ECMA
European Computer
Manufacturers Association

NIST
National Institute of Standards and Technology
United States Department of Commerce

REFERENCE MODEL FOR FRAMEWORKS OF SOFTWARE ENGINEERING ENVIRONMENTS

Edition 3 of

Technical Report ECMA TR/55

NIST Special Publication 500-211

Prepared jointly by NIST and ECMA

prepared by the
NIST ISEE Working Group and the ECMA TC33 Task Group on the Reference Model

August, 1993

Brief History

Work on a reference model for Software Engineering Environments (SEEs) has been in progress for the past years on both sides of the Atlantic. A key objective of the work in reference models for SEEs has been to find better ways to describe SEEs and to assist the SEE architectural standardization process. The reference model has gained increasing interest in the software environments community. It is hoped that the concepts described within the reference model can guide the evolution of SEE environment architectures.

When the European Computer Manufacturers Association (ECMA) Technical Committee for Portable Common Tool Environment (PCTE) standardization was formed, one aim was to create an environment framework reference model (RM) to assist the standardization process. A Task Group on the Reference Model (TGRM) was formed in 1988 to develop a complete reference model. During 1989 the first full version of the reference model was created, and during 1990 the model was evaluated by using it to describe a number of environment frameworks. That version of the reference model was published as an ECMA technical report in December 1990.

The model covers the set of services needed to describe environment frameworks. The particular services of the model are described to a degree of detail that is complete enough for the model to be used to describe existing systems and proposals. Although ECMA TC33 created ECMA TC33/TGRM to develop a model that could be used to describe PCTE, subsequent work on the reference model was totally independent of PCTE and was not intentionally biased towards it.

Beginning in 1989, the NIST Integrated Software Engineering Environment (ISEE) Working Group has also been developing a reference model for SEEs. This Working Group decided to adopt the ECMA RM as a basis for its own work and has enhanced and extended the model to support NIST goals. To test the model, it was used by NIST ISEE and ECMA TC33/TGRM to map several existing products to the RM in order to test the effectiveness and coverage of the services specified (CAIS-A [30], PCTE [35], SLCSE [80], AD/cycle [49], ATIS [26], Emeraude PCTE, and COHESION [14]).

Edition 2 of this document, published December 1991, represents the results of discussions held at various NIST ISEE workshops and comments provided by the participants of these workshops, as well as a review by the members of ECMA TC33/TGRM.

During 1992, a Reference Model Change Control Board, consisting of members of ECMA TC33/TGRM and the NIST ISEE Working Group, met several times to prepare this third edition of this document in response to receipt of approximately 200 requested changes. The U.S. Navy's NGCR PSESWG program was particularly helpful in pointing out issues that needed clarification.

Major changes for this third edition include the addition of operating system services, enhanced user interface services, a rewritten policy enforcement clause, and replacement of the graphic describing the model.

This report is published jointly as an ECMA Technical Report and a NIST Special Publication.

Accepted as an ECMA Technical Report by the General Assembly of June, 1993.

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

Table of Contents

Section I. Software Engineering Environment Frameworks

1 Scope

This document describes a reference model (RM) for Software Engineering Environment (SEE) Frameworks. This clause provides some motivation for the work presented in this document. It then provides the key aims of the SEE RM. Following clauses describe the structure of the SEE RM and the services that comprise it.

1.1 Scope of Software Engineering Environments (SEE) and SEE Frameworks

Software Engineering means the planned process of producing well-structured, reliable, good-quality, maintainable software systems within reasonable time frames[76]. *Software Engineering Environment* means the system[15] which provides automated support of the engineering of software systems and the management of the software process.

Alternative equivalent names for an SEE are IPSE (Integrated Project Support Environment), CASE (Computer Assisted Software Engineering), SDE (Software Development Environment), ISEE (Integrated Software Engineering Environment), and ISF (Integrated Software Factory). No distinction is made between these terms in this document.

A Software Engineering Environment deals with information about: a) the software under development (e.g., specifications, design data, source code, test data, and project plans); b) project resources (e.g., costs, computer resources, and people working on the production of software and their assignments and management duties); and c) organization policy, standards and guidelines on the production of software. An SEE may store information about target hardware for which software is being developed, but it typically does not store information about the design or development of the hardware. An SEE typically does not deal with company finances or marketing information because these do not relate directly to software engineering.

SEEs are typically built on hardware and operating system platforms. Current SEE architectures distinguish between the set of facilities in support of the life cycle project, usually denoted *Tools*, and a set of (relatively) fixed infrastructure capabilities which provide support for processes, objects, or user interfaces, denoted *SEE Frameworks* [54]. A major purpose of frameworks is to simplify the construction of tools by providing a set of commonly needed facilities, key integration components, and support for higher level constructs than those found in typical operating systems. Another purpose may be to support the porting of environments across a variety of hardware configurations and native operating systems. Tools may also use services provided by other tools, and framework components may use services from other framework components.

Groups needing access to framework components include the following:

Platform suppliers. They need to provide the hardware and operating system platforms to access the resources of the environment and provide the primitive functionality used by the framework.

Environment suppliers. They need to provide the framework components for interfacing between the user and the resources of the platform and provide most of the functionality described by this reference model.

