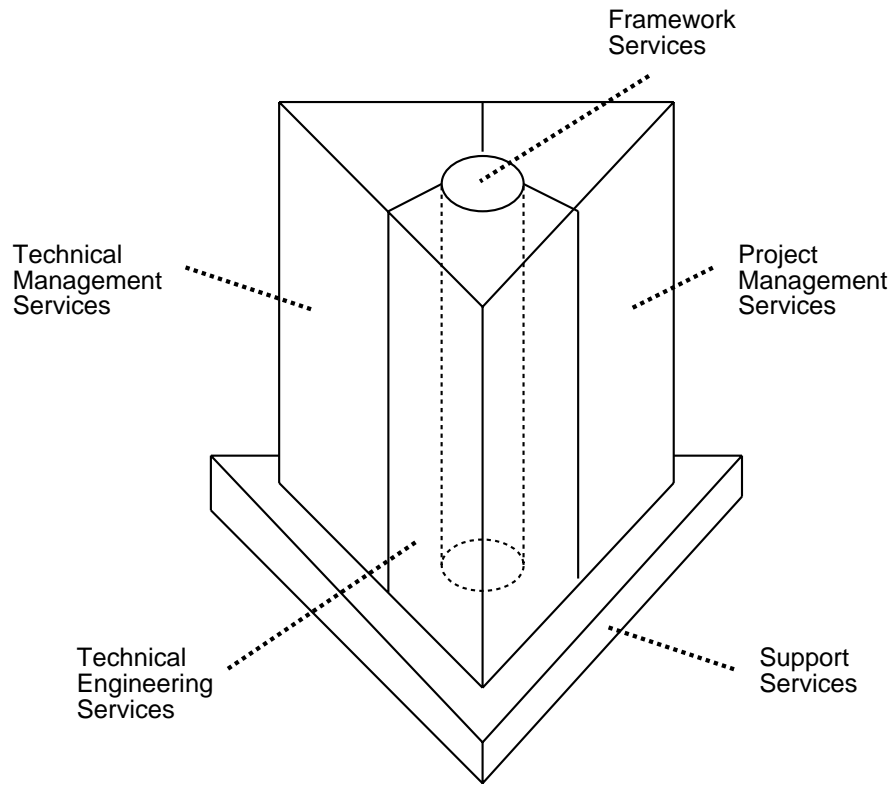


Figure 2.3: An Illustration of Service Groups.

have direct access to a framework facility and it is also possible for an internal function call to have access to an end-user service.

End-user services are further subdivided into Technical Engineering, Technical Management, Project Management, and Support services. The first three of these groups partition the execution of a project into engineering, management, and a middle category that partakes of both. The fourth group, Support services, is orthogonal to the other three, since it includes capabilities potentially used by all users, such as word processing, mail, and publication.

Figure 2.3 illustrates the logical relation of these service groups. Framework services form a central core with a potential relationship to all other services in the environment. Support services underlie the other end-user services. The remaining three groups, Technical Engineering, Technical Management, and Project Management, surround the Framework services and make use of the Support services. In addition, services from these three groups may have relationships with each other.

It is not the intention that the boundaries of the parts of this drawing explicitly indicate interfaces, since this figure is drawn at the level of service *groups*, not of individual services. Thus, it must be stressed that while a drawing such as this attempts to suggest in a very general manner

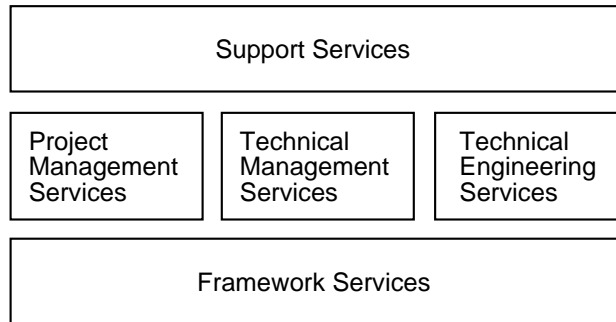


Figure 2.4: Another Illustration of Service Groups.

how the high-level service groups relate to each other, there is an express intention to avoid any sort of architectural implication. The Reference Model is a conceptual, not an actual, model, and no architectural choices are intended by this figure. To emphasize this point the same set of service groups, with the same set of potential relationships, could also be illustrated by figure 2.4.

The key point is that the figures are illustrative only and do not in any way connote layering of services, architectural decisions, or implementation choices for an actual environment.

2.2.1 Description of End-User Services

Each of the end-user service categories (Technical Engineering, Technical Management, Project Management, and Support services) is further subdivided by engineering domain, user role, or life-cycle phase.

Technical Engineering services focus on the technical aspects of project development. These services are subdivided by specific engineering domains (e.g., Software Engineering). Within an engineering domain the processes used in the life cycle of a project define a series of tasks, each requiring services for its support. Thus, within the software engineering domain, tasks typically include designing and coding, which require services such as compilation and testing.

Technical Management provides services that are closely related to engineering activities; these services provide a managerial complement to engineering activities in the areas of configuration management, reuse, and metrics.

Project Management services are relevant to the overall success of the enterprise. They include such things as scheduling, planning, and tracking the overall progress of a project.

Support services focus on tasks and activities common among all users of a PSE, regardless of the domain, role, or life-cycle phase in which the activity is taking place. They include a group of common services for information processing, as well as publishing, user communication, presentation, and administration services.

2.2.2 Description of Framework Services

The framework service categories include Operating System, Object Management, Process Management, Policy Enforcement, User Interface, and Communication services. Service descriptions for these groups are abstracted from the reference model developed by ECMA and modified by NIST in the “Reference Model for Frameworks of Software Engineering Environments,” Edition 3, NIST Special Publication Number 500-211, August 1993 [NIST]. In addition, the Framework Administration services that are included here in the chapter on Support services are also abstracted from that document.

2.3 Discussion of the Model

The following sections discuss the conceptual basis of the model and provide a rationale for how the service groupings were decided. A final section discusses how a target system is considered in the Reference Model.

2.3.1 Conceptual Models vs. Actual Environments

Since the reference model is conceptual as opposed to actual, the service descriptions tend neatly to partition the functionalities of a PSE. When an actual environment is examined, however, these neat conceptual groupings are seldom found. Real software components span various service groups, with many components considered to be end-user tools also providing capabilities properly regarded by the Reference Model as framework services. The likelihood of this functional overlap is the reason that a conceptual model is necessary: one of its principal values is that it provides a common conceptual basis against which to examine many different environment implementations. Figure 2.5 illustrates the distinction between conceptual service descriptions, having no duplication of functionality, and a set of actual software components, many of which may overlap in their functional capabilities. As the figure shows, tools may duplicate other tools’ functionality, and a tool often provides both framework and end-user services.

Note that even if actual environments show this mixing of framework and end-user functionality, it is nonetheless true that framework services tend to be a relatively fixed set of infrastructure services found in most environments, regardless of domain or tool content.

2.3.2 Rationale for the Groupings in the Model

In the widest sense, all users of the computer system are ultimately participating in project execution. However, the reference model distinguishes end-user services as those that are directly related to project execution. For example, managing computer memory clearly is necessary to facilitate an eventual (computer) engineering process. However, services that directly support the management of computer memory are conceptually different enough from services that directly

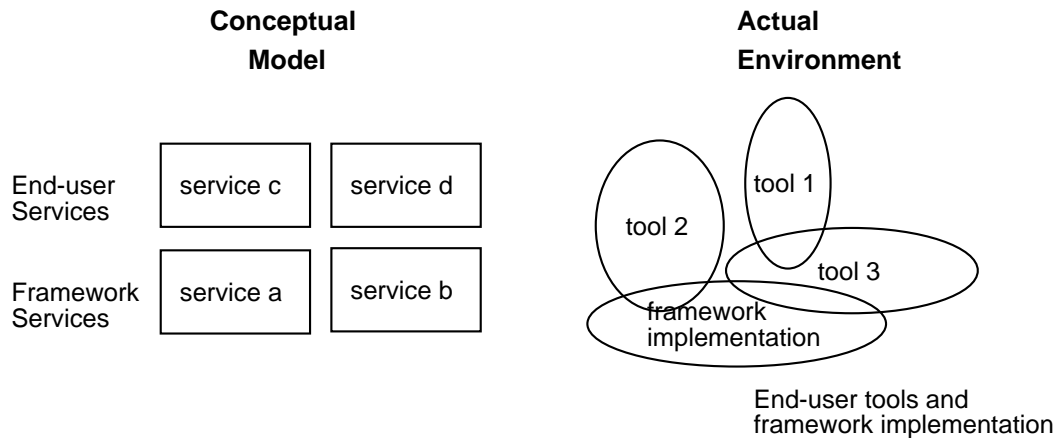


Figure 2.5: Conceptual Service Groups and Actual Software Components.

support high-level engineering activities that the Reference Model considers the classification of memory management appropriately as a framework service and not as an end-user service.

There are other criteria by which services are grouped in the Reference Model. Often a collection of services provides the functionality needed to support a common resource. For example, there is a large group of services in this reference model related to accessing data objects in a common repository. These services are all considered part of the Object Management services group. Since these services are related by creating, accessing, manipulating and using objects from a repository, their classification as a single group allows for a better conceptual understanding of the requirements imposed on any realization of these services and ultimately on any standards that address these services.

Another motivation for grouping some services together might be the roles individuals take in using them. Thus, the activities that go into designing and producing executable source programs will use services that are grouped under the heading of Software Engineering. In this case, the group is determined by the users of the service rather than the management of a common resource.

The boundary between service groups, particularly the boundary between end-user and framework services, is a dynamic one that changes over time. There is a general tendency for greater functionality to be gradually assumed by the underlying framework. For instance, historically most operating systems have included a directory structure and file system for data storage; a relational database is only occasionally added to a basic operating system. In the future, however, relational database functionality may be part of every operating system. It is precisely this growth of functionality that leads toward the notion of “framework,” in contrast to the notion of “operating system.”

2.3.3 Place of the Target System in the Model

While the target system may be the same as the development system, there is no requirement that this be so. The PSE reference model therefore differentiates between the services available on the host system used in the development of computer-based projects and services on the target system upon which the developed project will execute.

Within the NGCR program, some of the details of target system functionality are described elsewhere. One source for these details is the “Operating Systems Standards Working Group Reference Model,” June, 1990 [OSSWG]. Other services, in particular those involving the development system’s connection to the target system and the monitoring of the target system by the development system, are part of the PSE reference model.

Chapter 3

NOTES ON READING THE SERVICE DESCRIPTIONS

The remainder of this document consists of descriptions of the services of a PSE. The descriptions are grouped according to the division already noted, i.e., Technical Engineering, Technical Management, Project Management, Support, and Framework services. Each service group is prefaced with a general overview of the service group, followed by a detailed description of the services. For consistency throughout the model, PSESWG has adopted the convention of the NIST/ECMA Reference Model, by which a service is described through its dimensions:⁶

The term “dimensions” is used for the kinds of description the reference model emphasizes with regard to the services. This is to stress different dimensions are somewhat distinct (if not orthogonal) from one another. That is, if a feature in one service has changed in one dimension, it should not be assumed that changes had to be made to that part of the service in another dimension. Dimensions offer different ways of looking at a whole service....To provide descriptions of services from various perspectives a set of dimensions is associated with each service in the RM.

The eight dimensions are:

Conceptual: the semantics (e.g., functionality) of a service without reference to either its possible implementation or to its relation to other services.

Operations: a subset of the expected operational capabilities of an implementation that realizes the service. This subset is not intended to be complete, but only to provide examples of the typical operations of the service.

Rules: the set of rules that constrain the states the data may reach and the changes to states that operations may make.

⁶NIST, p. 14

Types: the possible types of data or data model used by an implementation of that service, information about those types (for example, metadata), as well as the data (for example, instances of those types) used in an implementation.

External: how the implementation of the service is made available to be used, e.g., by other services, by tools or application programs, or directly by users.

Internal: the place in which to discuss implementation issues such as whether the service might be supplied by the underlying framework.

Relationships to other services: the ways in which implementations of one service might interact with implementations of another service; this may include examples of typical relationships, as well as separation of static and dynamic relationships between services.

Examples: particular examples that implement a service, such as existing standards, interfaces, products, etc.

These dimensions are purely a conceptual means to extract different facets of information about a service in a consistent way. This information may not be apparent from a single detailed prose narrative, hence the choice of using dimensions. The reference model does not prescribe that a system described using the reference model must have every service, nor that every service must be explained from all dimensions. Some dimensions may be more important than others when dealing with particular services. Often, services or dimensions may not apply.

3.1 On the Relationships Dimension

Throughout the Reference Model, relationships between services are usually described by such words as: “This service *may interact with* the XXX service...” This wording has been chosen for several reasons. First, the nature of the relationship may be of many types, including dependency (mutual or otherwise), data sharing, or control. Since different implementations of services might make different choices, the use of “interact” is a neutral way of indicating a relationship without making an implementation choice.

Similarly, the existence of relationships between services is (or is nearly) an architectural decision. Since the Reference Model expressly avoids making architectural decisions, relationships between services are listed only as suggestions and are in no way intended to indicate implementation or architectural decisions.

Finally, almost all of the end-user services of an environment will typically have some relationship to implementations of the framework services, and especially on the object management system; they will also typically have some relationship with the Support services described in Chapter 7. In general, such relationships are noted in the Reference Model only when they might be of particular relevance to the service.

3.2 On Other Aspects of PSE Services

The services included in this reference model tend to be *discrete*, describing functionality that is explicitly invoked and that provides a standalone service. There are, however, some services that are *ubiquitous*, in which case their functionality is implicitly invoked and whose influence permeates many other services. A significant example of ubiquitous services can be seen in the Policy Enforcement services. While these services are described in a separate section (section 8.3), their operation can affect most other services in the model, even though that interaction may not be documented in the service descriptions. Where the influence of security and policy enforcement is documented, it is most often done through either the Rules or the Internal dimension.

A different example of ubiquitous services lies in the question of integration services. Sharing information among the services of a PSE is directly related to the degree of integration that the PSE exhibits. Integratedness is usually described by three integration attributes: data integration (sharing of data objects); presentation integration (a common interface with the user of a PSE); and control integration (appropriate sequencing among several services of the PSE). While integration is recognized as an important aspect of a PSE, there are few actual integration *services* that can be separated as discrete services. Data interchange within the framework's Object Management services (section 8.1) and some of the Life-Cycle Process Engineering services (section 8.2) are known to be related to integration, but the complete set of needed services to appropriately develop a fully integrated PSE is still an important research topic and not fully understood today.

While reading the service descriptions in the chapters that follow, it is important to keep this distinction between discrete and ubiquitous services in mind. In an actual environment, the influence of services such as policy enforcement and security could be relevant to most or all of the other services, and could have a significant impact on the architecture of any system that implements that service.

Chapter 4

TECHNICAL ENGINEERING SERVICES

Technical Engineering services support activities related to the specification, design, implementation, and maintenance of systems. In addition to “traditional” engineering domains, the reference model also considers life-cycle processes to be an area for which an engineering discipline is appropriate, and services related to that domain are included here as well. The following services are defined in this chapter:

- **System Engineering Services**
 - System Requirements Engineering
 - System Design and Allocation
 - System Simulation and Modeling
 - System Static Analysis
 - System Testing
 - System Integration
 - System Re-engineering
 - Host-Target Connection
 - Target Monitoring
 - Traceability
- **Software Engineering Services**
 - Software Requirements Engineering
 - Software Design
 - Software Simulation and Modeling
 - Software Verification
 - Software Generation
 - Compilation

- Software Static Analysis
- Debugging
- Software Testing
- Software Build
- Software Reverse Engineering
- Software Re-engineering
- Software Traceability
- **Life-Cycle Process Engineering Services**
 - Process Definition
 - Process Library
 - Process Exchange
 - Process Usage

There are many other engineering domains, e.g., mechanical, electrical, and manufacturing. Although these are omitted in the present edition of the model, future revisions of the Reference Model may be expanded to include them.

4.1 System Engineering Services

The System Engineering services support projects that engage in substantial development or maintenance activities involving both hardware and software.⁷ These services complement those in the specialized engineering domains (e.g., software engineering) by providing preparation for consistency between those specializations and for integration of their results.

4.1.1 System Requirements Engineering Service

Conceptual: This service provides the capabilities to capture, model, analyze, represent, and refine the system requirements that will ultimately be realized as some combination of software, hardware, facilities, people, and data. This service creates and manipulates representations of system requirements, which may include: system capabilities (such as design and related manufacturing, test, and support capabilities), data elements, internal, external, functional, and physical system interfaces, system software and hardware configuration items that communicate with software components, and system states and modes within which the specific system operates.

⁷There are many parallels between services in this section and services in the Software Engineering section; these parallels may indicate that the same conceptual service is actually common to the two engineering domains. In the present version of the reference model, it was deemed preferable to include these apparent duplications. In future revisions of the reference model, however, some of these services may be collapsed into descriptions of conceptual services common to both System and Software Engineering.

Operations: Examples of requirements engineering operations include:

- Elicit and capture system requirements
- Create, modify, and delete system requirements representations
- Model a system's requirements, including characteristics such as evolving interfaces, performance, and evaluation of risk impact; risks may be based on such factors as changing technology, supportability considerations, or financial risk
- Check consistency of system requirements
- Allocate system requirements

Types: The types used to represent system requirements may take the form of a diagram, textual description, table, icons, graphics, hologram, etc. The representation may be expressed in terms of a modeling notation with explicit rules.

External: This service includes the forms (such as textual, graphical and interchange) in which the service both accepts and provides representations of system requirements, their properties and their interrelationships. The styles of interface (such as command language, query language, message passing, procedure calling and graphical browsing) that the service provides to its operations are another external aspect.

Internal: Checking the consistency of system requirements, both internally (within a set of requirements) and externally (between sets of requirements), may include external constraints, which may impact the implementation of this service. If the service supports simultaneous alternative views of the same information, there may be complexities in the implementation regarding the impact that changes to one representation have on the other representations in order to keep the views of the same information consistent. In a secure PSE, the integrity of the data and traceability between data objects may be of primary significance.

Relationships to other services: This service may interact with the Traceability service, and also with the System Design, Software Design, and Software Requirements Engineering services.

4.1.2 System Design and Allocation Service

Conceptual: This service provides the capabilities to capture, represent, create, analyze, and refine an architectural design of a system's components. System designs describe the interrelationships between system components, including a partitioning of system functionality and constraints between hardware and software. Designs represent objects such as hardware and software components, component invocations, invocation parameters, component composition, data elements, internal and external interfaces, hardware and software configuration items, integrated logistics support elements, and states and modes within which the specific hardware and software sub-components execute.

The results of the system design service include a definition of the hardware and software sub-components that comprise a specific system component. Application of this service can also result in automatic generation of specification documents and work products from source information.

A system design may also accommodate issues related to product development and integrated logistics support.

Operations: Examples of system design operations include:

- Translate requirements into design elements
- Validate consistency of requirements to design
- Create, modify, and delete system design representation
- Define and manage system interface definitions, including those to the support system and their attributes (e.g., reliability, size, power requirements)
- Allocate system component to hardware or software

Types: The types used to represent system designs may take the form of a diagram, textual description, table, graphics, icon, hologram, etc. The representation may be expressed in terms of a modeling notation with explicit rules.

Internal: In a secure PSE, it may be critical to use labels and mandatory access controls to prevent unauthorized formal access to classified design components.

Relationships to other services: Typically, design activities are based on requirements analysis, and this service and the System Requirements Engineering service may have several levels of interrelatedness or may even be realized by a single tool or tool cluster. This service may also interact with the System Integration, System Testing, Traceability, Software Requirements Engineering, Software Design, Software Compilation, and Software Testing services.

4.1.3 System Simulation and Modeling Service

Conceptual: The ability to model a system concept in its entirety before implementation takes place is an important service needed in many phases of project development. This modeling may include both resources needed to create a product as well as the resources needed to deploy, support, and maintain it after development. Project management activities such as scheduling and estimation, as well as system design activities may need the ability to perform tradeoff studies of alternative strategies. During requirements analysis it must be determined if it is feasible to build, operate, and maintain the new product, and during design it must be determined which alternative is most effective. Starting with a high-level description of the component, the System Simulation and Modeling service creates a model of the component at a lower cost and in less time, in order to perform a quick evaluation of the product. In order

to build the model quickly, functionality required of the full product may be sacrificed or the model may execute with less than optimal performance. A design may be modeled to establish such quantified information as predicted and demonstrated failure rates and repair times.

Besides modeling, the terms prototyping or simulation are often used for similar services. A prototype usually has a structure that will be similar to the final product while a simulation is a model where we are mostly interested in the results and not in the structure of the system that produced it. Both terms, however, are often used interchangeably.

Operations: Examples of modeling operations include:

- Build model from requirements
- Execute model
- Compare design relative to required attributes (e.g., cost, performance, supportability)
- Capture results of model

Internal: Simulation models are generally either continuous or discrete. A continuous simulation is usually based upon some mathematical equation where output is produced based upon the independent time variable. In a discrete simulation, a finite state design is usually employed, and the system steps through execution based upon discrete “ticks” of the clock.

Prototyping models usually have a structure that is representative of the feature in the final product that is being studied.

A simulation model and the tool that generates it may require safety features for security purposes. These features might be, for instance, that the model runs to completion or, if an error occurs that the simulation can be rerun either from the start or from before the point of failure.

Relationships to other services: This service may interact with Project Management services and with the System Requirements Engineering and System Design and Allocation services.

Examples: Performance Oriented Design (POD) and Synthetic Environments for Requirements and Concepts Evaluation and Synthesis (SERCES) are examples of system modeling tools.

4.1.4 System Static Analysis Service

Conceptual: The System Static Analysis Service provides for the static analysis of system designs and components in order to determine attributes of the system. Information derived from the service includes:

- Product characteristics, such as component size, function calls, and operations.

- Complexity characteristics, such as cyclomatic complexity, spanning measures, other control flow and data flow measures, and other relationships derived from product characteristic measures.
- Cross reference lists and graphs, such as define/use graphs, call graphs, data flow graphs, and structure charts.

Operations: Examples of System Static Analysis operations include:

- Collect raw statistics from system representations
- Compute complexity measures
- Produce cross reference list
- Graph cross reference list

External: This service (especially cross reference information) is often provided as part of system generation or integration tools; however, it also can be provided by separate static analysis tools.

Relationships to other services: This service may interact with the System Design service, and the Software Static Analysis and Design services.

4.1.5 System Testing Service

Conceptual: This service supports testing of systems. The purpose of testing is to insure that all specifications have been met and that systems are operationally effective and suitable for intended use. Such testing may be intrusive (e.g., accomplished by instrumenting code or hardware elements), non-intrusive (i.e., accomplished by running the system in normal operational configurations or through the use of real-time non-intrusive instrumentation (RTNI) equipment), destructive (e.g., for survivability or ruggedization), or non-destructive (for both intrusive and non-intrusive tests).

Operations: Examples of System Testing operations include:

- Generate system test cases
- Generate test requirements matrix
- Perform system test
- Perform system regression test against all previous test cases
- Analyze completeness of test coverage
- Capture system test results

- Validate system test results against anticipated results
- Produce summary report of results

Internal: In a secure PSE, confidentiality of stored test data may be required if results are sensitive in nature.

Relationships to other services: This service may interact with the System Requirements Engineering, Traceability, Software Test, Configuration Management, and Risk Analysis services. It may also interact with the Host-Target Connection and Target Monitoring services.

4.1.6 System Integration Service

Conceptual: The product of an engineering process is often composed of a number of different pieces, each developed separately. This may be a result of a fundamental difference in nature of some of the pieces of a project (e.g., the hardware and software for an embedded real-time application), a technical distinction between the pieces (e.g., the control software and user interface software), or a pragmatic decision (e.g., to allow a group of people to work concurrently on portions of a large system). In any case, it must be possible to combine those pieces into a product. The resultant product must be identified as a new item of interest, and hence it can be tracked as a significant object from that point on. If the pieces exist in multiple versions, the correct version of each piece must be selected in creating the full product.

A description is required that defines which logical pieces make up a product and how those pieces are related (e.g., chapters in a document, a product-oriented family tree). As some of the pieces may need to be transformed before they are combined, this description may contain details of how that transformation takes place.

Operations: Examples of integration operations include:

- Identify the components and incremental builds that make up a product
- Define the transformations that are required to integrate the components of a product
- Build a product from its components
- Create product
- Audit product

Types: Objects that comprise a product may be CSCIs, textual or graphical components of documents, or technical data packages and parts lists. A product can consist of originated (“source” in its real meaning) objects as well as derived objects.

Internal: The use of labels and mandatory access controls may be necessary to prevent the unauthorized linkage of classified system components in a system build. If the product release

needs to be controlled, the build tape might require a cryptographic checksum to ensure that the build is not corrupted in distribution.

Frequently occurring transformations are often provided as an internal component of an integration service. This helps to simplify the integration descriptions used and to automate the building process.

Relationship to other services: This service may interact with the Configuration Management service, since the pieces used in a product may be part of a new configuration. It may interact with the Traceability service to determine which components need to be processed. It may also interact with the System Design and the Software Build services.

4.1.7 System Re-engineering Service

Conceptual: The System Re-engineering service is required when a system's requirements change. The changes may be related to functionality, performance, reliability, cost; to such factors as obsolescence, nonavailability of parts, changed manufacturing or logistics circumstances; or to the need to take advantage of technological improvement and evolution. This service takes as input an existing design and a new or modified set of requirements and produces a new or modified design according to the changed requirements. The System Re-engineering service encompasses hardware, software, manufacturing, and support elements of a system.

Operations: Examples of System Re-engineering operations include:

- Perform fault analysis and verification
- Analyze impact of modified requirements on existing design and implementation
- Modify design representation
- Analyze impact of new design on existing system components

Internal: In the re-engineering of secure systems, special precautions are likely to be required. These may include labelling and mandatory access controls, maintenance of confidentiality and integrity, and careful auditing of all changes to existing designs and implementations.

Relationships to other services: This service may interact with the System Simulation and Modeling service, the System Static Analysis service, the System Requirements Engineering service, and the System Design and Allocation service.

4.1.8 Host-Target Connection Service

Conceptual: The Host-Target Connection service is required to ensure the ability of a host PSE to communicate with a target system for the purpose of software downloading, system test

or debug, and system monitoring.⁸ The minimum capability for this service is for two one-way links. One is for the host PSE to be able to convey to the target such items as the loadable or bootable software executable image or test and debug commands. The other permits the target to convey to the host PSE the results of test or debug operations and target monitoring information.

Operations: Examples of host-target connection operations include:

- Establish system-to-system communications
- Maintain, control, and relinquish host-target connection

External: The host-target communication can be realized as a direct or indirect link. A direct link might consist of a cable or satellite connection. An indirect link might consist of an agreement on a particular format for floppy disk or magnetic tape.

Internal: In a secure PSE, this service may require authentication, both mandatory and discretionary access controls, careful attention to network security management, and more to ensure that no security breach occurs because of the host-target connection.

Relationships to other services: This service may interact with all other PSE services that rely on the ability to communicate with a target system. These would include at least the System Testing, Target Monitoring, and Software Debugging services.

Examples: A simple example of this service would be for a PSE to generate magnetic tape cartridges in a form for which the target system has a compatible drive. Likewise, when the target needs to convey information back to the host, it generates another magnetic tape cartridge. A more sophisticated capability would use a network link directly to download information to the target, dynamically interacting with it in real time.

4.1.9 Target Monitoring Service

Conceptual: The Target Monitoring service provides the ability of the host PSE to receive and interpret specified execution and performance information from an operational target system.

Operations: Examples of monitoring operations include:

- Specify target system monitoring information

⁸It is assumed, for the purposes of NGCR, that the target systems are built using the other NGCR standards. Eventually, that list of standards will include at least two levels of performance for local area networks, at least two levels of performance for backplane busses, an operating system interface standard, a database management system interface standard, and a graphics interface standards.

- Acquire and accumulate target system monitoring information
- Correlate and analyze target system monitoring information
- Report monitoring information

Types: These would include any types critical to the target system. These could include: frequency or length of calls, missed deadlines, length of queues, CPU time used, dynamic paging activity, number of requests, block sizes, fragmentation, and other types of data.

Relationships to other services: This service may interact with the Host-Target Connection Service for the acquisition of the information reported by the target system.

4.1.10 Traceability Service

Conceptual: The Traceability service supports recording of relationships between artifacts of the development process. These artifacts may be representations of requirements or designs, software items, hardware, test items, etc. The relationships permit other services to perform such operations as verifying existence, establishing dependencies, and similar operations whose aim is establishing and maintaining factors of constraint. This service may be used in maintaining consistency as well as performing change impact analysis.

The Traceability service generally imposes additional properties on the types of object defined for the other development process services. Thus, this service is similar to the framework's Object Management services, but is at a higher level of abstraction, since this service presumes additional semantic information than that present in the OMS.

Environments may provide a traceability service automatically, wherein development process activities inform the traceability service as derivations occur. More loosely coupled systems may require user convention or intervention to record relationships, such as the use of naming schemes that permit the deduction of relationships between artifacts. One common use of this service is to establish that a system's requirements can be traced throughout other stages of the lifecycle process.

Operations: Examples of traceability operations include:

- Create, update, and destroy relationships between two items
- Query current status of relationship or relationship history
- Navigate relationships and items
- Detect and report violations of traceability constraints

Rules: Relationships must point at existing items.

Internal: Fully automated trace recording will likely imply the sharing of schema and a data repository between multiple development process activities. In a secure environment, the integrity of the data links is especially important. Care must also be taken that a user navigating the traceability trees cannot infer information from relationships in the data.

Relationships to other services: This service may interact with most other engineering services. It may also have an important relationship with the framework's Process Support services and with the Lifecycle Process Engineering services.

Examples:

- ORCA (Object-based Requirements Capture and Analysis).
- RETRAC (REquirements TRACeability).

4.2 Software Engineering Services

The services in this category support the specification, implementation, debugging, and maintenance of software.⁹

4.2.1 Software Requirements Engineering Service

Conceptual: This service provides the capabilities to capture, represent, analyze, and refine those system requirements that are allocated to software components. This service creates and manipulates representations of requirements. These may include: software capabilities, data elements, internal and external software interfaces, system software and hardware configuration items that communicate with software components, and system states and modes within which the specific software executes.

Operations: Examples of software requirements engineering operations include:

- Elicit and capture software requirements
- Create, modify, and delete software requirements representations
- Check consistency of software requirements
- Allocate software requirements

⁹There are many parallels between services in this section and services in the System Engineering section; these parallels may indicate that the same conceptual service is actually common to the two engineering domains. In the present version of the reference model, it was deemed preferable to include these apparent duplications. In future revisions of the reference model, however, some of these services may be collapsed into descriptions of conceptual services common to both System and Software Engineering.

Types: The types used to represent software requirements may take the form of a diagram, textual description, physical artifact, graphical computer representation, hologram, etc. The representation may be expressed in terms of a modeling notation with explicit rules.

External: External aspects of this service include the forms (such as textual, graphical and interchange) in which the service both accepts and provides representations of software requirements, their properties and their interrelationships. The styles of interface (such as command language, query language, message passing, procedure calling and graphical browsing) that the service provides to its operations are another external aspect.

Internal: In a secure PSE, the integrity of the data and traceability between data objects may be of primary significance.

Relationships to other services: This service may interact with the System Requirements Engineering and System Design services. It may also interact with the Software Traceability, Software Design, Software Re-engineering, and Software Simulation and Modeling services.

Examples: OOATool and DCDS are examples of tools providing software requirements engineering.

4.2.2 Software Design Service

Conceptual: This service provides the capability to capture, represent, create, analyze, and refine the design attributes of the software components of a system or subsystem. These attributes can be the structure or functionality of the software or other characteristics such as user interface design or performance considerations. Software designs are typically dependent on a set of requirements; they describe interrelationships of software components, including interfaces, invocation parameters, data elements, and the states and modes within which the specific software sub-components execute. The outcome of the software design service includes definition of the software components and subcomponents.

Operations: Examples of software design operations include:

- Translate requirements into design elements
- Create, modify, and delete software design representation
- Validate design to requirements
- Produce structure charts, graphs, screens or other design information from design representation
- Evaluate design representation

Types: The types used to represent software designs may take the form of a diagram, textual description, physical artifact, graphical computer representation, screen display, hologram, etc. The representation may be expressed in terms of a modeling notation with explicit rules.

External: The external dimension includes the forms (such as textual, graphical and interchange) in which the service both accepts and provides representations of software designs, their properties and their interrelationships.

Internal: In a secure PSE, it may be critical to use labels and mandatory access controls to prevent unauthorized formal access to classified design components.

Relationships to other services: Typically, design activities are based on requirements analysis, and this service and the software requirements engineering service may have several levels of interrelatedness or may even be realized by a single tool or tool cluster. This service may also interact with the System Requirements Engineering and System Design services and with the Compilation, Debugging, and Software Testing services. It may also interact with the Software Simulation Service.

This service may also interact with the Software Reverse Engineering and the Software Re-engineering services, both of which have as goals the modification of an existing design into a new design.

Examples:

IDE's Software through Pictures (StP), Teamwork, ObjectMaker, and the Hierarchical Object-Oriented Design (HOOD) method and its associated design tools provide examples of this service.

4.2.3 Software Simulation and Modeling Service

Conceptual: The ability to model a component or software system before implementation is an important service needed in many phases of project development and in many engineering domains. Project management planning services may need simulation and modeling to perform tradeoff studies of alternative strategies. Requirements engineering can use simulation and modeling to determine if it is feasible to build a product to the requirements. Design services can use simulation and modeling to determine the effectiveness of alternative designs with respect to such attributes as user interface characteristics or execution flow.

Starting with a high-level description of the component, the Software Simulation and Modeling service creates a version of it that is less expensive than the desired product and built in less time in order to perform a quick evaluation. In order to build the model quickly, it either sacrifices functionality required of the full product, reduces the capability of the product, or executes with less than optimal performance.

Besides modeling, the terms prototyping, emulation, or simulation are often used for similar services. A prototype usually has a software structure that will be similar to the final product. An emulation tends to be relatively complete, in the sense of a rival, while a simulation is a model where one is principally interested in the results and not in the structure of the software that produced it. All three terms, however, are often used interchangeably.

Operations: Examples of Software Simulation and Modeling operations include:

- Build model from requirements
- Execute model
- Capture results of model

Internal: Simulation models are generally continuous or discrete. A continuous simulation is usually based upon some mathematical equation where output is produced based upon the independent time variable. In a discrete simulation, a finite state design is usually employed, and the system steps through execution based upon discrete “ticks” of the clock.

Prototyping models usually have a software structure that is representative of the feature in the final product that is being studied (e.g., a user interface design similar to the window structure the system will execute in). Often a high level language with easy modeling capabilities but slow execution characteristics is used to build a prototype (e.g., SetL, 4GLs).

A simulation model and the tool that generates it may require safety features for security purposes. These features might be, for instance, that the model runs to completion or, if an error occurs that the simulation can be rerun either from the start or from before the point of failure.

Relationships to other services: This service may interact with the Planning and Risk Analysis services, and with the Software Requirements Engineering, Software Design, and Software Re-engineering services. The System Simulation and Modeling Service provides similar functionality at the system level.

Examples:

- Simula, Simscript, and CPL (Common Prototyping Language) are example simulation and prototyping languages for software systems.
- Menu or screen simulators (DEMO program, TeleUse, Rapid/Use in IDE’s StP) also provide examples of this service.

4.2.4 Software Verification Service

Conceptual: It has long been demonstrated that *a posteriori* testing of software is most effective in showing the presence of errors and not their absence. Software verification uses formal mathematical methods to prove *a priori* that the software must execute according to its specifications. While proving that software does indeed meet its specifications has been shown to be an extremely hard problem, there are many critical applications where the needed reliability of the software simply requires it.

Formal verification first requires that a formal specification of a program be generated and then that a formal model exists that maps between the specification and the eventual design or implementation language. Given these two descriptions, a mathematical proof is generated that the written software and the specification are equivalent.

Operations: Examples of software verification operations include:

- Analyze specifications (for consistency to the formal model)
- Read source component (either source programming language or design language)
- Identify errors (between specifications and verified object)
- Produce summary report

Rules: Verification systems are based upon one of a few formal models:

- Axiomatic models which extend the predicate calculus with programming language constructs.
- Functional and denotational semantic models which assume that programs are mathematical functions with an input and output domain.
- Algebraic models which formally define the interface between program components as mathematical equations.

Relationships to other services: Verification may interact with Software Requirements Engineering, Software Design, Software Testing, and Software Debugging services.

Examples: VDM (Vienna Development Method), based upon denotational semantics, and Z, based upon the axiomatic model, are used to show the equivalence between a specification and a design. Affirm was a research system that used the algebraic model to show equivalence between a specification and a source program. Gypsy is a language that includes a verifier as part of its system.