Tool suppliers. The work of producing SEE components (e.g., additional framework components, tools, and generic utilities) is typically carried out by *environment* and *tool builders*. This level provides the "value added" needed to make the framework into an effective environment for solving environment user needs.

Users. They use the environment for solving various application problems. Designers, developers, managers, environment administrators, configuration controllers, and secretaries use the customized environment to build target systems according to the customers' software requirements. Some classes of users will inevitably use certain framework services directly (e.g., query and backup on the project's environment

database), analogous to direct usage of OS command on many systems in the past, rather than always using higher level functional abstractions which are tools.

In addition, SEEs may be adapted or instantiated by *environment adaptors* in order to produce specific environments. SEEs may also be extended or customized according to a defined process produced by *process developers* to support one or more processes or software engineering methodologies.

1.2 Scope of the SEE Frameworks Reference Model

The SEE frameworks reference model is **neither** an architecture nor a standard. By itself, it cannot be used to evaluate the effectiveness of an SEE for a particular application; other criteria and measurements need to be defined for that purpose.

This document provides a reference model for SEE frameworks and describes the role that the framework plays in an entire SEE. Therefore, some of the discussions may mention environments as a whole in addition to frameworks specifically. It does address tools and makes some statements about them. It does not, however, attempt to classify all types of tools or to relate the use of tools to particular methods of software development. Of particular concern in an SEE is the degree of integration among the various components in the environment. Clause 3.1 briefly introduces these integration issues. The RM may also be expanded to include non-service aspects of SEEs, such as models, properties, and characteristics.

1.3 Aims of the Reference Model

An SEE frameworks reference model is a conceptual (and functional) basis for describing and comparing existing SEEs or SEE components (including framework components). The purpose of this clause is to describe and provide rationale for the main requirements and aims of this reference model. The objective is to keep these aims in mind while the reference model is developed and discussed.

Description and Comparison

- The reference model should be suitable to describe, compare, and contrast existing and proposed environment frameworks.

This requirement enables validation that the reference model has the correct scope for addressing issues concerned with building environment frameworks. The area of SEEs is constantly evolving together with its related concepts and terminology. Much effort is often wasted through misunderstandings and misinterpretations when different groups meet to discuss problems and potential solutions or new approaches to problems.

The aim of the reference model is **not** to determine whether one system is better than another nor to define problems or impose solutions. One of its major contributions is to provide the necessary basis for discussing SEE frameworks, i.e., to be used as a vehicle to explore environments and to identify their capabilities, strengths, and weaknesses [13] [89].

Standard Identification and Relationship

- The reference model should provide a basis for the identification of existing and emerging SEE interface standards and their relationships.

There are various groups currently working on the definition and implementation of existing interface standards for SEEs and SEE frameworks. Unfortunately, their nomenclature and underlying models differ, posing difficulties for the identification of overlapping capabilities.

The RM may help in the identification of existing or emerging SEE interface standards and how they fit with respect to each other, by providing a common basis for description of their capabilities. The RM may also assist in identifying gaps in existing standards with respect to SEE architectural needs.

Standards may provide a way for environment users to buy facilities that meet their requirements from vendors who are not directly cooperating and may even be competitors. Standards are a means for achieving an open systems environment where multiple vendors provide competing services on compatible hardware and software platforms. Standards may be very useful in enabling tools to be ported to, or to interwork over, different types of hardware.

Integration and Interoperability

- The reference model should address interoperability and integration of tools.

It is essential that frameworks provide for tool integration and interoperability. The mechanisms involved in providing a coherent integrated environment with a diverse set of tools cooperating within a common environment framework are only now starting to be understood. Clause 3.1 provides a brief overview of these issues and discusses how framework mechanisms may evolve over time to address this issue.

Degree of Generality

- The reference model should be usable to describe a wide range of SEE framework designs, but should balance this against a requirement to be able to define points at which useful standards may be defined.

This requirement really reflects two possible extreme cases. A reference model may be formulated which remains at a level of abstraction above explicit designs for SEE frameworks. This may satisfy many of the requirements placed upon it. It is very unlikely that such a reference model provides a good framework for positioning standards.

At the other extreme, a reference model may be proposed which prescribes a single design for SEE frameworks. In this case, the places where standards may be defined would be obvious, but there would be a lack of any sense of generality. Such a model may not allow discussion about SEE frameworks which did not conform to the particular design chosen.

The reference model should find a way of enabling the general and specific to coexist.

Relationships among RM Elements.

- The reference model should recognize the importance of the relationships among its elements.

Understanding how SEE components relate to each other both statically and dynamically is not trivial. The RM may help in identifying these relationships.

It is also important to realize that this model is not an architecture for an environment framework. The model describes a set of services that frameworks may provide. The mechanisms for instantiating these services is up to the designers and implementors of frameworks.

Software Development Method Independence

- The reference model should cover all system aspects irrespective of implementation techniques or software development methods employed by particular SEEs or systems developed within SEEs.

The reference model should recognize that there are many approaches to software development that an environment may support and not support one to the exclusion of others.