4.2.5 Software Generation Service

Conceptual: Software generation provides automatic and semi-automatic production of software components using existing components or component templates.

The use of a Software Generation service is most frequently seen in well-defined application areas such as language parser generation, database application generation, and user interface design and production.

Operations: Examples of Software Generation operations include:

- Generate parser from a syntactic language description (e.g., a BNF representation of a language grammar)

- Generate script for the composition and interconnection of software components
- Generate rule-based system from a set of rules
- Generate user interface component for a software system
- Generate schema for database

Internal: In a secure PSE, the Software Generation service must insure data integrity of the descriptions, scripts, rules, schemata, etc. so that the documentation can be trusted to represent the software.

Relationships to other services: This service may interact with the Software Design and Software Compilation services.

Examples:

- Parser generators (LEX, YACC) for producing compilers.
- 4th Generation Languages (4GLs) and application generators provided by many relational database systems.
- IDL tools to generate the I/O for specific data formats.
- Application-specific language generators.

4.2.6 Compilation Service

Conceptual: The Compilation service provides support for the translation and linking of software components written in various programming languages. Source code is created either by means of text processing services, or by the automatic generation services described in the Software Generation service.

The principal outputs from this service are executable programs supporting some target system. Other products of this service may include metrics data and documentation aids such as compilation listings.

Operations: Examples of software compilation operations include:

- Find code and inheritance dependencies among a set of software components
- Preprocess source code to produce modified source code
- Apply macro expansions to source code
- Translate a source program (e.g., Ada, COBOL, C, Pascal, Assembly language) into some target object code language

- Produce report on the translation; this may include source listings of various complexity, including cross-reference data, compilation speeds, CPU usage, etc.
- Link object code into executable images. When intended for use on a remote target, link code into loadable/bootable images.
- Incrementally update compiled system to reflect new changes.

Rules: The Compilation service enforces the rules of the programming languages that it processes.

Types: Source code directly created by a human user is typically written in ASCII text. Source code produced by source code generators may rely on internal data types known to the compiler.

Internal: The compilation of object code must be free from insertion of malicious code. Compilers may not be trusted to the degree necessary for secure systems, but system integrity controls can be used to provide additional protection.

Relationships to other services: This service may interact with the Software Design, Software Generation, Debugging, Software Traceability, and Software Testing services. It may also interact with the System Testing service and the Host-Target Connection service. It may also interact with the Configuration Management, Project Management, and Lifecycle Process Engineering services.

Examples: Examples of compilation services include:

- Compiler systems (including linkers) for standard languages (Ada, COBOL, C, Pascal, etc.)
- Unix's Lint preprocessor

4.2.7 Software Static Analysis Service

Conceptual: The Software Static Analysis service provides for the static analysis, or source code analysis, of software components in order to determine structure within the component. Information derived from the service includes:

- Code characteristics, such as component size, number of statements, statement types, variables, function calls, operations, operands, data types and other programming language-specific data.
- Complexity characteristics, such as cyclomatic complexity, software science measures, spanning measures, other control flow and data flow measures, and other relationships derived from product characteristic measures.
- Cross reference lists and graphs, such as define/use graphs of variables, call graphs of functions and other subprograms, data flow graphs, structure charts, and variable and type definition lists.

- Software characteristics, such as: testability, completeness or consistency, reachability, reusability, and maintainability.

Complexity measurements are based upon various underlying graph models of the source program. Even simple measures, like lines of code, have different interpretations, so comparisons between two tools providing this service must be carefully analyzed before such comparisons are used.

Operations: Examples of Software Static Analysis operations include:

- Collect raw statistics from component
- Compute complexity measures from component
- Produce and graphically represent cross reference lists

Internal: Data is often collected from internally parsed forms of the source program.

Data and the traceability between data objects must be preserved and non-corruptible in a secure PSE.

External: This service (especially cross reference information) is often provided as part of compilation tools; however, it also can be provided by a separate static analysis tool.

Relationships to other services: This service is often provided as part of the Software Compilation service. This service may also interact with the Software Design service, the Metrics service, and with the System Static Analysis service.

Examples: NASA's SAP program for analyzing Fortran code is an example of this service.

4.2.8 Debugging Service

Conceptual: The Debugging service is for the location and repair of software errors in individual software components by controlled or monitored execution of the code. Unlike the the Software Testing service, which determines that an error is present, the Debugging service supports tracking down errors and replacing code.

Operations: Examples of Debugging operations include:

- Instrument source programs by inserting breakpoints, instruction traps, printing out data values, and modifying source text
- Execute programs incrementally

- Monitor and save execution output
- Analyze properties of programs and their current data values

Relationships to other services: This service may interact with the Text Processing service, with the various Software Engineering services such as the Software Design, Compilation, Software Test, and Software Generation services, and with the Host-Target Connection service.

Examples: The Unix *dbx* debugger is an example of this service.

4.2.9 Software Testing Service

Conceptual: This service supports the testing of software systems. Testing is performed on individual software components (unit testing), on collections of software components (integration testing), and on complete software systems (system testing).

A particular situation in which software testing occurs is when the target operating environment for an application is different from the environment on which the application is being developed. In this case, the system's Host-Target Connection and Target Monitoring services will be required for software testing.

Operations: Examples of software testing operations include:

- Generate test cases and test harness. Depending upon the testing method used (e.g., path testing, functional testing, statement coverage, boundary value testing), capabilities may be implemented to analyze source programs and generate such test values.
- Instrument source programs to output test results, depending upon testing method used. For example, each path (or branch or statement) can output data showing that each path was executed.
- Perform tests for resource utilization, reliability, and path and domain selection
- Perform timing analysis and real-time analysis (missed deadlines, deadlock, race)
- Perform mutation analysis
- Perform regression testing of all previous test cases on the tested object.
- Validate test results with expected results

Internal: In a secure PSE, confidentiality of stored test data may be required if results are sensitive in nature.

Relationships to other services: The System Testing Service may be needed to test systems involving both hardware and software components. The Debugging service may be used to repair errors found by testing. The Host-Target Connection service may be used to communicate

testing data with a target system different from the host system being used for development. This service may also interact with the Compilation service, the Configuration Management service, the Build service, and the Lifecycle Process Engineering services. Formal proofs of correctness are handled by the Software Verification service.

Examples: Software TestWorks is an example of this service.

4.2.10 Software Build Service

Conceptual: The product of software development is often composed of a number of components, each developed separately. Size of resulting product, number of personnel assigned to development, schedule, and development method (e.g., top down design, structured design, object oriented design) all influence the development of a software product into a set of separately compilable components. In any case, it must be possible to combine those pieces into a product, often called a release. The resultant product release must be identified as a new item of interest, and hence it can be tracked as a significant object from that point on. If the components exist in multiple versions, the correct version of each must be selected in creating a full release.

In the building of a release it is often possible to perform some transformations to the components automatically before they are combined. For example, the source code is typically converted to object code via compilation before the object code is combined through linking.

A description is required that defines which logical pieces make up a product and how those pieces are related (e.g., phases in a multi-phase program). As some of the pieces may need to be transformed before they are combined, this description may contain details of how that transformation takes place. Specific data, such as default file names, may be added to the product at this time.

The product release can be compared with the build description that was used to derive it. A list of the actual versions of components used in that build should be recorded, together with the operations that were used in transforming the components and deriving the release.

The resultant product release may also be versionable.

Operations: Examples of Build operations include:

- Define the relationships among the components that make up the product
- Transform the components that make up the product
- Build a product from its components
- Create product release
- Audit product release

Types: Objects that comprise a release may be source code, binary code, and textual or

graphical components of documents. A release can consist of originated (“source” in its real meaning) objects as well as derived objects.

Internal: If the software product release needs to be controlled in a secure PSE, the build tape may require a cryptographic checksum of the build to ensure that the build was not corrupted in the distribution of the system.

Frequently occurring transformations are often provided as an internal component of the Software Build service. This helps to simplify the build descriptions used and to automate the building process. Components used by the Software Build service may themselves be collections of objects in an Object Management System.

Relationship to other services: This service may interact with the Configuration Management service, since the pieces used in a build may produce a configuration. It may also use the Software Traceability service to determine which components need to be processed. It may also interact with the System Testing and System Integration services.

Examples: The Unix *make* tool is a well known example for software construction.

4.2.11 Software Reverse Engineering Service

Conceptual: The Software Reverse Engineering service provides the capabilities to capture design information from source or object code and produce structure charts, call graphs, and other design documentation of that information. The goal is to generate a design that represents an existing program which may then be re-engineered (using the Software Re-engineering service) to provide new functionality, perhaps retargeted to execute on a new hardware platform or translated into another source programming language.

It is sometimes necessary to also reverse engineer source code from executable object code. Disassemblers are tools that produce assembly language from such object code, and decompilers produce source programs from such object code.

Decompiling object code that was originally compiled in one language into another is an extremely difficult operation. It is often better to reverse engineer the object code and then use the Software Re-engineering service to produce the code in the new language.

Operations: Examples of Software Reverse Engineering operations include:

- Generate design from source code
- Generate source program from object code

Rules: Abstracting and partitioning the design will require specific rules and methods. These rules will have a significant effect on the usefulness of the design in future re-engineering efforts.

Types: This service produces a new design and uses as input either a source program or an object program. The types of design for reverse engineering are the same as the types used in the Software Design service, such as Data Flow diagrams and ER diagrams.

External: Software code and its language definition are the primary input interfaces to this service. Standard output formats (e.g., the Common Data Interchange Format) are required for sharing a design with other services in the environment

Internal: The most difficult problem in this service is the definition of the boundary of the existing software to be reverse engineered. In the case of very large amounts of code, current reverse engineering techniques would require extensive processing, possibly similar to the processing needed to compile it. This situation implies partitioning of the code space and further implies a partial design as a result.

Relationships to other services: The Software Compilation service may be used to translate a source program into another source programming language. The Software Re-engineering service may be used to modify the reverse-engineered design. The Software Static Analysis service may be used to produce some of the design information, such as structure charts and call graphs. This service may also interact with the Software Design service.

4.2.12 Software Re-engineering Service

Conceptual: The Software Re-engineering service is used when software requirements change. This service takes as input an existing design and a new or modified set of requirements and produces a new or modified design according to the changed requirements. The service may also check that the new set of requirements is consistent with the existing system and may determine the impact of the altered design on the existing set of components. Such concepts as altered functionality, modified performance, and new capacities may also be evaluated. Use of this service may also be appropriate when source code is deemed in need of restructuring for improved maintainability. It may also be used when code is to be translated from one notation into another.

Operations: Examples of Software Re-engineering operations include:

- Revise or restructure existing code
- Perform impact analysis of new design on existing software components
- Translate from one notation into another (e.g., a COBOL-to-Ada translator)

Types: The input to this service is a design, an altered set of requirements, and possibly a set of source code components. The output will be a design. The format of a design may be a diagram, textual description, physical artifact, graphical representation, hologram, etc.

Internal: In the re-engineering of secure software, special precautions are likely to be required. These may include labelling and mandatory access controls, maintenance of confidentiality and integrity, and careful auditing of all changes to existing designs and implementations.

Relationships to other services: This service may interact with the Software Design service to perform some of the design activities associated with re-engineering. It may also interact

with the Software Requirements Engineering, Software Simulation and Modeling, and System Re-engineering services.

4.2.13 Software Traceability Service

Conceptual: The Software Traceability service supports recording of relationships between artifacts of the development process. These artifacts may be representations of requirements or designs, code items, test items, etc. The relationships permit other services to perform such operations as verify existence, establish dependencies, and similar operations whose aim is establishing factors of constraint.

The Software Traceability service generally imposes additional properties on the types of object defined for the other development process services. Thus, this service is similar to the framework's Object Management services, but is at a higher level of abstraction, since this service presumes that additional semantic information is present in establishing the relationships than is present in the OMS.

Environments may provide a traceability service automatically, wherein development process activities inform the traceability service as derivations occur. More loosely coupled environments may require user convention or intervention to record relationships, such as the use of naming schemes that permit the deduction of relationships between artifacts. One common use of this service is to establish that requirements can be traced throughout other stages of the lifecycle process.

Operations: Examples of traceability operations include:

- Create, update, and destroy relationships between two items
- Query current status of relationships
- Query relationship history
- Navigate relationships and items

Rules: Relationships must point at existing items.

Internal: Fully automated trace recording will likely imply the sharing of schema and a data repository between multiple development process activities. In a secure environment, the integrity of the data links is especially important. Care must also be taken that a user navigating the traceability trees cannot infer information from relationships in the data.

Relationships to other services: This service may interact with many process-related services, both at the Lifecycle Process Engineering level as well as at the framework's Process Support level. It may also interact with the System Traceability service. It may also interact with the Software Requirements Engineering and Software Design services.

Examples:

- ORCA (Object-based Requirements Capture and Analysis).
- RETRAC (REquirements TRACeability).

4.3 Life-Cycle Process Engineering Services

The Life-Cycle Process Engineering services support projects in achieving discipline, control, and clear understanding in their life-cycle development processes and individual process steps. The activities of the role of Process Engineer are sometimes shared by various management and technical roles on a project and are sometimes performed by a distinct role (Process Engineer). The services in this section distinguish process-driven (or -managed, -sensitive, -centered, or -controlled) PSEs from collections of project tools. Life-Cycle Process Engineering services include Process Definition, Process Library, Process Exchange, and Process Usage.

The Life-Cycle Process Engineering services differ from the framework's process management services in several ways. At the framework level, the process management services produce and manipulate the basic data needed to define processes. These include: definition of pre- and post-conditions for enactment of processes; definition of project data needed for process enactment; specification of relevant events; and creation of both the basic process elements that define the life-cycle processes and the basic primitives to enact processes. At the end-user level, the Life-Cycle Process Engineering services described in this section use the framework's process management services to define the relationships among the various services in the PSE and

various roles users take in developing a product in order to implement a process for achieving that development.

4.3.1 Process Definition Service

Conceptual: This service provides the capabilities for projects to create, maintain, tailor, adapt, and validate definitions of processes in formal, semiformal, and informal forms. This is a comprehensive service that provides for process definition the analogy of a wide range of systems or software development services, from requirements activities through architecture, design, instrumentation, and verification activities.

A process definition prescribes the interaction between process participants (managers, engineers, etc.), technology (framework services, tools, etc.), and the methods, organizational policies, and procedures used to create the interim and final products that result from the execution of the process.

The features of a process definition notation can lead to the automation of, guidance for, control over, and enactment of the defined process. The process mechanism will be driven by formal detailed specifications of the process that include definitions of the triggers, activities, work products, completion criteria, and other elements of the process (e.g., data schemata for project databases).

Operations: Examples of process definition operations include:

- Analyze process requirements, including domain-specific analysis and application-specific analysis
- Instantiate, compose, decompose, tailor, and modularize process definitions
- Simulate, model, and validate process definitions

Internal: Discretionary access control might be used to provide need-to-know protection.

Relationships to other services: This service may interact with the Process Library and Process Exchange services. It may also interact with the System Traceability and Software Traceability services.

Examples: Existing notations and languages for representing processes include:

- Structured Analysis and Design Technique (SADT), data flow diagrams, state transition charts, Petri nets, Work-flow
- Marvel, APPL/A
- Action diagrams, HIPO, PMDB

4.3.2 Process Library Service

Conceptual: The Process Library service supports reuse capabilities for processes, analogous to software reuse. The process reuse concept is that life-cycle processes need not always be defined from scratch and that previous instances of process assets may be made available in libraries that may be national, organizational, or local in scope and that may be interconnected by networks. Process assets may range from complete life-cycle process definitions to individual process steps. Process assets may also be objects that can be versioned.

Operations: Examples of process library operations include:

- Create, update, and delete process assets
- Certify, measure, and manage process assets

Relationships to other services: This service may interact with the Process Definition and Process Exchange services. It may also interact with the System Traceability and Software Traceability services.

Examples: The Process Asset Library (PAL) currently under development at the SEI will provide a library of process assets and will involve infrastructure capabilities (e.g., guidelines, library access mechanisms). The PAL will be a national library available through software repository services, but it is also designed to be retrieved and instantiated for organization- or projects-specific purposes.

4.3.3 Process Exchange Service

Conceptual: The Process Exchange service supports interchange of process definitions between projects and between PSEs. It deals with transformations between different representations, integration of heterogeneous representations, and interchange formats for process assets. This service is principally intended for representations that are machine processable, that can be electronically transmitted between environments, and that embody formal syntax and semantics.

Operations: Examples of Process Exchange operations include:

- Encode and decode process metamodels
- Encode and decode process language syntax
- Transfer process definitions (output, export, import)
- Manage a “foreign language” interface for process exchange

Relationships to other services This service may operate in conjunction with the Process Definition or Process Library services and may interact with the Framework’s Network and Data Interchange services.

4.3.4 Process Usage Service

Conceptual: The Process Usage service supports the carrying out, enactment, or execution in a PSE of a project's defined and installed process. Installed project processes are typically carried out by a combination of manual human activities and PSE automated capabilities; hence, both humans and machines may serve as "enactment agents." The scope of this service includes capabilities for:

- users' selection, guidance, and control of process steps
- navigational and help facilities for users to query the installed process for information on succeeding actions
- varying the rigidity of enforcement of policies and constraints
- process metrics specification, collection, and reporting
- interactions of process definitions, simulations, and high-level representations with PSE data management

Operations: Examples of process usage operations include:

- Manage help and guidance facility for process users
- Query and report on process utilization and status
- Manage analysis and decision aids for users

Relationships to other services: This service may interact with the Process Definition service and the Process Library service. It may also interact with the System Traceability and Software Traceability services.

Examples: Emerging PSEs with process enactment support include ProSLCSE, EAST, and Cohesion.

Chapter 5

TECHNICAL MANAGEMENT SERVICES

The services in this chapter fall into a middle category that partakes of both Technical Engineering and Project Management. These services pertain to activities that are often equally shared by engineers and managers; the operations of these services do not clearly fall into one or the other category.

This chapter describes the following services:

- **Configuration Management service**
- **Change Management service**
- **Information Management service**
- **Reuse Management service**
- **Metrics service**

5.1 Configuration Management Service

Conceptual: The goal of configuration management is to identify, document, and control the functional and physical characteristics of configuration items to ensure traceability and reproducibility of a project's end products. This involves controlling, recording, and auditing the baseline and changes (pending or made) to the components of these end products. In the context of a PSE, an end product could be any of a wide range of items, including software, hardware, or a manufacturing process. A configuration item may be an aggregation of hardware, firmware, or software, or any of their discrete portions that satisfy an end user function. An item required for logistic support and designated for separate procurement may be a configuration item. A configuration item may also include a unique signature.

Examples of the end products or configuration items to be managed include:

- a computer program (e.g., operating system or application program)
- an integrated micro circuit device (e.g., ASIC, gate array, or hybrid)
- a board or circuit card assembly (e.g., graphics engine, microprocessor, signal processor, memory, or local area network)
- an equipment item that may be comprised of the above items (e.g., mini- or microcomputer, workstation, or network bridge)
- a system that is a collection all the above mentioned items that when interconnected support a specific domain (e.g., command and control system, missile guidance system, or jet propulsion system) or that is the evolving developmental support environment used to generate, test, and maintain the product.

This service provides the management required to maintain a product's many constituent pieces, including requirements statements, specifications, designs, drawings, CAD/CAM files, source code files, test documentation, logistic documentation, baseline definitions, and end-user documentation. This service supports identification and management of interrelationships between system components, any of which may themselves be composite objects. For various reasons a component with the same logical function may have several (sometimes alternative) implementations. This may be the result of fixes to errors, different operating requirements, and variety of interfacing requirements.

Configuration control measures may be applicable to configuration documentation for each configuration item, e.g., software identification numbers could be embedded in source code headers, and included with the engineering drawing of a product.

Operations: Examples of Configuration Management operations include:

- Create a new configuration
- Modify a configuration
- Recover an older configuration
- Delete a configuration

Internal: Each version of a configuration will have a unique identifier. The external names used for configurations may be different from internal identifiers used for consistency and tracking. Hence, external names may be changed, or multiple names may be defined for the same internally identified configuration.

For clarity and control, a configuration may be identified by identification number, release date, or by means of descriptive documents such as manuals.

In a secure PSE, all changes to items would be audited. Mandatory and discretionary access control services can be used to support need-to-know.

Relationships between configurations may be through a naming convention, held as relationships between nodes in a version graph, or in some other form.

Rules: There are dependency rules that govern the deletion of a configuration.

Relationships to other services: This service may interact strongly with the framework's Version service, as well as the Access Control and Concurrency Control services.

5.2 Change Management Service

Conceptual: The Change Management service supports the creation, evaluation, and tracking of change requests generated in response to errors, omissions, or required enhancements to a product. This service provides support for the resolution of a change request in terms of any decisions, task assignments, and product changes.

Change requests are evaluated for their criticality and benefit to provide an understanding of the potential impact if they are not addressed. An estimate of the resources required to carry out the change request may also be provided. Such decisions often involve complex human activities such as review boards or change control boards.

Based on the evaluation of a change request, one or more change orders may be created for it. Typically information recorded with a change order includes the date of the change order and the identifier of the change request initiating the order.

When a product is in use, new releases of the product will need to include changes made in response to particular change orders.

Operations: Examples of change management operations include:

- Create a change request in response to a reported error, omission, or required update
- Evaluate, classify, and retain historical record of a change request
- Create, evaluate, and track a change order for a change request
- Provide selection support for controlling elements from different configurations
- Generate a report or document; this may include:
 - change request status
 - change order status
 - audit trails of changes to a product component

Internal: A way to uniquely identify each change order is required. This allows the progress of the change order to be tracked. A status indicator allows new, in progress, and completed change orders to be distinguished. A priority level helps users to determine the importance (or otherwise) of non-completed change orders.

In a secure PSE, all changes to items would be audited. Mandatory and discretionary access control services can be used to support need-to-know.

Relationships to other services: This service may interact with the Configuration Management service.

Examples: Netherworld and ChangeVision provide examples of this service.

5.3 Information Management Service

Conceptual: The Information Management service provides the ability to manage, classify, and retrieve information within a PSE. Unlike the framework object management services which are concerned with the organization and storage of data objects and the relationships among them, the information management service provides for higher-level relationships among the information components within an information base. While the object management services are concerned with the internal data model (e.g., relational, object-oriented, hierarchical), information management provides for user-level manipulation of information needed to support tasks within the PSE.

There are many applications of this service found in most existing environments. Applications whose principal work is to perform some kind of database lookup or modification, e.g., a computerized telephone directory or namelist, are typical instances of this service. While the Reference Model considers these as potentially significant support services, it was considered impractical to enumerate each instance as a separate service.

Operations: Examples of information management operations include:

- Create, access, modify, and delete metamodels of information objects.
- Create, access, modify, and delete information objects.
- Create, access, modify and delete relationships among information objects.

Types: Information consists of objects, relationships among those objects and schema (meta-data) describing composite objects.

External: Access to information is often via user command language queries describing constraints on the information base. Commands generally contain information about what data is needed rather than how it is to be retrieved.

Internal: This service generally uses the facilities of the framework object management system to implement such services. The embedded data model is often hidden from the user.

In a secure PSE, all changes to items would be audited. Mandatory and discretionary access control services can be used to support need-to-know.

Examples:

- Remote information query services such as Archie, Gopher, WAIS
- Directory systems such as phone directories, parts lists, catalogs, etc.

5.4 Reuse Management Service

Conceptual: The Reuse Management service supports the storage, inspection, and reuse of assets related to many stages of engineering processes. These assets include such artifacts as requirements, designs, software components, test cases and documents.

There are three elements of Reuse Management: (1) storage, (2) indexing and classification, and (3) browsing and retrieval. The storage facility in which the assets are kept is commonly referred to as a repository or library. Indexing and classification are done through one of several competing strategies for reusable assets, such as “faceted” and “knowledge-based.” Browsing and retrieval are related to the actual mechanism by which the Reuse service is provided to users.

Although management of a reuse repository may occur in a local sense, a useful reuse repository will likely be a virtual construct, with reuse operations taking place throughout a distributed, heterogeneous network of actual repositories. Any reuse management operations that occur in such a larger context are necessarily bound by constraints external to an individual PSE.

Operations: Examples of reuse management operations include:

- Deposit, acquire, or submit asset into the repository
- Catalog, register, classify, accept, or index the asset
- Search or browse the repository
- Browse, examine, or extract the asset
- Register the user, extractor, or submitter
- Report problem with or use of an asset

Rules: The Repository administration must define the policies and rules with regard to all the operations, such as: who may extract an asset; the criteria for asset deposit; and the policies for charging and liability. The Reuse Management service must support and enforce these policies and rules.

The indexing strategy for browsing (e.g., faceted or knowledge-based) may greatly affect the management and user view of the assets.

Types: The primary unit of concern for the Reuse Management service is the asset. An asset is composed of elements linked together to form a reusable entity. The elements of an asset are themselves typed, such as Ada code, tagged document, or executable binary.

External: The external interface to the Reuse Management service is similar to a database system providing search and retrieval services. It is usually managed by an organization dedicated to providing reusable assets to a broad set of customers. The formats of the reusable assets are defined by their types.

Internal: The implementation of the Reuse Management service must provide reasonable performance and capacity. The relationship between the local repository and the external repository (connectivity and networks) must also be considered.

A common implementation model is the client-server model. In this model, the server can be located inside or external to the PSE. The client provides the user interface to the reuse operations.

In a secure PSE, all changes to items would be audited. Mandatory and discretionary access control services can be used to support need-to-know.

Relationships to other services: The Reuse Management service may interact with many of the Software Engineering services, in particular with the Software Design service.

Examples (storage facilities): The Asset Source for Software Engineering Technology (ASSET), Reusable Ada Packages for Information System Development (RAPID), and Central Archive for Reusable Defense Software (CARDS) are examples of storage facilities for reusable assets. InQuisiX and the Reuse Library Framework (RLF) are examples of classification and retrieval mechanisms.

5.5 Metrics Service

Conceptual: The ability to manage project development in a PSE depends upon the collection and understanding of quantitative data. Facilities are provided by the service both for primitive data collection as well as the organization of that data into information meaningful to end-users of the PSE.

Data is generally organized into three classes, of which the first and third class are typically the output of the System or Software Static Analysis services.

- Resource data provides information about product characteristics, such as number of components, size, and various static analysis measures.
- Performance data provides information about time-dependent processes, such as computer usage costs, error reports and personnel time charges.
- Complexity data provides information about the structure of the development project, both static analysis measures of the source documents and dynamic measures (e.g., of the executing program).

This data must then be transformed and used by various models. Various reliability models, often derived from hardware reliability theory, can be used to predict errors. Resource models depend upon various regression models, and various complexity models are based upon information theory, entropy or other finite state processes. These models are then used by other services, particularly the various Project Management services.

Operations: Examples of metrics operations include:

- Insert and delete data from data set
- Pick appropriate model for given data set
- Compare data set to predicted model
- Compute standard error and deviation in data set
- Graph data set
- Predict next point(s) in data set

Rules: The underlying model usually represents some formal mathematical property or equation.

Types: Data usually represents: product characteristics (e.g., size), process characteristics (e.g., errors), or structure (e.g., software science or cyclomatic complexity measures for a module).

Confidentiality of stored metric data may be required if the results are considered sensitive in nature.

Relationships to other services: The raw data of this service may be provided by the Metrication service and by the Software Static Analysis service. The models produced by this service may be used by the various Project Management services.

Examples:

- COCOMO cost estimation model
- Software science or cyclomatic complexity structure models

Chapter 6

PROJECT MANAGEMENT SERVICES

The services in this chapter support these activities related to planning and executing a project. Project planning is the activity by which efforts of all personnel responsible for a project are coordinated and integrated. Coordination and integration occur through a comprehensive plan for fulfilling the project's need in a timely manner and at a reasonable cost. Project planning takes place throughout the life of a project, from the project inception to completion. Typically, one of the first steps in a project involves assessing customer needs, examination of strategies to meet these needs, and discussion of the implications and effects of such strategies. A plan for producing a proposal may also be necessary.

A project may be carried out by a number of cooperating or subcontracting organizations. If this is the case, planning is necessary to manage the request for and selection of those organizations. Following project initiation (e.g., contract award) detailed planning of the project activities will be necessary, together with ongoing monitoring and re-planning of the project to ensure its continued progress.

This chapter describes the following services:

- **Planning service**
- **Estimating service**
- **Risk Analysis service**
- **Tracking service**

The activity of planning a project also requires definition of the process or processes that will support the project; services related to those activities will also contribute to project management.

6.1 Planning Service

Conceptual: The Planning service provides operations that permit handling of data according to a set of chronological constraints relevant to a project (i.e., describing the sequence of work and identifying significant task interdependencies). These include start and finish times for the project (as well as all component parts of the project). This service will support creation of structures such as a Work Breakdown Structure (WBS), the most common means for planning and scheduling a project.

Operations: Examples of planning operations include:

- Generate key project events
- Quantify inputs and outputs for work activities
- Compute event lead times (from manpower and cost models)
- Calculate start and finish dates
- Analyze critical path
- Generate part or all of the WBS in a database; this may include generation of detailed event schedules
- Register assignment of work responsibilities to individuals or organizations
- Modify work breakdown structure based on actual vs. scheduled progress

Rules:

- Project schedules may not have cycles or loops.
- Start and end dates for all work activities must be not be contradictory (e.g., no activity can be scheduled to complete before it begins).
- Individual work assignments can constrain the schedule.
- Some events may be unchangeable

Types: A work schedule may be created as a graph with nodes being individual activities and the edges being constraints that one activity has on another. Such graphs are commonly nested, with a node on a higher-level graph representing entire lower-level subgraphs. The data structure for this service may or may not use such a representation, but at least some equivalent for it will probably be necessary.

Internal: Creating and modifying the WBS may be implemented through a “Work Breakdown Structure editor” or an expert system. Such tools will have built-in knowledge of the normal components of a WBS, such as start/stop dates, no cycles, etc.

Relationships to other services: This service may interact with the Life-Cycle Process Engineering services. It may also interact with the Risk Analysis and Tracking services, as well as with the Configuration Management and Metrics services.

Examples: Examples of this service are TimeLine and MacProject.

6.2 Estimation Service

Conceptual: The Estimation service supports quantification, analysis, and prediction of project cost and resource needs. These include estimates for the size of a project, labor, equipment, facilities costs, and cost of computer resources allocated throughout the project lifecycle. Estimates may need to factor in the concept of multiple activities assigned to the same entity. This is more likely to concern a person being assigned two or more parallel jobs, although the same is possible for hardware allocations as well.

Operations: Examples of estimation operations include:

- Create cost, size, and resource estimates for product development, production, installation, operation, and support
- Generate estimates for variable parameters such as workload mixes or for differing design characteristics (e.g., safety, standardization, maintainability, survivability)
- Produce impact analyses based on alterations to variable parameters
- Perform sensitivity analysis on variable parameters
- Modify cost and size estimates and resource allocations

Internal: The principal component of an Estimation service could be an expert system. An implementation of this service may also choose that estimates from one area (e.g., size estimates) can constrain estimates in another area (e.g., cost).

Relationships to other services: This service may interact with the Numeric Processing service through the use of spreadsheets, etc. It may also interact with the Tracking service and with the Metrics services.

Examples: Tools that implement the Constructive Cost Model (COCOMO) estimation model provide examples of this service.

6.3 Risk Analysis Service

Conceptual: The Risk Analysis service supports those planning activities that consider elements related to the success or failure of a project. Tracking of items such as expected resource

usage (e.g., productivity, reliability errors) versus actual usage allows for predictions of total system needs. This service also includes calculation of various probabilities, such as those for budget overrun and technological failure, and such commonly used indices for success as the Mean Time Between Failures (MTBF) of a system or a system module.

Operations: Examples of risk analysis operations include:

- Perform trade-off analyses based on differing parameters for resource allocation and scheduling data
- Produce cost, schedule, and performance risk statistics and analyses
- Create decision trees, alternative and payoff matrices
- Calculate probabilities and generate reports for various allocation strategies

Relationship to other services: The Risk Analysis service may interact with the System Testing service, as well as with the Planning, Estimation, Numeric Processing, and Metrics services.

6.4 Tracking Service

Conceptual: This service supports correlation of estimated cost and schedule data with actual performance of a project; it also provides the capability to track action items to closure. It may provide triggers or alarms when actual data differ from planned resource usage, or when action items have not been closed after a certain period.

This service also supports the presentation of such data in human-readable form. Presentation will typically include transformation of project information into specific formats, such as Gantt or Pert charts. It could also include hardcopy generation of a WBS, interim briefing charts on project status, and the like.

Operations: Examples of tracking operations include:

- Gather metrics related to current status of a project and its constituent work activities
- Compare cost, size, and resource estimates with actual amounts
- Read and display status of all project variables (e.g., milestones met, cost, labor hours, etc.)
- Produce project data and summary information in various formats (e.g., according to the formatting rules of a given template, as a group of slides, based on a WBS, etc.)

Types: This service may need input from a personnel or an accounting database. As project personnel change through the life of a project, the cost for individuals will change, thus altering the overall cost estimates.

Relationships to other services: This service may interact with the Estimation and Planning services. It may also interact with the Software Requirements Engineering service for providing prototyping capabilities and with several of the Support services for manipulating and presenting project data.

Examples: SME (Software Management Environment) from NASA/GSFC allows for a knowledge-based approach to compare current resource expenditures to historical baselines. Presentation tools that produce Gantt and Pert charts are also examples of the operations of this service.

Chapter 7

SUPPORT SERVICES

Support services include services used by all users. They generally include those services associated with processing, formatting, and disseminating human-readable data; they also include services that provide support for use of the computer system itself.

This chapter describes the following services:

- **Common Support services**
 - Text Processing service
 - Numeric Processing service
 - Figure Processing service
 - Audio and Video Processing service
 - Calendar and Reminder service
 - Annotation service
- **Publishing service**
- **User Communication services**
 - Mail service
 - Bulletin Board service
 - Conferencing service
- **PSE Administration services**
 - Tool Installation and Customization service
 - PSE User and Role Management service
 - PSE Resource Management service
 - PSE Status Monitoring service
 - PSE Diagnostic service
 - PSE Interchange service
 - PSE User Access service
 - PSE Instruction service

7.1 Common Support Services

Interaction among PSE users is generally based on a set of standard representations. Of these the most important are textual information, numbers, and figures. In addition, a number of emerging technologies indicate that PSE users will also make growing use of digitized audio and video information. The Common Support services create representations relating to all of these information media that other services may use in providing their services.

Most of these services, as well as some other Support services, have a set of basic operations, such as create, modify, delete, move, copy, or save. The item in question for each service, however, is considerably different, and though there is an apparent duplication in the descriptions of these operations, there is a substantial difference in the nature of the operations themselves. The redundancy in the service descriptions below is, therefore, a necessary one.

7.1.1 Text Processing Service

Conceptual: The ability to create and manipulate textual information within the PSE is a service of primary importance. It is involved in supporting the activities of planning, design, documentation, engineering, and most management activities throughout the lifecycle of any project.

Text may be viewed either as characters on a two dimensional plane, with operations available to navigate on the plane, inserting and modifying text as needed, or as structured objects (e.g., graph, tree, table, or formula) with operations available to also navigate around this structure making changes as needed.

Operations: Examples of text processing operations include:

- Create, modify, delete, and save text for future use
- Import externally produced text into a format usable by this service
- Export text in various formats
- Format or print text or documents
- A collection of text manipulation primitives including
 - move, copy, or input text
 - include or merge previously saved text with current text
 - search for, replace, or compare text strings
- Format document, create template, print
- Check spelling and grammar, lookup (i.e., in dictionary or thesaurus)
- Create and manipulate textual table

- Scan externally produced text into a format that can be manipulated by this service.

Rules: Depending upon the services implemented in the PSE, the Text Processing service may need to implement text format or template design format rules, spelling or grammar rules, thesaurus substitution rules, tool or file format rules, and textual table design or format rules.

Types: The Text Processing service requires an underlying character set for representing text as well as any required special symbols. Schema may be defined giving templates for words, paragraphs, chapters, tables, equations and other design template aspects.

External: This service may be used by other services within the PSE (e.g., electronic mail) or may be invoked directly by the user of the PSE to create textual objects.

Internal: Objects referenced by the Text Processing service are often files in the file system or objects in the object management system. More complex structures can also be used, such as graphs or tree structures linking parts of a document, such as in a syntax-directed editor.

Relationships to other services: Text Processing services may interact with any PSE service requiring the use and manipulation of textual information for user input, display, output or tool or file textual format conversion.

Examples:

- General editors such as vi or emacs
- Specialized editors, such as context sensitive editors
- Syntax-based editors (DEC's LSE, research editors like Mentor, SUPPORT, or CPS)
- Text manipulation tools such as awk and grep.

7.1.2 Numeric Processing Service

Conceptual: The ability to create and manipulate numeric information (e.g., spreadsheets, libraries of standard mathematical functions, statistical packages) within the PSE is one of the major services involved in supporting the activities of planning, budgeting, and management of projects.