The reference model should not be unduly influenced by existing SEE frameworks. There is very little experience of their application to real projects, and thus there is no hard evidence that their designs are the most appropriate. However, existing products and the results of research projects provide a useful body of knowledge applicable to a reference model.

Education

- The reference model should be useful as the basis for educating systems engineers in the subject of SEE frameworks.

Even though the RM is to be used by people familiar with SEEs, it is important that its description be clear and precise. It must provide examples to help clarify its concepts. Therefore, it is envisioned that the SEE RM may be used for educational purposes.

Unifying Concepts

- The number of unifying concepts required to describe the reference model should be small.

As a general principle, the fewer the number of concepts employed, the easier the reference model will be to understand, although this principle should not be taken to its extreme.

Related Models

- The reference model should be compatible with other appropriate reference models.

There are other reference models which address standards and related SEE architecture issues. It is desirable that the RM does not duplicate work done elsewhere, if these related models are deemed appropriate for insertion into the RM.

1.4 List of Acronyms

Acronyms used in this report include:

ANSI	American National Standards Institute
APPL/A	A Process Programming Language based on Ada
ASCII	American Standard Code for Information Interchange
ATIS	A Tools Integration Standard
CAIS-A	Common APSE (Ada Programming Support Environment) Interface Set - Revision A
CASE	Computer Assisted Software Engineering
CCA	Computer Corporation of America
CDIF	Common Data Interchange Format
CM	Configuration Management
DBMS	Database Management System
DDL	Data Definition Language
DES	Data Encryption Standard
DSEE	Domain Software Engineering Environment
EAST	European Advanced Software Technology
ECMA	European Computer Manufacturers Association
EFA	European Fighter Aircraft
E-R	Entity Relationship (data model)
ESF	EUREKA Software Factory
I/O	Input/Output
IPC	Interprocess Communication
IPSE	Integrated Project Support Environment
IRDS	Information Resource Dictionary System
ISEE	Integrated Software Engineering Environment
ISF	Integrated Software Factory
ITSEC	International Trusted Security Evaluation Criteria
JSD	Jackson System Development
LID	Logical Identifier
MIL	Module Interconnection Language
MIT	Massachusetts Institute of Technology
NGCR	Next Generation Computer Resources
NSE	Network Software Environment
NIST	National Institute of Standards and Technology
OID	Object Identifier
OM	Object Manager (data repository)
OMG	Object Management Group
OMS	Object Management System
O-O	Object-Oriented (data model)
OS	Operating System
PACT	PCTE Added Common Tools
PCTE	Portable Common Tool Environment
PDES	Product Data Exchange with STEP (Standard for Exchange of Product Model Data)
PHIGS	Programmer's Hierarchical Interactive Graphical System
PMDB	Project Master Data Base
PML	Process Modeling Language
PSESWG	Project Support Environment Standards Working Group
QA	Quality Assurance
RM	Reference Model

RPC	Remote Procedure Call (process communication)
SCCS	Source Code Control System
SDE	Software Development Environment
SDS	Schema Definition Set (from PCTE)
SEE	Software Engineering Environment
SLCSE	Software Life Cycle Support Environment
SQL	Structured Query Language
STL	Semantic Transfer Language
TCSEC	Trusted Computer System Evaluation Criteria
TGRM	(ECMA) Task Group on the Reference Model
UA	User Assistance
UI	User Interface
UIMS	User Interface Management System

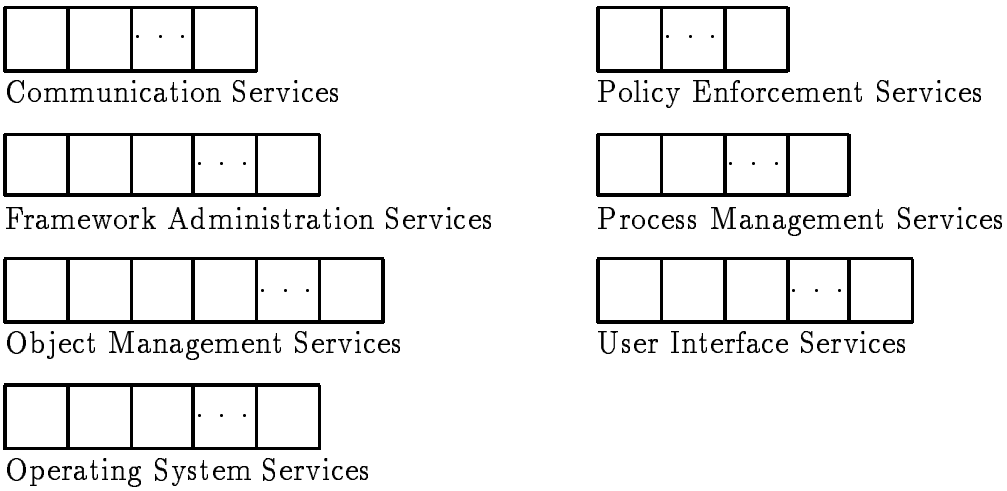


Figure 1: Service groupings

2 SEE Frameworks Reference Model

The reference model for SEE frameworks describes an environment framework as functional elements called "services." Services are partitioned where they are closely related from the point of view of their functionality. This clause summarizes these services as well as describes the concept of a service dimension, the mechanism used to give the detailed description of each service later in this report.