Operations: Examples of numeric processing operations include:

- Create, modify, edit, delete, and save formulas and spreadsheets
- Import or export spreadsheet data in various tool or file formats
- Format or print numbers, formulas, or calculated results
- A collection of number and formula manipulation primitives including

- move, copy, or input numbers or formulas
- include or merge previously saved spreadsheets with current values
- search for, replace, or compare number or formula strings
- Create, manipulate, calculate, and print mathematical formats and templates
- Calculate general arithmetical operations such as square root, logarithm, sine, cosine, etc.
- Create and manipulate numeric tables

Rules: Depending upon the services implemented in the PSE, the Numeric Processing service would need to implement mathematical format or template design format rules, including mathematical function input parameter or calculation rules, spreadsheet tool or file format rules, and numeric table design or format rules.

Types: The Numeric Processing service would need to define valid number formats, that is, integer, fixed point, floating point or scientific and limits, implemented mathematical functions and their parameters, the basic formula format or symbol definitions and calculation rules, and the specific spreadsheet or non-spreadsheet token model implemented.

Internal: The Numeric Processing service would normally be implemented by means of runtime libraries of mathematical functions, as well as by separate PSE tools, such as a spreadsheet or calculator applications, designed to be enacted whenever numeric processing services are needed.

Relationships to other services: Numeric Processing services may interact with any PSE service requiring the use and manipulation of numeric information for user input, display, output, tool or file format conversion.

Examples: Spreadsheets such as Lotus 1-2-3, Microsoft Excel, Borland's Quattro pop-up calculators, and separate tools for creating equations or tables (e.g., EQN and TBL) are examples of this service.

7.1.3 Figure Processing Service

Conceptual: The Figure Processing Service deals with the creation and manipulation of graphic, image, or documentation figures within the PSE. It involves supporting images for any other end-user service and activity.

Operations: Examples of figure processing operations include:

- Graphic or image creation, modification or editing, deletion, and saving for future use
- Graphic or image manipulation primitives such as zoom, size, shrink, rotate, fill, align, move to foreground or background, compose or decompose, include or merge previously saved graphic or image objects with current graphics or images; search for, replace or compare graphic or image objects
- Import or export graphics or images in various formats
- Format or print graphics, images or documents
- Scanning externally produced pictures into a format that can be manipulated by this service.

Rules: Depending upon the services implemented in the PSE, the Figure Processing service would need to implement graphic or image or figure format and template design format rules, tool or file format rules, and graphic, image or figure table design or format rules.

Types: The Figure Processing service may need to define valid graphic, image or figure object primitives and special graphic object representations, as well as the basic graphic, image or figure token model implemented.

External: The PSE would automatically invoke Figure Processing service functions whenever needed by other services; these services may also be invoked directly by the PSE user.

Internal: The Figure Processing service would normally be implemented as separate PSE tools, that is, graphic, image or figure editor applications, designed to be invoked by the framework whenever figure processing services are needed by other services, tasks, functions or tools active in the PSE.

Graphic data is often stored either as a bit-mapped (or raster) data object representing the pixels that will be the picture, or as a set of rules for drawing the picture (e.g., as a set of vectors).

Relationships to other services: The Figure Processing service may interact with any PSE service requiring the use and manipulation of graphic, image or figure information for user input, display, output, tool, file format conversion or directly by the PSE user.

Examples: Examples of the Figure Processing service include independent drawing tools such as MacDraw, MacPaint, xfig under X Windows, and pic for Unix troff.

7.1.4 Audio and Video Processing Service

Conceptual: The ability to capture, create and manipulate data that is based on audio- or video-based sources will soon become a necessary capability of PSEs. There are numerous technologies now emerging, including enhancement of graphical devices, animation, “ink” (computer

manipulation of hand-written text), and similar means of computer processing of digitized audio and video data. Although these capabilities are all currently in relative infancy, many of them will soon be common. It may be the case that some of these capabilities will eventually be regarded as proper either to the Framework or even platform or hardware services.¹⁰

The integration of these services with currently existing services will also be likely. For instance, voice tagging of an ASCII text file, or freezing a video image containing numeric data and then capturing that data for inclusion in a spreadsheet, are likely possibilities for integrating multimedia operations with “traditional” computer tools.

Operations: Examples of audio and video processing operations include:

- Create, modify, and delete sound and video data objects
- Record, playback, and transmit audio and video data objects
- Transform data from one form (e.g., audio) into another form (e.g., ASCII text).
- Store audio and video data in other formats (e.g., PostScript)

Relationships to other services: This service may have a close relationship with the Framework’s Dialog service. This service may also interact with the Text, Numeric, and Figure Processing services, the Publishing service, the Mail service, and any other services in the environment that deal with human-readable information.

7.1.5 Calendar and Reminder Service

Conceptual: This service provides the means for a user to keep an electronic schedule of meetings, deadlines, and similar important dates and times. This service may be a passive one, i.e., simply an electronic form of a traditional desk schedule. It may also be a more active service, such as sending automatic reminders of dates and times for deliverables to project members, initiating actions on objects, or automatically triggering process steps.

Operations: Examples of operations of this service include:

- Insert and delete items in an electronic calendar
- Select actions for execution at a given date and time
- Display and print calendar information in different modes (e.g., entire year, particular months, weeks, and days)

¹⁰There has been a proposal to modify the NIST/ECMA Framework’s Presentation service to accommodate some of these emerging technologies.

- Select actions for execution when specified event occur (e.g., alarms)

Rules: Entries are associated with a date and time

External: The external interface to the Calendar and Reminder service is based on the display of a calendar. This calendar can be displayed at different levels of granularity (day, week, year).

Relationship to other services: This service may interact with the other Common Support services, with the Mail service, and with several of the Project Management services.

Examples: *Synchronize* is an example of this service.

7.1.6 Annotation Service

Conceptual: The Annotation service provides for associating comments with existing objects. The comments may be in the form of text, diagrams, audio, or video.

Operations: Examples of operations of this service include:

- Add, delete, or modify commentary associated with an object
- Copy or display object with or without commentary
- Order commentary by date, time, size, or author
- Connect commentary to multiple objects

Rules: Attaching a comment to an object cannot adversely affect existing uses of that object.

Modification of a comment may be limited to the author or administrator.

Reading of comments or certain aspects of the comments may be restricted to specific individuals.

Types: The comment may be text, diagram, audio, video, or a mixture of these.

Internal: A common implementation model is the use of a central comment database (e.g., RDBMS, OODBMS). All comments are gathered and managed in that database, although the objects to which the comments apply may be external to it.

Relationships to other services: The Annotation service may interact with the Text, Numeric, Figure, and Audio and Video Processing services.

7.2 Publishing Service

Conceptual: The basic function of the Publishing Service is to create and print documents. Until recent times, documents have been understood to be collections of printed pages. But

with the advent of digital technology, this definition is no longer satisfactory. A more abstract notion of what “documents” are sees them as structured, portable collections of related objects that intend to communicate information. This includes books, letters, brochures, hypermedia collections, and on-line services. It includes both documents that are read (e.g., papers, etc., consisting of pages and divided into multipage segments called chapters or sections) and documents that are viewed (e.g., slides, etc. where one page is independent of another page, documents with video content, etc.).

Given this broad definition, the function of the Publishing service must be equally broad; it is therefore:

- to create, edit, and delete the objects that compose documents;
- to structure these objects in relation to each other;
- to deliver the resulting document in a portable form.

Publishing involves three aspects of a document:

1. *Content* The actual content of the objects that compose documents can be created, edited, and deleted by the Publishing service. The objects that the Publishing service manipulates includes textual objects, vector graphics, bit-mapped images, data-driven charts, and equations. Thus, the Publishing service requires other services for editing each of these types of object (i.e., text editors, graphics editors, etc.). In addition, document objects can be temporal in nature, such as audio or video objects, or animations. Documents may also include finding aids, such as tables of contents, hyperlinks, or full-text query facilities.
2. *Format* Each object in a document has not only *content* but *appearance* as well, which we refer to as “format.” Format for text objects includes attributes such as font, type, size, face, and leading. format for bit-mapped objects includes color/gray-scale, size, and resolution. Format for audio objects includes amplitude and dynamic range.
3. *Structure* The objects in a document are related to each other in well-defined ways. The rule base (implicit or explicit) that describes document structure defines the type of document in question. For example, if the document is a business letter, it has exactly one date followed by exactly one address, etc. If the document is a technical manual, it has a different structure, one that includes rules such as “When an illustration object is included, it must be followed by exactly one descriptive caption.” The Publishing service requires tools for creating the rule base for each type of document that it publishes.

Operations: Examples of publishing operations include:¹¹

¹¹These are often the same operations available through the Common Support services such as Text Processing, Figure Processing, Audio and Video Processing, etc.)

- Create, modify, delete, and save text for future use
- Import or export text in various formats
- Create tables of contents, indices, bibliographies, and glossaries
- Format or print text or documents
- A collection of text manipulation primitives including:
 - move, copy or input text
 - include or merge previously saved text with current text
 - search for, replace or compare text strings
- Import figure, graphic or table into document
- Create or modify layout or style for document
- Format document, create template, print
- Check spelling and grammar, lookup (i.e., in dictionary or thesaurus)
- Create and manipulate textual table
- Build document
- Preview document
- Print template

Rules: In producing a document, many visual design decisions must be considered. Issues such as font selection, format and structure rules, and page layout are very important to producing quality documents. These decisions are supported by the Publishing Service.

Types: The page is the basic element of the Publishing Service. Pages contain items with specific semantics, such as words, paragraphs, titles, and figures.

Documents are defined as a collection of pages. Documents have parts with specific semantics, such as title page, abstract, preface, table of contents, section, bibliography, and appendix. There are types of documents, such as memoranda, projection charts, technical reports, presentation slides and letters. Documents may also contain data whose sources include multimedia data objects, such as a frozen image from an animated video data file.

Often style guides (i.e., schema) are used to represent the semantics of the parts of a document.

External: The contents of documents are constructed by an author entering it via the keyboard or by importing it from another service. The output of the Text Processing Service may serve as input to the Publishing Service.

The Publishing Service produces data directly to a printing device or into a file for subsequent processing. The Publishing Service may produce a variety of output formats depending on the target printer.

Documents may be electronic and should be able to be viewed in that manner. Documents also may have sound and video content.

Internal: Because of the complexity of the Publishing Service, significant attention is placed on internal performance and construction of the visual presentation.

There are generally two models for building Publishing Service products. In one case, formatting information is inserted directly into the document (e.g., troff, WordPerfect). In the other case, editing commands are external to the document (e.g., TeX style guides) and can be altered independently to altering the document itself.

Relationships to other services: The Publishing Service may interact with the Text, Numeric, and Figure Processing services.

Page items and document parts (e.g., paragraphs, figures, abstracts, appendices) may be objects visible to the framework Object Management services.

Examples: Interleaf5 from Interleaf, Framemaker from Frame Technology, Pagemaker from Aldus, Tex, Latex, and troff are all examples of this service. Freelance, the Slitex variant of Tex, and Powerpoint are examples of Publishing service tools that emphasize slide presentation. Postscript is an example notation for describing text and graphics within such documents.

7.3 User Communication Services

Interaction between individuals is accomplished by many mechanisms. When such interactions are pertinent to the activities of a project and are supported by services of the PSE, they are handled by the User Communication services.

7.3.1 Mail Service

Conceptual: The Mail Service provides for simple communications of notes between computer system users. It follows a paper letter mail paradigm, while taking advantage of the speed and connectivity of wide area computer networks. Automatic note forwarding and collecting are also part of the Mail service.

Operations: The Mail service has the common set of operations needed to create and save textual information. In addition, the Mail service has the following unique operations:

- Receive, compose, send, reply, forward, broadcast, and acknowledge mail

- Electronically review mail
- Customize the mail's send and receive capabilities

Types: Mail is often stored in objects called folders. Folders are usually related to specific subjects or addressees.

Mail can be sent to individuals, or mailing lists of many individuals can be created to broadcast messages to many users at one time.

The basic object of the Mail service is the message; a message is the item that is mailed. Messages have subfields, such as addressee, author, date, subject, salutation, body, and closing.

External: The external view of the Mail service is via simple, easy to use tool kits (or utilities).

The Mail service is also the primary connection for most users who are communicating across LAN and WAN networks.

Internal: The major distinguishing feature of the Mail service among the User Communication services is that it represents one way asynchronous communication between two users. Mail is sent by one user and the PSE stores the mail until retrieved by the receiver. An acknowledgment is sometimes sent to the sender stating that the message is waiting for the receiver, but whether the receiver actually read the message is generally not known.

In a secure distributed PSE, authentication of the sender by the receiver is required. There are also other security considerations, such as Secure Mail and Privacy Enhanced Mail, that are still research topics.

Relationships to other services: The Mail service may use the Text Processing service to create messages. It may also interact with the Figure Processing and Numeric Processing services for creating complex messages. Other services in the PSE may use the Mail service to communicate with other users, e.g., informing a test team that a program build is completed and ready for testing.

Examples: Unix mail, MailTool, and Andrew Tool Kit are examples of this service.

7.3.2 Bulletin Board Service

Conceptual: The Bulletin Board service is a form of mail with a single sender and multiple recipients. Information is mailed to the bulletin board and users access the bulletin board as desired. Unlike mail, which prompts receivers to read the message and may acknowledge receipt to the sender, a Bulletin Board service simply posts the information, and it is up to each intended receiver to interrogate the Bulletin Board for new information.

Operations: The Bulletin Board service has the common set of operations needed to create and save textual information. In addition, the Bulletin Board service has the following unique operations:

- Subscribe and post messages
- Browse a bulletin board's messages
- Reply or add information to posted messages
- Forward messages

Types: Often users can “subscribe” to certain bulletin boards, and therefore lists of such users must be maintained. There are often multiple bulletin boards and the names of such bulletin boards must be maintained also.

External: Access to the Bulletin Board service is often via an interface that looks very much like a Mail service interface.

Internal: There are generally two methods for implementing a Bulletin Board service. One way is via the Mail service. Lists of users subscribing to a given bulletin board are maintained, and a post operation is simply a Mail service broadcast operation to all users who have subscribed to this service.

A second method is for the Bulletin Board service to post items into an object in the object management system, and the browse operation reads that object.

Like the Mail service, this is also an asynchronous communication service.

Relationships to other services: The Bulletin Board service often uses the Mail service to broadcast messages and the Text Processing service for creating messages. The framework Communication service is used to send messages across a LAN or WAN.

Examples: Readnews, VAXnotes, and Unix notefiles are examples of this service.

7.3.3 Conferencing Service

Conceptual: Often it is necessary for users to engage in interactive synchronous communication. This is handled in the PSE by the Conferencing service. While the model implemented often resembles a two-way telephone call, the service could also be implemented to permit computer-based conferencing for many users.

Operations: Examples of conferencing operations include:

- Open connection
- Send message
- Close connection

Types: Objects that can be sent and received through this service may include ASCII text, sound, video, and graphical objects.

External: This is often a separate service that creates multiple viewing areas on the screen, each representing one end of the conversation.

Relationship to other services: This service may use the framework's Communication service to permit synchronous communication to proceed.

Examples: The Unix talk program is an example of this service.

7.4 PSE Administration Services

The administration of a computer system includes services that span all of the levels of the system. They include services whose province is low-level operating system support, as well as services that affect the engineering and management users of the environment.

While these services provide administration capabilities that pertain to an overall environment, they also have considerable overlap with the functionality of the Framework Administration services originally described in the ECMA/NIST reference model. Most of those services have now been subsumed by the service below.

7.4.1 Tool Installation and Customization Service

Conceptual: This service supports the installation, testing, and registration of tools into a PSE. The service provides the necessary operations to set up default resource limits, default names, and other operational characteristics for a tool to be used in a PSE. The installation of a tool may involve significant changes to the tool's implementation or encapsulation of the tool in some form of tool wrapper. Installation of a tool may also involve specific vendor constraints, such as the operation of tool-specific daemons to ensure that the number of users agrees with the number of licenses purchased.

Tools are designed to operate with a specific user interface. However, it is desirable to have consistency of user interface style and operation across a number of tools. Hence, in addition to the work required to install a tool, customization of a tool's user interface may be required.

Being able to define a test environment for a recently installed tool protects the PSE from errant behavior on the part of the new tool. For example, names and defaults are kept local to the tester and may not affect other versions of the tool present in the PSE.

It may be possible to access alternate versions of the same tool within a single PSE.

Operations: Examples of tool installation operations include:

- Install tool by setting default resource limits (e.g., CPU time, objects manipulated, file

- sizes, and default names) for that tool, or according to a vendor's licensing regulations
- Customize tool, e.g., by providing a new user interface
 - Create a test environment for a recently installed tool
 - Register a new tool in a PSE by making it known to potential tool users
 - Provide access to multiple versions of the same tool
 - Unregister an existing tool in a PSE, making it unavailable for use

Internal: Installing a tool in a PSE may involve a major amount of work on the part of the installer. Amongst other things, the amount of work required is dependent on the compatibility of the internal architecture of the tool and the PSE. In addition, porting an existing tool to a PSE may require encapsulation of the tool in some form of tool wrapper.

Registering a tool makes that tool known to potential users. This may take a number of forms, including making the tool's name known in a public directory, placing the tool itself in a well-known place in the PSE, extending the project schema of a database-centered PSE, and so on.

Relationships to other services: In installing and testing a tool it may be necessary to make use of the User and Role Management service to partition the PSE and Policy Enforcement services to protect other users of the PSE. User and Role Management and Policy Enforcement services are needed to ensure that only authorized individuals can access these Tool Registration Services and the registered tools themselves.

7.4.2 PSE User and Role Management Service

Conceptual: Users of a PSE must be made known to the system through some form of registration. Users may be grouped according to particular user roles within the PSE (e.g., developer, designer, manager). Each role may require particular tools, permissions, data, and so on. These provide (potentially) overlapping partitions of the PSE.

The utility of this service may be to facilitate security, to aid partitioning of the PSE data for distribution purposes, to allow easier communication between groups of PSE users, to allow tasks to be assigned to groups, and so on.

Operations: Examples of role management operations include:

- Register a new PSE user
- Deregister an existing PSE user
- Create a new role of PSE user
- Delete an existing role of PSE user

- Assign a user as a member of a PSE role
- Deassign a user from being a member of a PSE role
- Select a view or partition of the PSE tools, data, etc., as visible (i.e., accessible) to a particular role
- Amend the view defined for a role

Rules: A user can be assigned to more than one role.