2.1 Frameworks

A software engineering environment provides all the software facilities for the development and support of software products. At a conceptual or modelling level, we view an environment as providing an abstract set of services in support of this development. These services provide the functionality needed to build the software and related artifacts (e.g., specifications, documentation, test cases).

In an SEE, we often separate the set of services into those services needed for application development and the set of services to provide a rich infrastructure to support that development. The latter we usually call the *framework* and is the major object of study in this reference model.

The application development services in an SEE are called *end user, engineering,* or *project life cycle* services. These provide the functionality usually associated with the construction of software. While not explicitly described by this reference model, the NGCR PSESWG (Navy's Next Generation Computer Resources Project Support Environment Standards Working Group) program has produced an end-user service-based SEE reference model [70] based upon this document's reference model.

2.2 SEE Reference Model Structure

The SEE framework reference model consists of a set of services which correspond to SEE framework functional capabilities. To ensure the clarity and manageability of the model, services have been grouped into larger "service groups." Each grouping corresponds to one characteristic of an SEE, such as the objects that are manipulated by the SEE (data), the steps performed by an SEE (processes), interactions with users (user interface), communication between components in an SEE, and administration of the SEE.

The grouping that was found convenient is represented by the section headings in Part II of this report (e.g., Object Management services, Process Management services, Communication services, and others). (See figure

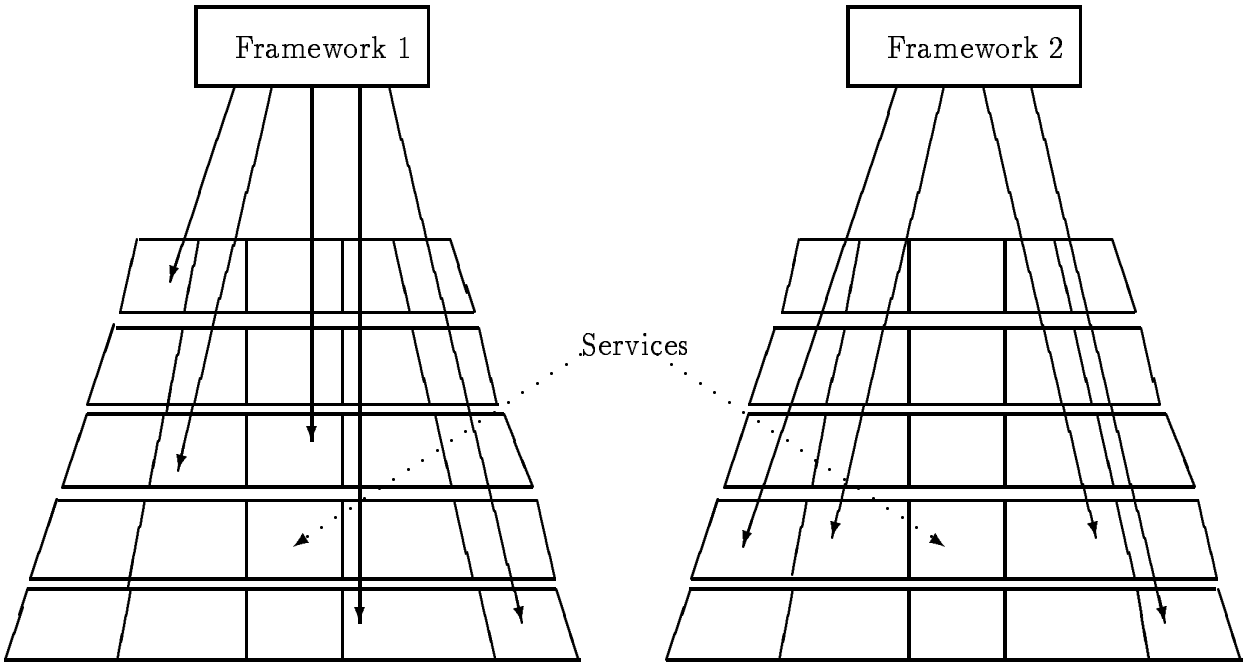


Figure 2: Using the RM to describe frameworks

1.) This particular set of groupings was chosen as it allows the RM to be viewed from perspectives that are close to user understanding of SEEs. Some of these groupings also ease the discussion of various kinds of integration: presentation integration (user interface services); control integration (process management services plus communication service); and data integration (object management services).

By describing the functionality of an SEE, SEE framework or SEE framework component using a commonly accepted nomenclature, we support some of the aims of the reference model, as given in 1.3. For example, figure 2 shows two frameworks and indicates which reference model services each provides. The grid in the figure represents the RM with each row representing one grouping and each square one service. Note that the grid is not really a square, since different groupings may have different numbers of services. By identifying the set of services each framework provides, we may be able to determine the degree to which the systems/frameworks are similar, compatible and/or redundant. It should be noted that such descriptions of framework components are more than mere "check lists" of which services each component supplies, and that true redundancy or complementarity can only be checked by further analyzing each component.

By jointly mapping multiple framework components to RM services (as in Figure 3), it is possible to investigate their combined properties. In particular, this can indicate services that are not provided by the combination and may highlight areas of overlap which may need to be resolved. Such mappings should go a long way towards increasing our understanding of the structure and future development of such framework components and may lead to the identification of further standardization areas.