Internal: Many of these operations could be provided by (or complemented by) framework mechanisms for the administration of mandatory or discretionary access control.

Relationships to other services: This service may interact with the Tool Installation and Customization service and with the Lifecycle Process Engineering services.

Examples: Most available PSEs have user registration facilities. Grouping services are available to Unix groups and as roles in ISTAR, ASPECT and other PSEs.

7.4.3 PSE Resource Management Service

Conceptual: The Resource Management service provides the ability to monitor, add, change, or delete resources available to a PSE. The resources supported include disks, memory, tapes, quotas, and workstations.

Operations: Examples of resource management operations include:

- Add a resource to the PSE
- Delete a resource from the PSE
- Amend a resource of the PSE
- Query status of a resource in the PSE
- Provide statistics on a resource of the PSE
- Make a resource available to user and/or role
- Interrogate PSE for availability of a resource

Types: The following are types of resources: disks, memory, tapes, user quotas, workstations, tape drives, terminals, mainframes.

External: Other services will use the resource management service to determine availability of resources for an operation.

Internal: This service is concerned only with the user, role, and tools levels; it does not address the system level platform resource management.

Relationships to other services: This service may interact with the User and Role Management and the PSE User Access services.

7.4.4 PSE Status Monitoring Service

Conceptual: During execution of the PSE it is necessary to monitor and control the actions that take place. The information obtained can be used to adjust, or tune, the PSE to improve its availability and performance. Particular information of use to the PSE administrator may include statistics on the uptime of the PSE, overall tool usage, resource usage by individuals, average response time, and so on. This service is analogous to the monitoring functions performed by a database administrator or a system administrator.

Operations: Examples of status management operations include:

- Log actions and events that occur during normal execution of the PSE
- Produce report on PSE usage

Internal: This service may be heavily dependent on the underlying monitoring services of the operating system on which the PSE is implemented.

Relationship to other services: This service may interact with the Metrics service.

7.4.5 PSE Diagnostic Service

Conceptual: A PSE must be able to perform self-testing and diagnosis to determine irregular conditions. The PSE may then be able to correct problems automatically or to send messages to a PSE administrator for human intervention. The irregular conditions may include inability to achieve expected network connections, lack or fragmentation of disk or secondary storage facilities, or inconsistent relationships in the environment (e.g., missing tools).

Operations: Examples of diagnostic operations include:

- Initiate self-test of PSE facilities
- Collapse fragmented storage
- Send diagnostic message to system administrator
- Perform automatic rollback from an invalid environmental condition

Relationships to other services: The service may interact with the PSE Status Monitoring service, the PSE User Access service, and the Framework's Archive service.

7.4.6 PSE Interchange Service

Conceptual: Communication and sharing between PSEs requires services for interchange between PSEs. For example, tools and data from one PSE may be transferred to another PSE to facilitate reuse. This requires external protocols from the PSE to the outside world.

Also, initial loading of data and tools into a PSE must be handled by this service.

Operations: Examples of interchange operations include:

- Transfer data between PSEs
- Transfer a tool between PSEs
- Transfer user/role between PSEs
- Transfer task description between PSEs

Rules: Consistency of the receiving PSE must be maintained following the interchange.

Internal: Some form of encryption may occur between the PSE and the outside world in order to ensure greater security.

Relationships to other services: This service may interact with the framework's Data Interchange service. Also, the PSE user access service may constrain access to transfer operations to trusted users.

7.4.7 PSE User Access Service

Conceptual: The PSE needs to know who is accessing resources and to provide control over access to the PSE.

Operations: Examples of user access operations include:

- Login a user into the PSE
- Authenticate a user of the PSE
- List all current users of the PSE
- Logout a user from the PSE

- Set privileges of a PSE user
- Set access to resources by another PSE user

Rules: All access to a PSE must be preceded by a login operation.

Users often have a predefined role associated with a login operation, constraining the PSE resources available to them.

External: Authentication of a user by a login or authentication operation is typically by a password, but can include other characteristics such as personal facts, fingerprints, ID cards, handwriting, etc.

Relationships to other services: This service may interact in many ways with access control at the framework level.

Example: Unix root privileges provide an example of the set-privileges operation, and the Unix `chmod` function for altering file access attributes is an example of the set-access operation. The ASPECT PSE uses `publish` and `acquire` operations to allow other roles to have access to private data.

7.4.8 PSE Instruction Service

Conceptual: The appropriate use of tools often requires online information to be given to the user of the tool. This information may either be in the form of an interactive tutorial run by the user in order to learn the features of a new tool or may be invoked within the tool as part of a “help” facility.

Often the Instruction service will enact the actual tool and guide the user through the use of that tool with specified input. In other cases, the Instruction service may simply display a predefined file of information on the desired topic (and thus behave much like the Information Management service).

Operations: Example operations of this service include the following:

- Display information
- Browse (search for) information
- Explain (interpret) program output, command input, or data item
- Simulate (with input and output) tool activity

Internal: This service is often implemented as a separate tool from the tool it is explaining. However, at other times it may be implemented as a “hot key” within the tool itself. Enacting this key causes the tool to enter a “help” mode where it displays useful information to the user.

Relationship to other services: This service may interact with the Information Management service as well as the framework User Assistance service in intercepting and explaining error and warning messages. This service may also interact with any tool supporting any service of this reference model, as described in the Internal dimension, above.

Examples: Examples of online instruction include the following classes of tools: Computer Aided Instruction (CAI) and online tutorials; and Browsers and online help facilities (e.g., UNIX man utility).

Chapter 8

FRAMEWORK SERVICES

These services comprise the infrastructure of a PSE. They include those services that jointly provide support for applications, CASE tools, etc. and that are commonly referred to as “the environment framework.” The source for these service descriptions is the “Reference Model for Frameworks of Software Engineering Environments, Edition 3,” NIST Special Publication Number 500-211, August, 1993. Although most of the following text has been extracted directly from this source, the text has been substantially abbreviated. For a full description of each service, the reader should consult the NIST/ECMA Frameworks Reference Model itself.

In the case of one section of the NIST/ECMA reference model, (section 10, Framework Administration services), it was felt that most of the services were essentially subsumed by other services already described in the PSE reference model. These included the following:

- Registration service (section 10.1): for this service, see section 7.4.1, Tool Installation and Customization Service.
- Resource Management service (section 10.2): for this service, see section 7.4.3, PSE Resource Registration service.
- Metrication service (section 10.3): for this service, see section 5.5, Metrics service.
- Sub-Environment service (section 10.4): for this service, see section 7.4.2, PSE User and Role Management service.
- Self-Configuration Management service (section 10.5): for this service, see section 7.4.1. Tool Installation and Customization service.

One remaining service from this section of the NIST ECMA reference model (License Manager service, section 10.6) is not duplicated in the PSE reference model. This service provides for the enforcement of licensing requirements on PSE components. As with all of the other abbreviated framework service descriptions, the reader should consult the NIST/ECMA reference model for its full description.

This chapter describes the following services:

- **Object Management services**
- **Process Management services**
- **Communication services**
- **Operating System services**
- **User Interface services**
- **Policy Enforcement services**

8.1 Object Management Services

NB: These service descriptions have been abstracted from the NIST/ECMA Reference Model, section 4.

Conceptual: The general purpose of the object management services is the definition, storage, maintenance, management, and access of object entities and the relationships among them. These services are generally built upon the database and file system services of the platform. These services include the following:

8.1.1 Metadata service (NIST/ECMA, section 4.1): provides definition, control, and maintenance of metadata, typically according to a supported data model. Metadata (e.g., schemas) is data about the structure and constraints of data and objects in the object manager. A Metadata Service allows generic tools to be written which operate according to the structure of the objects in a particular environment.

8.1.2 Data Storage and Persistence service (section 4.2): provides definition, control, and maintenance of objects, typically according to previously defined schemas and type definitions. It is this service which permits an object to live beyond the lifetime of the process that created it and allows access to it by that process or by other processes until it is deleted. This service provides the essential storage characteristic of a “database” or object management system.

8.1.3 Relationship service (section 4.3): provides the capability for defining and maintaining relationships between objects in the object management system. These relationships provide the links to move between objects in the object management system. For models, like the E-R data model, they provide the essential links for building such models. For others, like object-oriented models, they provide a mechanism for building inheritance and like properties.

8.1.4 Name service (section 4.4): supports naming objects and associated data and maintains relationships between surrogates and names. This service provides for the translation of external names known to users and tools within an environment to internal, often arbitrary, identifiers of those objects. The external names can be file names, function names, process names, etc.,

while the internal names often represent physical locations in the object management system, arbitrary counters or hash-coded table-lookup values.

8.1.5 Distribution and Location service (section 4.5): provides capabilities that support management and access of distributed objects. The Location Service may have both a physical and logical model of the object management system. Distributed software development support is firmly established as a requirement for PSE frameworks, and this service permits users and tools to locate necessary objects in the environment.

8.1.6 Data Transaction service (section 4.6): provides capabilities to define and enact transactions. Transactions are units of work made up from a sequence of atomic operations. Such operations must not terminate in a half-completed state. Implementation of such operations is handled by this service via operations like commit and rollback.

8.1.7 Concurrency service (section 4.7): provides capabilities that ensure reliable concurrent access (by users or processes) to the object management system. In multiprocessing systems and distributed networks it is imperative that certain critical operations execute to completion before their data can be interrogated by another process. This service provides these capabilities via operations like acquire and release locks.

8.1.8 Operating System Process Support service (section 4.8): provides the ability to define OS processes (active objects) and access them using the same mechanisms used for objects. This provides integration of OS process and object management. This service provides the basic support mechanisms for enacting and controlling active objects in addition to the more static data in the object management system.

8.1.9 Archive service (section 4.9): allows on-line information to be transferred to off-line media and vice-versa. This service allows users to determine which objects are readily available via the object management system and which require increased access time by retrieval from off-line media such as tape. Size of the object management system and size of the individual objects determines whether and how often the Archive service needs to be used.

8.1.10 Backup service (section 4.10): restores the development environment to a consistent state after any media failure. While the Archive service is often viewed as one used by users of an environment to manage the objects under their control, the Backup service is often viewed as an administrative function that provides reliability and integrity to the data in the object management system and is generally transparent to the users of an environment.

8.1.11 Derivation service (section 4.11): supports definition and enactment of derivation rules among objects, relationships or values (e.g., computed attributes, derived objects). Many objects in the object management system are often related (e.g., type definitions, source code, object code, executable modules), and changes to one often affect the others. This service provides the capabilities to link these objects in such a way that rules are established for deriving the related objects from other objects.

8.1.12 Replication and Synchronization service (section 4.12): provides for the explicit replication of objects in a distributed environment and the management of the consistency of redundant copies. The basic operations of this service are to provide synchronization of multiple objects

and manage replicated objects so that ownership is not hindered.

8.1.13 Access Control and Security service (section 4.13): provides for the definition and enforcement of rules by which access to PSE objects (e.g., data, tools) can be granted to or withheld from user and tools. This service utilizes the operations within the Policy Enforcement services of the framework.

8.1.14 Function Attachment service (section 4.14): provides for the attachment or relation of functions or operations to object types, as well as the attachments and relation of operations to individual instances of objects. This provides the basic functionality to implement inheritance properties in the object-oriented data model.

8.1.15 Common Schema service (section 4.15): provides mechanisms for integrating tools into a PSE by providing a means to create common (logical) definitions of the objects (and operations) these tools may share from the underlying objects in the object management system. This service provides the capabilities for creating and modifying such schema in order to integrate new tools into an existing set of tools and their related data in an environment.

8.1.16 Version service (section 4.16): provides capabilities for managing data from earlier states of objects in the OMS. Change throughout development has to be managed in a PSE and the inclusion of versioning is one of the means of achieving this. This service provides the capabilities to create new versions of objects and to recover earlier versions of objects.

8.1.17 Composite Object service (section 4.17): creates, manages, accesses, and deletes composite objects, i.e., objects composed of other objects. It may be an intrinsic part of the data model or a separate service. Complex objects in a PSE (e.g., source code, a report) may consist of collections of other objects linked in specific ways. This service allows for such objects to be considered as either a single composite object (e.g., a report) or as subsets of this object (e.g., a chapter, a table).

8.1.18 Query service (section 4.18): an extension to the Data Storage and Persistence service's read operation. It provides capabilities to retrieve sets of objects according to defined properties and values. These capabilities can be fairly simple navigation operations (e.g., "Get all objects linked to X") or more complex inference rules (e.g., "Get all objects linked to X and Y but not to Z").

8.1.19 State Monitoring and Triggering service (section 4.19): enables the specification and enactment of database states, state transformations, and actions to be taken should these states occur or persist. This service provides an asynchronous event mechanism among independent tools and provides the capabilities for the object management system to become an inter-tool signalling channel.

8.1.20 Data Subsetting service (section 4.20): enables the definition, access and manipulation of a subset of the object management model (e.g., types, relationship types, operations if any) or related instances. The PSE framework may provide support for selecting parts of the OMS necessary to carry out specific subprocesses of the overall project.

8.1.21 Data Interchange service (section 4.21): offers two-way translation between data repos-

itories in the same or different PSEs. This permits the object management system to transfer data to other PSEs.

8.2 Process Management Services

NB : These service descriptions have been abstracted from the NIST/ECMA Reference Model, section 5.

Conceptual: The general purposes of the Process Management services in a PSE are the unambiguous definition and the computer-assisted management of project development activities across total project lifecycles. In addition to technical development activities, these potentially include management, documentation, evaluation, assessment, policy-enforcement, business control, maintenance, and other activities. The services are:

8.2.1 Process Development service (section 5.1): provides for creation, control, and maintenance of process definitions, sometimes represented in a process formalism. These definitions correspond to process assets needed by an organization; these assets may be definitions of a complete process, a subprocess (or process element), a process model, a process architecture, or a process design.

8.2.2 Process Enactment service (section 5.2): provides for the instantiation and execution of process definitions by process agents that may be humans or machines. It also provides services to access, maintain, and control the persistent state of the process.

8.2.3 Process Visibility service (section 5.3): provides facilities for the definition and maintenance of visibility and scoping information associated with enacted processes. Several enacting processes may cooperate to achieve the goal of a higher level process or a complete process; the extent of such cooperation is part of the definition of processes and may be provided by integrated visibility features in a particular process definition representation; however, independent services may be provided to deal with interprocess interactions such as visibility of common data, common events, and propagation of information.

8.2.4 Process Monitoring service (section 5.4): observes the evolving enactment state of processes, detects the occurrence of specific process events, and enacts other processes to respond to these detected events.

8.2.5 Process Transaction service (section 5.5): supports the definition and enactment of process transactions, which are process elements composed of a sequence of atomic process steps, and which are to be completed in their entirety or rolled back to their pre-enactment state.

8.2.6 Process Resource service (section 5.6): supports the assignment of process agents (e.g., tools or user roles or individual users) to enact various processes and process elements, which is typically done under constraints of time, budget, manpower assignments, equipment suites,

and process definition technology (e.g., insufficient formality may be used for totally automated enactment).

8.3 Communication Service

NB: This service description has been abstracted from the NIST/ECMA Reference Model, section 6.

Conceptual: This service provides a standard communication mechanism that can be used for inter-tool and inter-service communication. The services depend upon the form of communication mechanism provided: messages, process invocation and remote procedure call, or data sharing. This service may be built upon the framework network services, but it is also relevant when the environment does not involve a network. Communication is provided tool-to-tool, service-to-service, tool-to-service, or framework-to-framework. The services are:

8.3.1 Data Sharing service (section 6.1): Data sharing is supported via data sharing operations within the OMS or memory by other data manipulation services. Data sharing often occurs because of common agreement on the location of the data, either in memory or in various forms of secondary storage, including the OMS.

8.3.2 Interprocess Communication service (section 6.2): provides primitive operating system process communication via the RPC mechanism of the network service.

8.3.3 Network service (section 6.3): supports communication among collections of processes. If the processes are executing on different computers, then this service must be invoked to cause the actual transfer of data between machines.

8.3.4 Message service (section 6.4): supports managed communication among a large number of elements of a populated environment framework. The Message service provides communication over a distributed (in both the logical and physical senses) collection of services and tools.

8.3.5 Event Notification service (section 6.5): supports the notification of messages based upon certain triggering conditions. This service provides a consistent way for various services to report information, as in an asynchronous event, a “return code” upon exiting a program, or standard error messages reported to the user.

8.4 Operating System Services

NB: These service descriptions have been abstracted from the NIST/ECMA Reference Model, section 7.

Conceptual: These services include those services that are usually considered part of an operating system or executive. The set of services includes system process management, file management, input and output, memory management and print spoolers. These services also include timing mechanisms, device drivers, and services related to distributed systems.

Operating System services include the following:

8.4.1 Operating System Process Management service (section 7.1): provides the ability of an operating system to create processes and to schedule each independently for execution.

8.4.2 Operating System Environment service (section 7.2): provides for passing of information between operating system processes.

8.4.3 Operating System Synchronization service (section 7.3): provides for appropriate synchronization of the execution among all operating system processes in a PSE.

8.4.4 Generalized Input and Output service (section 7.4): provides basic operations to transfer data between processes and input and output devices attached to a PSE.

8.4.5 File Storage service (section 7.5): provides the basic operations to read and write files and to store and access them in directories.

8.4.6 Asynchronous Event service (section 7.6): provides for the creation and sending of signals between operating system processes.

8.4.7 Interval Timing service (section 7.7): provides for the ability to set and test timers on individual operating system processes.

8.4.8 Memory Management service (section 7.8): provides the ability to manage the main memory of the framework.

8.4.9 Physical Device service (section 7.9): provides the ability to manage the physical devices attached to the framework.

8.4.10 Operating System Resource Management services (section 7.10): provide general computer system management, including:

- System Operator services to access and control the system to allow the platform to perform properly.
- System Administration services to assume management and allocation of system services to system users.
- Capability and Security services that support the ability to control usage such that system integrity is protected from inadvertent or malicious misuse. This includes prevention of

unauthorized access, prevention of data compromise, prevention of service denial, and security administration.

8.5 User Interface Services

NB: These service descriptions have been abstracted from the NIST/ECMA Reference Model, section 8.

Conceptual: These services involve all aspects of the PSE and provide for the integration of the object management system, the process management services, and the tools themselves into a consistent set of presentation attributes between tools and users of the PSE.

8.5.1 User Interface Metadata service (section 8.1): provides for describing the objects used by the User Interface Services. While similar to the object management system Metadata service, for efficiency, many systems will create presentation schema outside of the object manager.

8.5.2 Session service (section 8.2): provides the functionality needed to initiate and monitor a session between the user and the environment. It provides the tool-to-session transformations needed to run multiple tools on multiple UI devices. This provides the essential characteristics viewed by the user (e.g., windows, colors, menus, icons).

8.5.3 Text Input service (section 8.3): provides for textual input processing by application programs. Lines of text are entered by the user via a keyboard, and the application reads the text one character, or line, at a time.

8.5.4 Dialog service (section 8.4): provides the interface between the application program and physical display devices. It includes operations for controlling execution of applications, such as starting, suspending, or aborting execution, or moving execution from foreground to background. It is the function of this service to seamlessly pass information between programs and users.

8.5.5 Display Management service (section 8.5): provides for interaction among individual windows. In general, each application writes to a specific set of windows. This includes services for capturing and redirecting command-line input and output. This includes interacting with other windows and enforcing integrity constraints between the user and the framework.

8.5.6 Presentation service (section 8.6): provides capabilities to create and manage the physical interface between the user and the PSE. This includes the screen display area as well as sight, sound, touch or other sensory interfaces in the PSE.

8.5.7 UI Security service (section 8.7): provides the security constraints needed by the UI. This requires authentication, via the Policy Enforcement services, of the user to the environment and creation of a trusted path between the user and the data to which the user has access.

8.5.8 User Interface Name and Location service (section 8.8): permits the framework to manage multi-user and multi-platform environments. It permits various sessions to communicate with various tools and various display devices. It provides the mechanism for tools to link to the appropriate display device (e.g., correct window).

8.5.9 Internationalization service (section 8.9): provides capabilities concerned with different national interests. This includes local formats for dates and other data, collating sequences and national character codes, scanning direction, and other country-specific symbols or icons.

8.5.10 User Assistance service (section 8.10): provides a consistent feedback from various tools to the user for help and error reporting.

8.6 Policy Enforcement Services

NB: These service descriptions have been abstracted from the NIST/ECMA Reference Model, section 9.

Conceptual: 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 PSE reference model regards security as a service that intersects many of the boundaries of the reference model service groupings. The set of services is:

8.6.1 Security Information service (section 9.1): supports the establishment of security information for use within the PSE. It provides the basis upon which different operational roles for users (e.g., administrators, subcontractors, programmers, managers) can be built, provides the ability to grant the same security-related privileges to groups of users, and provides confidentiality levels, which represent security classifications, and integrity levels, which represent the “purity” or “goodness” of an object, for each subject and object in a secure PSE.

8.6.2 Identification and Authentication service (section 9.2): provides for the ability to identify users and to properly associate them with appropriate access rights prior to any operations being carried out on their behalf.

8.6.3 Mandatory Access Control Service (section 9.3): provides capabilities to assign access values by a security officer to govern access to the information contained in an object.

8.6.4 Discretionary Access Control service (section 9.4): provides the ability to permit users to control (i.e., permit and deny) individual modes of access to objects that they own by individual users and all members of sets of users.

8.6.5 Mandatory Integrity service (section 9.5): provides the capabilities to protect objects from unauthorized or unconstrained modification as determined by the PSE security officer.

8.6.6 Discretionary Integrity service (section 9.6): provides the capabilities to protect objects from unauthorized or unconstrained modification as determined by a user. Since we can never be completely sure that some part or feature of the product has not been maliciously corrupted, this service provides assurance that a user may determine that a PSE maintains the “purity” or “goodness” of an object owned by the user.

8.6.7 Secure Exportation and Importation of Objects service (section 9.7): provides the ability to export and import objects in a secure manner. This is especially important since data often must pass through non-trusted channels.

8.6.8 Audit service (section 9.8): provides the ability to record information about calls on the PSE facilities in order to track and control security-related actions.

Appendix A

EXTENDED DEFINITIONS OF KEY TERMS

Environment

An environment is a collection of software and hardware¹² components; there is typically some degree of commonality that renders these components harmonious. There are certain characteristics, evidenced in the goals and aims of many existing research efforts, that an environment is likely to exhibit. The definition of an environment is actually a description based on three key characteristics of an environment.

First, environments are not restricted to facilitating engineering, but provide software support for many other processes, managerial as well as engineering, necessary to complete projects. The second characteristic is that the components of the environment will have some degree of integration, facilitating the interoperation and communication between components, sharing of data, and showing a common appearance to a user.

The third characteristic is that environments contain components at different levels of size, purpose, and complexity. Some portions of an environment comprise an infrastructure, whose main role is to provide support only for other software components rather than for end-users. These capabilities may even be invisible to an end-user. Other capabilities, however, will more likely be directly accessed by the end-user and will provide explicit support for the various activities of a project. This distinction is not always clear, and the gap between the two categories is really a spectrum, with some components spanning both rather than simply being in one or the other. Still, the distinction provides a useful structuring device for the model.

Process and Task

The concepts of “process” and “task” are based on the following definitions:¹³

Process: A set of partially ordered steps intended to reach a goal. A process is decomposable

¹²For the purposes of this document, PSESWG concentrates on the software components of an environment.

¹³These definitions are borrowed from [FEILER].

into *process steps* and *process components*. The former represent the smallest, atomic level; the latter may range from individual process steps to very large parts of processes.

Task: A process step typically enacted by a human, requiring process planning and control.

The work carried out by a project can be considered to be a set of tasks that support some particular development process. Since environments of interest to this reference model will be used in widely differing application domains to support many types of project, it is necessary that the model be general enough to be widely applicable. The model therefore does not represent particular processes or their constituent tasks; instead, it models the functionality provided by a populated environment in support of *any* chosen process.

Service

A service is an abstract description of work done by one or more software components; it is the term we use to describe the functional capabilities of an environment. By using an abstract description, we can enumerate the capabilities of an environment without reference to any particular implementation choices.

A service is self-contained, coherent, and discrete. In addition, the notion of service is essentially relative, and thus services can be composed of other services, creating a service hierarchy. Decisions about the scope of a service description, i.e., on the appropriate functional area of any particular service, are made through *ad hoc* knowledge of the expected capabilities of a populated environment. Key factors for these decisions are lifecycle phase of a project and end-user roles in the lifecycle.

There is a close relationship between services and tasks: in some ways, these two terms provide different views of the same activity. For instance, one view might be that the environment provides an editing *service*, another view being that to perform the *task* of editing a user receives support from the environment. Whichever view one takes, both refer to the same basic activity, e.g., a human making use of a piece of software, such as emacs, to create or revise textual data. We can contrast these viewpoints by noting that services are the capabilities of the environment, while tasks make use of and provide context for those capabilities. For example, in the domain of Quality Assurance, it is reasonable to refer to such things as testing a new release of a software system as a task that requires the support of services such as test case generation and report production.

Framework

The most widely accepted use of this term derives from the NIST/ECMA Reference Model [NIST]. Current [environments] distinguish between the set of facilities in support of the lifecycle project, denoted tools, and a set of (relatively) fixed infrastructure capabilities which provide support for processes, objects, or user interfaces, denoted frameworks.

The NIST/ECMA model describes a set of fifty services common to Software Engineering frameworks. These services manage information and computing resources, and also provide for tool execution, inter-tool communication, user access, and input and output for all user interactions with a computer-based collection of tools. With minor modification in the service groupings,

this document has accepted the NIST/ECMA definition of a framework and framework services.

The extent of a framework can vary both in its complexity and in the breadth of its services. In the case of complexity, a framework can span the spectrum from a minimal set of services needed for software operation to a more complete set of services that represent data and operations at higher levels of abstraction. The first of these extremes might be an operating system kernel providing minimal support for input and output and data access (e.g., POSIX 1003.1); the second extreme might be an implementation that includes a full data repository, complex user interfaces, life-cycle process management services and other comparable services (e.g., framework implementations incorporating ECMA PCTE, X Window System, etc.).

In the case of the relative breadth of framework services, an overriding factor is the domain that the framework must support. In general, the more restricted the domain, the more a service will become apparent as a common one, and thus be considered for inclusion in the framework. In the PSESWG Reference Model, the set of services included are thought to be general enough to be common to the engineering domains that are included. This may change over time, however, as more is understood about all of the domains and how they relate to one another.

Interface

The definition of interface from IEEE Software Engineering Standards [IEEE] is:¹⁴

A shared boundary across which information is passed; [a software] component that connects two or more other components for the purpose of passing information from one to the other.

This boundary, or interface, provides an external entry point for a software component that permits either invocation of the software, insertion of input to it, or reception of output from it. When the software is described in an abstract manner (as when we use the term “service”), then the interface is at a conceptual level. For instance, in the case of a data storage service and an access control service, one might assume some relationship between them that would permit one service to make use of the other; this implies a mechanism by which data or control might pass between these services. But at the conceptual level, the specific choice of mechanism by which this occurs is not of interest.

By contrast, in an actual environment, the choice of mechanism by which an interface is realized is a vital issue. The realization of an interface might include choices of formats or protocols, or could include procedures that exchange invocations and data across the shared boundary. In either case, this is called a specified interface. As an example, a requirements definition service and a design definition service in a particular environment might share data through a common format such as the proposed Common Data Interchange Format (CDIF) standard or by using a shared Schema Definition Set in PCTE.

Finally, it is useful to note the distinction between a specified interface and implementations of it by different vendors. Different implementations of the same interface may exhibit different characteristics that may have a significant effect on the practical utility of an implementation for a given project.

¹⁴ *Glossary of Software Engineering Terminology, ANSI/ISS Std 610.12-1990*

Tool

The definition of a software tool is of great importance to an environment Reference Model, since the intuitive picture of a populated environment is a framework with a set of installed tools. However, the definition of a tool is very difficult to achieve. The IEEE definition is:¹⁵

A computer program used to help develop, test, analyze, or maintain another computer program or its documentation.

This definition is useful, but is not complete. For example, it does not specify whether tools can be part of the framework. Said differently: must tools be independently executing programs (such as a compiler or editor) or can they be interfaces into the operating system (e.g., is the PC-DOS file system a tool? Is an X-Windows implementation a tool?)? These questions are probably not susceptible to simple answers, nor to answers that will have broad acceptance.

For the purposes of this reference model, however, it may be sufficient to note that whether perceived as realizing a framework service or an end-user service, a tool is an actual realization of one or more conceptual services. But there is no strict correlation between a service and a tool, since one tool may realize many services, or a single service may be realized by several tools. Tools and services are in many ways similar, but are not the same thing.

¹⁵ *ibid.*

Appendix B

COMMON PROJECT ACTIVITIES AND THEIR RELATION TO REFERENCE MODEL SERVICES

The purpose of this appendix is to describe several activities typically performed by users of a PSE as part of project execution. The need for this description is that, while many project activities occur with a one-to-one agreement between Reference Model services and a user's tasks, this correspondence between task and service is not evident for all activities. For example, there is a one-to-one agreement between the Software Design service and the task of creating a design before building a software product. Similarly, programmers need to compile source programs, and they make use of the Compilation service through the functionality of such tools as compilers and preprocessors. However, not all tasks have corresponding services in the reference model. For example, a common task is often called quality assurance, yet there is no Quality Assurance service in the model. This is because the task of quality assurance uses existing services already present in the model.

The following presents several common tasks and the set of services that may be used to implement parts of them. In almost all cases, common support services like Text Processing and User Communication services will be needed and will not explicitly be mentioned.

B.1 Management Activities

B.1.1 Acquisition Management

Acquisition management supports the activities necessary to develop, award, and track procurements. While some of the examples used here are expressed in terminology common to

government acquisitions, the concepts are applicable to acquisition activities in general.

Generally, these acquisition activities are developed in conjunction with the Proposal Preparation activity, described later. That is, an organization, typically a governmental unit, will plan for an acquisition (Acquisition Planning) and develop a request for procurement (RFP) by performing RFP Generation. In response, other organizations will respond to the RFP with the Proposal Preparation task. The acquiring unit then performs a Proposal Evaluation task, choosing from among the submitted proposals.

Examples of Acquisition Management activities include:

Acquisition planning. Creation of the acquisition plan – **Services:** Project Management Planning, Estimation and Risk Analysis services.

RFP generation. Create, maintain and modify the statement of work – **Services:** Technical services such as System Requirements Engineering and Software Requirements Engineering services as well as Project Management services such as Estimation Service and the Numeric Processing service.

Proposal evaluation. Evaluate set of submitted proposals – **Services:** Numeric Processing and Estimation services.

Acquisition tracking. Monitor contract once it is awarded – **Services:** Project Management Estimation, Planning, Risk Analysis and Tracking services.

B.1.2 Project Management

Project Management activities are those that track and manage the development of a project from concept to completion. Examples of these activities include:

Proposal preparation. Develop proposal in response to RFP – **Services:** Technical Engineering services such as System Requirements Engineering, Software Requirements Engineering, System and Software Design services, Project Management Planning, Estimation and Risk Analysis services and Numeric Processing, Publishing, and Presentation Preparation services.

Project Management. Plan and execute project from concept through deployment – **Services:** Project Management Planning, Estimation, Risk Analysis and Tracking services.

Configuration Management. Ensure traceability and reproducibility of a project's end products – **Services:** System Integration, Software Build, System and Software Traceability, Configuration Management, Change Management, and Reuse Management services.

B.1.3 Quality Assurance

The purpose of Quality Assurance (QA) is to assure that the product meets certain standards before it is delivered by the developer or accepted by the purchaser. Reliability and correctness of the source programs are certainly important components of QA, but QA includes many other attributes.

Examples of Quality Assurance activities include:

Quality assurance planning. Quality objectives must be determined and data needed to measure such quality must be determined – **Services:** System Requirements Engineering, Software Requirements Engineering, Metrics, and Risk Analysis services.

Develop test plans. Develop test plans for achieving quality objectives – **Services:** System Requirements Engineering, Software Requirements Engineering, System and Software Testing, Code Verification, Configuration Management and Traceability services.

Quality assurance testing. Quality objectives are monitored and tested – **Services:** System and Software Testing and Metrics services, as well as Project Management Estimation and Tracking services.

B.2 Engineering Activities

B.2.1 System Engineering

System engineering involves those activities that support the technical development and maintenance of hardware and software components of a project. For the most part, these activities fall into the services described by the System Engineering services, but include other services as well.

Typical System Engineering activities include:

System requirements analysis. Develop requirements and specifications – **Services:** System Requirements Engineering, System Design and Allocation, and System Simulation and Modeling Service.

System development. Build the product – **Services:** System Engineering services with software components developed by the Software Engineering services.

System deployment. Operation and maintenance of the product – **Services:** System and Software Traceability and Testing services, Configuration Management and System and Software Design services.

B.2.2 Software Engineering

Software engineering activities are those activities involved in building and maintaining the software components of a product. For the most part, these tasks use the Software Engineering Services.

Typical Software Engineering activities include:

Software requirements engineering. Develop software requirements – **Services:** Software Requirements Engineering, Software Design and Software Modeling services.

Software development. Build the software components – **Services:** all of the Software Engineering Services.

Software deployment. Operation and maintenance of the software – **Services:** Software Traceability and testing services, Configuration Management Services, Reuse Management, Software Reverse Engineering and Software Re-engineering services, and the Software Design service.

B.2.3 Process Engineering

Process engineering activities develop the steps in the development process that are to be taken by other members of the development group. The process may be relatively fixed (e.g., following a strict “waterfall” development using specific design method, specific compiler, specific testing and validation tools) or may be partially or totally dynamic (e.g., testing method depends upon the results of the previously performed code verification activity). For the most part, these tasks make use of the Life Cycle Process Management services.

Process Engineering activities include:

Process definition. Define development process – **Services:** Process Definition, Process Library and Process Exchange services. Processes may be enacted using the Process Usage service.

Process enactment. Perform the set of development processes – **Services:** Process Usage service.

B.3 Supportability Activities

B.3.1 Logistics Support

Supportability and logistics activities ensure the operational availability of systems, including supportability, readiness and survivability. For computer-based products, logistics supports the

operation and maintenance of such systems. Although many logistics operations are outside of the purview of a PSE (e.g., several aspects of personnel training, payroll issues), many are fully covered by existing PSE services.

The following are those logistics tasks that will undoubtedly be part of the operational characteristics of a PSE.

Supply support. Support the identification, selection for acquisition, cataloging, receipt and storage, provisioning, issue and disposal of the component parts of a computer-based product – **Services:** Configuration Management, Numeric Processing, Estimation, Risk Analysis services, as well as most of the Acquisition Management tasks mentioned above.

Personnel support. Support personnel requirements including training and operational requirements – **Services:** Mostly outside the purview of a PSE, although data may be stored in and make use of PSE object management system. May use Project Management Planning, Estimation, and Tracking services.

Documentation support. Maintain logistics support and product technical documentation – **Services:** Text, Numeric and Figure Processing services, Publishing and Presentation Preparation services, and Configuration Management services.

Computer resources support. Support the management of the logistics support facility – **Services:** Project Management Services of Planning, Estimation, Risk Analysis and Tracking services.

B.3.2 Operation and Maintenance

Post-deployment logistics maintains the product in the field. Errors or anomalies must be tracked from their source to the maintenance organization, and the distribution of corrections or new system enhancements from the maintenance organization to the field must be supported and monitored.

Error correction. Correction of errors and anomalies found using the product – **Services:** Change Management and Configuration Management services, Project Management services, such as Planning, Estimation and Tracking, and Technical Engineering services for correction of errors.

System enhancement. Modification of the product due to changed requirements – **Services:** Configuration Management, Re-engineering, Reverse Engineering and all other Technical Engineering services to produce a new version of the product.

Appendix C

RATIONALE

1. What different users/uses are there of the reference model?

The PSESWG intends to use this reference model as a source document for the identification of interfaces in a PSE. Once those interfaces have been identified, it will be possible to examine them and determine those that it would be beneficial to standardize. There are likely to be two sorts of interfaces in this category: those for which candidate standards exist or which are being actively examined by organized standardization efforts and those for which no such activity can be identified. The first kind will drive the organized selection process PSESWG will use to determine the contents of the military standard it is chartered to produce. The second kind will be used as the basis for encouraging appropriate research, development, and standardization efforts.

In addition to identifying potential standards, this reference model can be used in many other ways:

- To understand the architecture of a proposed or realized system.

An actual or proposed product (for example, a tool or framework) can be characterized in terms of the elements (services and relationships) of the reference model and the explicit realization of those elements as a set of operations and data objects. In some ways we can see this as a cross-section or instantiation of the PSE reference model. Such a characterization can help others to better understand the product thus described.

- To compare products

Comparing different PSE products is difficult without a consistent conceptual model within which to analyze all products. The descriptions of products through services provides a common vocabulary for discussing them and helps to ensure that any comparison compares like with like. In addition, the categorization of services into end-user and framework services means that products can be compared at different abstract levels: comparison of abstract functionality (end-user services) and comparison of support mechanisms (framework services).