The groupings of services are briefly described in this clause with complete descriptions later in this document.

2.2.1 Object Management Services

The general purpose of the object management grouping is the definition, storage, maintenance, management, and access of object entities and the relationships among them. Those services are described in more detail in clause 4.

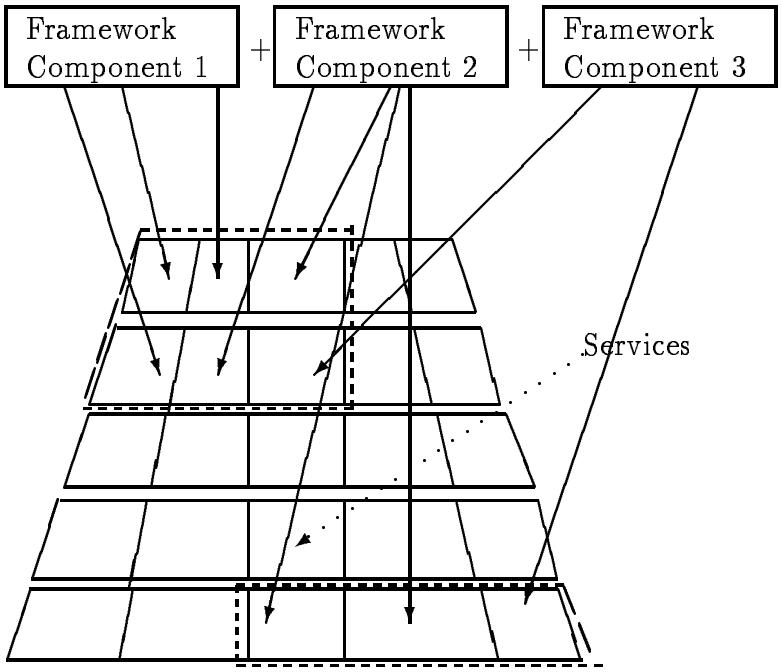


Figure 3: Mappings of multiple framework components

Metadata Service. This service provides definition, control, and maintenance of metadata (e.g., schemas), typically according to a supported data model.

Data Storage and Persistence Service. This service provides definition, control and maintenance of data typically according to previously defined schemas and type definitions. It provides the means for persistence of environment related objects, i.e., it allows objects to live beyond the process that created them.

Relationship Service. This service provides the capability for defining and maintaining relationships between objects in the object management system. It may be an intrinsic part of the data model supporting relationships either at the type level or at the instance level, or it may be a separate service.

Name Service. This service supports naming objects and associated data and maintains relationships between surrogates and names. When people communicate with a computer-based environment there has to be an agreed means of identifying objects in the environment. Computers may use unique, arbitrary identifiers, while people often need to use textual identifiers called "names."

Distribution and Location Service. This service provides capabilities that support management and access of distributed objects. Data (e.g., objects, resources, processors) and possibly services of the SEE framework may be distributed among a collection of processors and storage devices.

Data Transaction Service. This service provides the capabilities to perform a unit of work made up from a sequence of atomic operations.

Concurrency Service. This service provides capabilities that ensure reliable concurrent access (by users or processes) to the object management system.

Operating System (OS) Process Support Service. This service provides the ability to define OS processes (i.e., active objects) and access them using the same mechanisms used for objects, i.e., integration of process and object management. This is distinguished from life cycle process support which is the topic of clause 5.

Archive Service. This service supports the transfer of information to off-line media and vice-versa. The amount of data which is associated with a project in a SEE may be more than may be held online. There are requirements that once software is developed, it may have to be maintained for tens of years. It would not be feasible to keep all development data online for such periods of time.

Backup Service. This service supports the restoration of the development environment to a consistent state after any media failure.

Derivation Service. This service supports the definition and enactment of derivation rules among objects, relationships or values (e.g., computed attributes, derived objects, inherited objects).

Replication and Synchronization Service. This service provides for the explicit replication of objects in a distributed environment and the management of the consistency of redundant copies.

Access Control and Security Service. This service provides for the definition and enforcement of rules by which access to SEE objects (e.g., data, tools) may be granted to or withheld from users and tools.

Function Attachment Service. This service provides for the attachment or relation of functions or operations to object types, as well as the attachment and relation of operations to individual instances of objects. These capabilities may be provided as part of the OM data model (e.g., O-O model), supported by the data model (e.g., relationships in PCTE), or as a set of capabilities built outside of the OM.

Common Schema Service. This service provides a common schema of the objects and (possibly) process descriptions in the database in support of tool integration. This means that tools may use the Common Schema service to describe and access the data they manipulate.

Version Service. This service provides capabilities to create, access, and relate versions of objects and configurations. Change throughout development has to be managed in an SEE and the inclusion of versioning is one of the means of achieving this.

Composite Object Service. This service 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.

Query Service. This service is an extension to the data storage service's read operation. It provides capabilities to retrieve sets of objects according to defined properties and values.

State Monitoring and Triggering Service. This service supports the definition, specification and enaction of database states and state transformations and the actions to be taken should these states occur or persist.