- To describe a proposed or required system.

The PSE reference model can be used as the basis for describing a set of PSE requirements by giving the services corresponding to a required system, as opposed to an actual system. This allows the requirements to be described in an abstract way, in terms of required services and the interface among those services. This is independent of particular implementation constraints, which can then be examined in the light of the abstract requirements.

- To discuss implementation of services.

The separation of end-user and framework services means that particular tools and framework realizations will be characterized as providing essentially equivalent end-user services using different framework services. For example, the end-user service on inter-tool communications can be realized via different framework services: a remote procedure call mechanism, message server facilities, data sharing with triggers, and so on.

- To examine product integration issues.

By describing services of actual PSE products a characterization of both their abstract functionality and implementation mechanisms is provided. When users wish to determine the extent to which those products can be integrated, these descriptions provide the necessary basis for answering important questions regarding the ease with which the integration can take place. For example, the end-user service aspects can reveal the extent to which the products provide similar services, while the framework aspects allow issues of mechanism interoperation to be discussed. Hence, a PSE integrator may use the PSE services reference model to determine, for a collection of tool products to be integrated, what services each tool provides and, based on the overlap of provided services and available base computing environment, to develop a strategy for integration in terms of a particular environment architecture, identifying interfaces relevant for its realization.

In summary, we note that in presenting the PSE reference model we abstracted from a notion of tools and framework realizations towards higher level concepts of services and interfaces. In examining an actual tool or PSE product, the reference model can be used to reflect issues of functionality and architecture by allowing an abstract description of that product to be produced.

2. Why a service-based approach?

The approach taken in deriving this reference model is one based on services. There are a number of reasons why this was chosen as the most effective means:

– Path of least resistance and most familiar: Functional decomposition is a straightforward approach for most people working in this area. Many people think of their environments in terms of “what the PSE does” for them, and this thinking is well-captured in a service-based approach.

– Nature of related reference models: The PSE reference model builds upon the work of other related models. Both the NIST/ECMA reference model and the reference model in the POSIX

Guide to Open Systems Environments had already (independently) taken very similar service-based approaches. In order to be able to capitalize on this wealth of available work, it made good sense to adopt a compatible approach.

– Natural fit with end goal: A major goal of the PSE reference model is to help identify interface areas for standardization. Although such standards are largely known as “interface standards,” they are by-and-large described from a service-based viewpoint. This consistency will make it much easier to use the reference model in the identification of candidate interface standards than other approaches.

It should also be understood that taking a service-based approach at this point in the evolution of the reference model does not mean that other viewpoints were not considered or will not be found to be important in the future. Early discussions about the approach to be taken recognized that a complete reference model might well include a number of different points of view of a PSE. But it was concluded that one of them had to be first and that a service-based approach was at least as viable as any of the others.

A data-oriented approach is often mentioned as an alternate to the service-based approach used here. It is highly likely that such an approach will become very important in the process of understanding the data interface requirements of the services articulated in this reference model, and may play a role in the future.

A process-oriented approach is also mentioned as an alternate. It is actually not far removed from the approach taken. As described in chapter three, the determination of end-user services was driven largely by knowledge and consideration of end-user processes and the service requirements they generate.

3. How did we select services?

The selection of services has been guided by a number of important principles. These include:

- By considering typical activities that a PSE supports, it is possible to define the functionality that is necessary in order to support those activities. For example, in considering the maintenance activities of a typical software development project, it is possible to ask yourself the question “what functionality would I need or expect from a PSE to support software maintenance activities?”. It is those support services that we have captured in this reference model.
- There are a wide range of potential users of a populated PSE. This includes many forms of project engineers, project managers, administrative staff, and PSE support staff. Considering the required functionality of a PSE from each of these PSE users’ perspectives provides a useful way to describe a set of PSE services.
- A number of existing studies have described the expected functionality of some part of a populated PSE in terms of a set of service descriptions. We have analyzed and expanded on this work.

The result is a model that provides a description of the functionality that can be expected from a populated PSE without being tied to a particular architecture for implementing a PSE, tools that must be part of the PSE, or expected uses and users of the PSE.

4. How did we group services?

Grouping of services has been based on a combination of factors which imply a coherence to those services. In many cases this coherence is a result of a functional relationship between the services (e.g., the OMS services), a temporal relationship (e.g., System Requirements Engineering services), or based on the expected role of the users (e.g., PSE User and Role Management service). In all cases the aim of the group is to provide an abstraction of those services that allows them to be discussed as a whole without concern for the details of which services form part of that group.

5. How do interfaces facilitate/relate to integration?

The use of the reference model for PSESWG is to act as a basis for identification of interface areas where existing standards exist, or where future standards might be profitable. By identifying these standards, the possibility exists that tools (potentially from different vendors) will be available that support the standard.

The consequence of this tool support for standards is that a basis is provided through which sharing is possible. Integration of tools is facilitated by selecting and agreeing interface standards, but it is not a necessary consequence of standard interfaces. In general, the interfaces provide the syntactic agreements on which semantic agreements between tools can be built. Without the interface standards providing that syntactic level agreement, the more useful semantic agreements that are needed are less likely, and more costly to implement.

Appendix D

ABBREVIATIONS and ACRONYMS

4GL	Fourth Generation Language
ANSI	American National Standards Institute
APPL/A	A Process Programming Language based on Ada
APSE	Ada Programming Support Environment
ASCII	American [National] Standard Code for Information Interchange
ASIC	Application Specific Integrated Circuit
ASSET	Asset Source for Software Engineering Technology
CAD/CAM	Computer-Aided Design/Computer-Aided Manufacture
CAI	Computer-Aided Instruction
CARDS	Central Archive for Reusable Defense Software
CASE	Computer-Aided Software, System Engineering
CSCI	Computer Software Configuration Item
CDIF	CASE Data Interchange Format
CEARM	Conceptual Environment Architecture Reference Model
CM	Configuration Management
COCOMO	Constructive Cost Model
COTS	Commercial off-the-shelf
CPL	Common Prototyping Language
CPS	Cornell (University) Program Synthesizer
CPU	Central Processing Unit
DCDS	Distributed Computing Design System
DOD	Department of Defense
DOS	Disk Operating System
EAST	Environment of Advanced Software Technology
ECMA	European Computer Manufacturers Association
EIS	Engineering Information Systems
EQN	Equation preprocessor for nroff and troff
GOTS	Government off-the-shelf

GSFC	(NASA) Goddard Space Flight Center
HIPO	Hierarchical Input Process Output
HOOD	Hierarchical Object-Oriented Design
IDE	Interactive Development Environments
IDL	Interface Description Language
IEEE	Institute of Electrical and Electronics Engineers
ILS	Integrated Logistics Support
ISEE	Integrated Software Engineering Environment
LAN	Local Area Network
LEX	Lexical Analysis Program Generator
LSE	Language Sensitive Editor
MTBF	Mean time between failures
NASA	National Aeronautics and Space Administration
NGCR	Next Generation Computer Resources
NIST	National Institute of Standards and Technology
OMS	Object Management System
OODBMS	Object-Oriented DataBase Management System
ORCA	Object-based Requirements Capture and Analysis
OS	Operating System
PAL	Process Asset Library
PC	Personal Computer
PCTE	Portable Common Tool Environment
PMDB	Project Master Data Base
POSIX	Portable Operating System Interface for Computer Environments
PSE	Project Support Environment
PSESWG	Project Support Environment Standards Working Group
QA	Quality Assurance
RAPID	Reusable Ada Packages for Information System Development
RDBMS	Relational DataBase Management System
RETRAC	Requirements Traceability
RFP	Request For Proposal
RM	Reference Model
RPC	Remote Procedure Call
RTNI	Real-Time Non-Intrusive Instrumentation
SADT	Structured Analysis and Design Technique
SEE	Software Engineering Environment
SLCSE	Software Life-Cycle Support Environment
SME	Software Management Environment
SOW	Statement of Work
StP	Software Through Pictures
TBL	Table formatting preprocessor for nroff and troff
TC33	Technical Committee 33
TCOS	Technical Committee on Operating Systems
UI	User Interface
VDM	Vienna Development Method

WAN	Wide Area Network
WBS	Work Breakdown Structure
WYSIWYG	“what you see is what you get”
YACC	Yet Another Compiler Compiler

Appendix E

REFERENCES

- NIST** *Reference Model for Frameworks of Software Engineering Environments*. NIST Special Publication 500-211, August 1993; ECMA TR/55 Edition 3, June 1993.
- POSIX** *Draft Guide to the POSIX Open Systems Environments*. P1003.0, June 1992.
- OSSWG** *Reference Model for Embedded Operating Systems*. NGCR Operating System Standards Working Group, June 1990.
- FEILER** *Software Process Development and Enactment: Concepts and Definitions*. Peter Feiler and Watts Humphrey, SEI, 1991.
- IEEE** *Glossary of Software Engineering Terminology*. ANSI/ISS Std 610.12-1990.

Index

- Abbreviations, 103
- Acquisition Management Activities, 93
- Acronyms, 103
- Activities
 - Acquisition Management, 93
 - Engineering, 95
 - Logistics Support, 96
 - Management, 93
 - Operation and Maintenance, 97
 - Process Engineering, 96
 - Project Management, 94
 - Quality Assurance, 94
 - Software Engineering, 95
 - Supportability, 96
 - System Engineering, 95
- Activities and Services, 93
- Administration Services (of PSE), 71
- Annotation Service, 65
- Audio and Video Processing Service, 63
- Authentication, 76

- Background, 1
- Build Service, Software, 36
- Bulletin Board Service, 69

- Calendar and Reminder Service, 64
- Change Management Service, 47
- Comments on RM, Submission, 113
- Common Support Services, 60
- Compilation Service, 32
- Conceptual Dimension, 13
- Conferencing Service, 70
- Configuration Management Service, 45
- Customization and Installation (of Tool) Service, 71

- Debugging Service, 34
- Definitions of Key Terms, 89

- Diagnostic Service, *see* PSE Diagnostic Service
- Dimension
 - Conceptual, 13
 - Examples, 14
 - External, 14
 - Internal, 14
 - Operations, 13
 - Relationships, 14
 - Rules, 13
 - Types, 14

- End-User Services, 9
- Engineering Activities, 95
- Engineering Services, Technical, 17
- Environment, 6, 89
 - vs.* Conceptual Model, 10
- Estimation Service, 55
- Examples Dimension, 14
- External Dimension, 14

- Figure Processing Service, 62
- Framework, 90, 91
- Framework Services, 6, 10, 79

- Grouping, 10

- Host-Target Connection Service, 24

- Installation and Customization (of Tool) Service, 71
- Interchange Service, *see* PSE Interchange Service
- Internal Dimension, 14

- Key Terms, Definitions, 89

- Life-Cycle Process Engineering Services, 40
- Logistics Support Activities, 96

Mail Service, 68
 Management Activities, 93
 Management Services
 Project, 53
 Technical, 45
 Metrics Service, 50
 Model, 7
 Conceptual vs. Actual Environment, 10
 Discussion, 10
 Model Description, 5
 Modeling
 Software Simulation and Modeling Service, 29
 System Simulation and Modeling Service, 20

 Next Generation Computer Resources, *see* NGCR
 NGCR, xi, 1, 12
 Numeric Processing Service, 61

 Operation and Maintenance Activities, 97
 Operations Dimension, 13

 Planning Service, 54
 Process, 89
 Process Definition Service, 41
 Process Engineering Activities, 96
 Process Exchange Service, 42
 Process Library Service, 42
 Process Usage Service, 43
 Project Activities, 93
 Project Management Activities, 94
 Project Management Services, 53
 Project Support Environment Standards Working Group, *see* PSESWG, 1
 PSE Administration Services, 71
 PSE Diagnostic Service, 74
 PSE Interchange Service, 75
 PSE Resource Management Service, 73
 PSE Status Monitoring Service, 74
 PSE User Access Service, 75
 PSE User and Role Management Service, 72
 PSESWG, xi, 1–3

 Quality Assurance Activities, 94

 Rationale, 99
 Re-engineering
 Software Re-engineering Service, 38
 System Re-engineering Service, 24
 Reading the Service Descriptions, Notes, 13
 Reference Model, 7
 References, 107
 Relationships Dimension, 14
 Reminder and Calendar Service, 64
 Resource Management, *see* PSE Resource Management Service
 Reuse Management Service, 49
 Reverse Engineering Service (Software), 37
 Risk Analysis Service, 55
 Role Management, *see* PSE User and Role Management Service
 Rules Dimension, 13

 Scope of the Model, 3
 Service, 6, 7, 90
 Administration (of PSE) Services, 71
 Annotation, 65
 Audio and Video Processing, 63
 Bulletin Board, 69
 Calendar and Reminder, 64
 Change Management, 47
 Common Support Services, 60
 Compilation, 32
 Conferencing, 70
 Configuration Management, 45
 Customization, *see* Tool Installation and Customization
 Debugging, 34
 End-User Services, 6
 Estimation, 55
 Figure Processing, 62
 Framework Services, 6
 Host-Target Connection, 24
 Information Management, 48
 Installation, *see* Tool Installation and Customization
 Life-Cycle Process Engineering Services, 40
 Mail, 68
 Metrics, 50

- Numeric Processing, 61
- Planning, 54
- Process Definition, 41
- Process Exchange, 42
- Process Library, 42
- Process Usage, 43
- PSE Administration Services, 71
- PSE Diagnostic, 74
- PSE Interchange, 75
- PSE Resource Management, 73
- PSE Status Monitoring, 74
- PSE User Access, 75
- PSE User and Role Management, 72
- Publishing, 65
- Reminder, *see* Calendar and Reminder Service
- Reuse Management, 49
- Risk Analysis, 55
- Software Build, 36
- Software Design, 28
- Software Engineering Services, 27
- Software Generation, 31
- Software Re-engineering, 38
- Software Requirements Engineering, 27
- Software Reverse Engineering, 37
- Software Simulation and Modeling, 29
- Software Static Analysis, 33
- Software Testing, 35
- Software Traceability, 39
- Software Verification, 30
- System Design and Allocation, 19
- System Engineering Services, 18
- System Integration, 23
- System Re-engineering, 24
- System Requirements Engineering, 18
- System Simulation and Modeling Service, 20
- System Static Analysis, 21
- System Testing, 22
- System Traceability, 26
- Target Monitoring, 25
- Text Processing, 60
- Tool Installation and Customization, 71
- Tracking, 56
- User Communication, 68
- service
 - Framework Services, 79
- Service Descriptions, Notes on Reading, 13
- Service Groups, 8
- Simulation
 - Software Simulation and Modeling Service, 29
 - System Simulation and Modeling, 20
- Software
 - Build Service, 36
 - Design Service, 28
 - Engineering Activities, 95
 - Engineering Services, 27
 - Generation Service, 31
 - Re-engineering Service, 38
 - Requirements Engineering Service, 27
 - Reverse Engineering Service, 37
 - Simulation and Modeling Service, 29
 - Static Analysis Service, 33
 - Testing Service, 35
 - Traceability Service, 39
 - Verification Service, 30
- Static Analysis Service
 - Software, 33
 - System, 21
- Status Monitoring, *see* PSE Status Monitoring Service
- Submission of Comments on RM, 113
- Support Services, 59, *see* Common Support Services
- Supportability Activities, 96
- System
 - Design and Allocation Service, 19
 - Engineering Activities, 95
 - Engineering Services, 18
 - Integration Service, 23
 - Re-engineering Service, 24
 - Requirements Engineering Service, 18
 - Simulation and Modeling Service, 20
 - Static Analysis Service, 21
 - Testing Service, 22
- Target Monitoring Service, 25
- Target System, 12
- Task, 6, 7, 89

Technical Engineering Services, 17
Technical Management Services, 45
Testing Service
 Software, 35
 System, 22
Text Processing Service, 60
Tool, 6, 92
Tool Installation and Customization Service,
 71
Traceability Service
 Software, 39
 System, 26
Tracking Service, 56
Types Dimension, 14

User Access Service, *see* PSE User Access
 Service
User and Role Management Service, 72
User Communication Services, 68

Verification, *see* Software Verification Ser-
 vice
Video, *see* Audio and Video Processing

SUBMISSION OF COMMENTS

When you submit comments on Version 2.0 of the Reference Model, please send them by electronic mail to the following address:

djc@sei.cmu.edu

If you do not have access to an electronic network, please send the comments by postal mail or FAX to:

David Carney
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3890
phone: (412) 268-5758
fax: (412) 268-6525

To assist us in the processing and tracking of your comments, please use the format below for each comment.

```
! NAME ...
! PHONE ...
! FAX ...
! EMAIL ...
! MAIL
: multi-line address
! DATE ...
! SECTION ...
! VERSION 1.0
! TOPIC ...
! COMMENT
: text of comment
! RATIONALE
: text of rationale
! END
```

The NAME line contains your name or affiliation (or both).

The PHONE line contains your phone number.

The FAX line contains your FAX phone number.

The EMAIL line contains your electronic mail address.

The lines following the MAIL line contain your postal mailing address.

The DATE line includes the date of you comment. It should be in ISO standard form (year-month-day), for example, 4 July 1992 is 92-07-04.

The SECTION line should include the Reference Model section number and title, for example, "4.2.7 Software Static Analysis Services." To help identify it better, this line can also include the page number.

The TOPIC line should contain a one-line summary of the comment. This line is essential.

The lines following the COMMENT line contain your request for a change, an addition, a deletion, or anything else about the Section. This can be as long or as short as necessary. When you make suggested wording changes or additions, please be as specific as possible.

The lines following the RATIONALE line explain why the suggested change(s) (if any is requested) should be made. Please be as clear and concise as possible.

The END line marks the end of the comment form.

A sample comment is shown below for illustration.

```
! NAME A. Reviewer, ABC Inc.
! PHONE 909-555-5555
! FAX 909-555-4444
! EMAIL reva@lizard.abc.com
! MAIL
A. Reviewer
ABC Inc.
MS:23AB-WX
1234 Somestreet St.
Somecity, XX 98765
! DATE 93-11-03
! SECTION 7.1.2 Numeric Processing Service
! VERSION 1.0
! TOPIC Numeric Processing should not include formatting of formulae
! COMMENT
The text discussing the formats and formula strings should be removed or moved to the
section on Text Processing. Also the examples of EQN and TBL should be removed.
! RATIONALE
The Numeric Process Service should provide calculation operations of a numerical nature it
should not include the operations that are purely text formatting and document processing
in nature.
! END
```