Data Subsetting Service. This service supports the definition, access, and manipulation of a subset of the object management model (e.g., types, relationship types, operations if any) or related instances (e.g., actual objects).

Data Interchange Service. This service supports two-way translation between data repositories in different SEEs. The object management services handle objects within an SEE framework, but there is a need to be able to exchange objects among environment frameworks.

2.2.2 Process Management Services

The general purposes of the Process Management services (clause 5) in an SEE are the unambiguous definition and the computer-assisted performance of software development activities across total software life cycle. In addition to technical development activities, these potentially include management, documentation, evaluation, assessment, policy-enforcement, business control, maintenance, and other activities. The services here are:

Process Development Service. This service 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.

Process Enactment Service. This service 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.

Process Visibility Service. This service 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 representations; however, independent services may be provided to deal with interprocess interactions such as visibility of common data, common events, and propagation of information.

Process Monitoring Service. This service observes the evolving enactment state of processes, detects the occurrence of specific process events, and enacts other processes to respond to these detected events.

Process Transaction Service. This service 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.

Process Resource Service. This service 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).

2.2.3 Communication Services

These services provide for communication among the services of an SEE (clause 6).

Data Sharing Service. This service is actually a restatement of data sharing through the OMS or memory by other data manipulation services.

Interprocess Communication Service. This service provides primitive operating system process communication via the RPC mechanism of the network service.

Network Service. This service supports communication among heterogeneous collections of processors.

Message Service. This service supports communication among processes within the elements of a populated environment framework.

Event Notification Service. This service supports the notification of messages based upon certain triggering conditions.

2.2.4 Operating System Services

Since an SEE is usually realized as an extension to an existing operating system, these services describe the functionality generally ascribed to operating systems, although the realization of the service may or may not involve the native operating system. These are given in greater detail in clause 7.

Operating System Process Management Service. This service provides the ability of an operating system to create processes and to schedule each independently for execution.

Operating System Environment Service. This service provides for passing of information between operating system processes.

Operating System Synchronization Service. This service provides for appropriate synchronization of the execution among all operating system processes in an SEE.

Generalized Input and Output. This service provides basic operations to transfer data between processes and input and output devices attached to an SEE.

File Storage Service. This service provides the basic operations to read and write files and to store and access them in directories.

Asynchronous Event Service. This service provides for the creation and sending of signals between operating system processes.

Interval Timing Service. This service provides for the ability to set and test timers on individual operating system processes.

Memory Management Service. This service provides the ability to manage the main memory of the framework.

Physical Device Service. This service provides the ability to manage the physical devices attached to the framework.

Operating System Resource Management Service. This service provides the ability to allocate system resources among the various operating system processes in a framework.

2.2.5 User Interface Services

The User Interface services provide the main conduit for user involvement with the environment and they provide the major data path between tools and user. They are described in greater detail in clause 8.

User Interface Metadata Service. This service provides for defining schemata used by other user interface services. The ability to define windows, menus, icons, and reference them by name is the function of this service.

Session Service. This service provides the functionality needed to initiate and monitor a session between the user and the environment.

Text Input Service. This service manages character input for the framework.

Dialog Service. This service provides the interface between the application program and physical display devices. It is the function of this service to seamlessly pass information between programs and users.

Display Management Service. This service provides for interaction among individual windows.

Presentation Service. This service provides capabilities to create and manage the physical interface between the user and the SEE. This includes the screen display area as well as sight, sound, touch or other sensory interfaces in the SEE.

UI Security Service. This service provides for mediation between user views and actions and the security policies enforced by the framework Policy Enforcement Services.

User Interface Name and Location Service. This service permits the framework to manage multi-user and multi-platform environments. It permits various sessions to communicate among various tools and various display devices.

Internationalization Service. This service provides those capabilities necessary to support local conventions for accessing certain data (e.g., character codes, dates) within the worldwide SEE community.

User Assistance Service. This service provides a consistent feedback from various tools to the user for help and error reporting. User Assistance (UA) (or Help) is an environment-wide, cross-tool concern.

2.2.6 Policy Enforcement Services

The general purpose of the Policy Enforcement services is to provide support for confidentiality and integrity in an SEE, for both mandatory and discretionary security (clause 9).

Security Information Service. This service supports the establishment of security information for use within the SEE.

Identification and Authentication Service. This service 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.

Mandatory Access Control Service. This service provides capabilities to assign access values by a security officer to govern access to the information contained in an object.

Discretionary Access Control Service. This service provides the ability to have personal privacy.

Mandatory Integrity Service. This service provides the capabilities to protect objects from unauthorized or unconstrained modification as determined by a system's security officer.

Discretionary Integrity Service. This service provides the capabilities to protect objects from unauthorized or unconstrained modification as determined by a user.

Secure Exportation and Importation of Objects Service. This service provides the ability to export and import objects in a secure manner.

Audit Service. This service provides the ability to record information about the actions that are taken that relate to security.

2.2.7 Framework Administration Services

An SEE framework has to be carefully administered because its precise configuration may be constantly changing to meet the changing needs of the software development enterprise. These services (clause 10) provide for general framework administration:

Registration Service. This service provides a means for incorporating new tools into an environment based on the framework in such a way that different framework components coordinate effectively with the new tool.

Resource Management Service. This service provides for the management, modelling, and control of the physical resources of the environment.

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

Sub-Environment Service. This service provides the capability to divide the SEE into separate sub-environments and to restrict the access of users to only a subset of those SEE resources that are available.

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

License Management Service. This service provides for the enforcement of licensing requirements on SEE components. Software, especially on a network, is often licensed for use by a specific subset of users of the SEE.

2.3 Reference Model Dimensions

It is useful to structure descriptions of the previously mentioned services to ensure that descriptions of systems under review are compatible and comparable and are clear, precise, and comprehensive in describing what the system does and does not provide. The "dimensions" chosen are those that make distinctions clear where it is easy to blur them by using imprecise terminology. Dimensions provide the necessary structuring of service descriptions.

The term "dimensions" is used for the kinds of description the reference model emphasizes with regard to the services. This is to stress the fact that different dimensions are relatively distinct (if not orthogonal) from one another. That is, if a feature in one service was 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.

In order to provide descriptions of services from various perspectives, a set of dimensions has been identified to be associated with each service in the RM. Those dimensions are: Conceptual, Operations, Rules, Types, External, Internal, Related Services, and Examples; they are defined below.

The advantage of using dimensions to structure a service description is that it not only enables important characteristics of services to be identified and emphasized, but it also offers consistency between descriptions of different services (thus highlighting relationships among services) and possibly offers correlations between different systems described using the reference model.

Another goal of the dimensions is to provide ways to describe framework or SEE services using clear, precise and comprehensive descriptions. Note that some dimensions are relatively distinct from each other and some are related but used to describe different perspectives, e.g., external versus operations. It is hoped that the dimensions provided also help point out distinctions between systems.

The service dimensions are as follows.[1]

1. **Conceptual.** This dimension describes the semantics (i.e., functionality) of a service without reference to either how it is implemented or to the ways in which it may be made available to other services or to users. Neither a language nor a notation for making the conceptual description is offered here (although this is an ideal area for the application of abstract formalisms).

   When describing services it is often important to characterize the maturity or general acceptance of the underlying need for this service. Such information may be addressed in the conceptual dimension.

2. **Operations.** Services typically provide a set of operations that implement the functionality described by the conceptual dimension, although the explicit format of that functionality is described by the external dimension and any implementation details are given by the internal dimension.

3. **Rules.** Associated with the objects and operations of a service is a set of rules to constrain the states the objects may reach and the changes to states that operations may make. Pre- or post-conditions or restrictions on the use of objects or operations are examples of rules. In addition, some services may provide the ability for additional rules to be defined and associated with that service. Examples of this may be a service which ensures a manager manages no more than five programmers or a service that allows access controls to be re-defined.

[1] *Some of the terminology and much of the semantics is borrowed from the ANSI/SPARC 3-schema architecture [4]. Its aim is to allow separate discussion of what a service is (conceptually), how it is implemented (internally); and the ways in which it is made available (externally) to other services.*

4. **Types.** This dimension describes the possible types of objects (or data model) used by that service, information about these types (metadata), as well as the objects (instances of these types) which are used in the service. Additional type information may also be found in the model in at least three other places: objects needed by the "operations," objects needed in the "implementation," and objects provided externally to be used by other services.

   It may be useful in some cases to make a distinction between instances, types, and information about types (metadata).[2]

5. **External.** This dimension discusses how the service is made available to be used. A service may be used by other services, by tools (or application programs), or quite directly by users. For example, a query service may be provided externally by means of a procedural interface, an embedded query language, or a full tool for user support.

6. **Internal.** This dimension discusses implementation issues. In general, good design separates implementation issues from the functionality provided. Services available to tools executing within the SEE might be supplied by a specific framework implementation, by the underlying native operating system upon which the framework executes, or by other tools executing within the framework. In all of these cases, these services are considered to be supplied by the framework and the internal dimension may describe the underlying implementation.

7. **Related Services.** How one service may and does interact with another service is one of the most important areas in SEE framework development. This document provides examples of typical relationships between services; it does not dictate a set of relationships that must exist. The identification and possible standardization of relationships between services is very important due to the desire for an open, extensible, heterogeneous environment, allowing for different vendors' hardware and software components to work together effectively.

   It is also interesting to separate static and dynamic relationships between services. In this way it is not only possible to describe which services may use which others, but also to look at the sequences of interactions which may take place.

8. **Examples.** This is the place where examples of services are provided in the reference model. Examples of specific systems interfaces and languages may be provided together with the other dimensions.

It is important to understand that the reference model is not prescribing that a system being described using the reference model utilize every service, nor that every service needs to be explained from all dimensions. Some dimensions may be more important than others for specific services; in other cases, a dimension may not apply.

Section II of this report describes the services of the SEE framework RM, each clause corresponding to a service grouping. For each service, descriptions about the dimensions described above are provided. Just because a dimension is not fully described, it does not mean that it is not applicable to specific implementations or systems. Also, the service descriptions may include a note to indicate the usefulness of the service.

Each feature in a service description describes a point in the multi-dimensional space defined by the reference model dimensions, and dimensions are independent. Two frameworks may have similar services that differ only in one or two dimensions. The dimensions should be used at each level they are felt to be appropriate. For example, if the external dimension of a high-level service reveals that the service is being implemented by using a complex underlying service, it is correct to look at the underlying complex system through its own conceptual (and other) dimensions.

[2] *The terms are borrowed from the database world (the IRDS work [64] is a reasonable source), but they are used more generally here.*

3 SEE Issues

The emergence of software engineering environments is a relatively new phenomenon within the software engineering discipline, and as such, the technology is still rapidly developing. The need to standardize frameworks for such environments has been recognized with the development of standards like PCTE [36] and CAIS-A [30]. However, these do not go far enough towards developing *integrated* SEEs. CAIS-A and PCTE were designed with a major goal being that a common data repository would meet the needs to integrate various tools into a consistent SEE framework. However, more must be addressed than object management. Communication among SEE components, support for defined life cycle development processes, user interface concerns and administration of the framework are all issues that address the useability of an integrated SEE. There is no industry-wide consensus, however, on all of these issues.

While this report addresses current opinions on the set of services needed with regard to building SEE frameworks, there are still many open questions remaining to build SEEs on top of frameworks that address the needs for different application areas. This clause discusses several of these issues.

3.1 Integration

The concept that most differentiates a software engineering environment from a set of tools simply executing on a computer under some operating system is the degree of *integration* that the environment provides. By integration we mean several related ideas:

- The degree to which different tools may effectively communicate among one another within the given environment framework.
- A measure of the relationship among components of an environment.
- The ease, interoperability, portability, scalability, productivity and other "ilities" produced by the seamless interaction among a set of environment components.

Sharing a common OMS versus separate file systems for each tool is one important component of integration, but by no means the only one. A framework must provide additional capabilities allowing independent vendors to provide tools and services available for multiple purposes within the environment. While several tools provided by the same vendor may be made to interoperate, the goal for an SEE is to provide a set of interfaces that allows for arbitrary cooperation among tools from various vendors.

Related to integration is the concept of an *encapsulation* service. This service is used when a tool exists (but was not written to make use of any of the environment framework services) and is made to work in an environment framework by surrounding the tool with software that acts as a layer between the tool and the framework (e.g., encapsulation "wrappers"). This is an important idea because it allows tools to be introduced into a framework in quite a straightforward way without modifying the tools themselves. However, integration means much more than this simple level of tool execution.

3.1.1 What is Integration?

The incorporation of integration concepts into the reference model of environment frameworks requires understanding of how integration relates to the set of services described by the reference model. Integration involves all of the following:

- **It is a set of services.** Many of the services described in this document are applicable to integration. Using a common OMS with common schemata permits tools to share objects. Common presentation characteristics in the user interface allow for building common "look and feel" features across tools.

Process management services and communication services are certainly needed for tools to communicate with other tools. Function Attachment (see 4.14) and Common Schema (see 4.15) are examples of services that provide integration mechanisms.

- **It is a new dimension for each service.** Having common services allows for but does not force integration. Having common OMS and schemata permits but does not force the tool builder's use of it. For any individual environment framework, the integration dimension is the degree to which this service may contribute towards integration.
- **It is a policy.** Providing integration services to achieve integration also requires an enforcement policy so that the various platform, framework, and tool builders use the various integration services effectively. This may either be implemented as enforcement services or a "style guide" of expected practices by builders of tools to incorporate into an existing framework (e.g., PACT Tool Writer's Guide[82]).

3.1.2 Integration Mechanisms

As defined above, integration is a property of the relationship among (several) components in an SEE. This relationship may be between tool and framework, tool and user interface, tool and tool, tool and OMS, etc. There are several perspectives in describing integration:

From the user's perspective. An integrated SEE provides a common view into the system. The entire environment operates as one consistent tool rather than a collection of separately invoked or distinct functions. For example, this means common access to data or common presentation (e.g., windows, mouse, control commands, error messages, tool invocation) of the services that the user initiates.

From the tool developer's perspective. An integrated SEE provides a consistent interface for building tools. The functions needed to interact with the OMS, process management, user interface and other services should be clearly specified. Tools should be able to pass information to other tools in an easy manner.

In general, integration has been identified in several areas:

Data integration. Data integration is the ability to share information throughout the environment.

Different tools and services within the environment have their own requirements to access and share data. A high degree of data integration may mean that the tools in the SEE use a common database with a common schema (e.g., Diana). Other degrees may include using common data formats or using translation mechanisms (e.g., MILs like Polylith [71]). One aspect of data integration may include composition of data.

Control integration. Control integration is the ability to combine the functionalities offered in an environment in flexible ways. The combinations may correspond to project preferences and be driven by the underlying software processes.

Presentation integration. Presentation integration is the ability to interact with environment functionalities with similar screen appearance and similar modes of interaction.

Process integration. Process integration is the abililty to access environment functionalities based upon a pre-defined enactable development process.

Framework integration. This refers to the degree to which the tools are integrated with (i.e., make effective use of) the framework. Does the framework provide services for common mechanisms for installation, modification, and deletion of tools? Do similar mechanisms exist for establishing user authentication, security classes, and other operating characteristics?

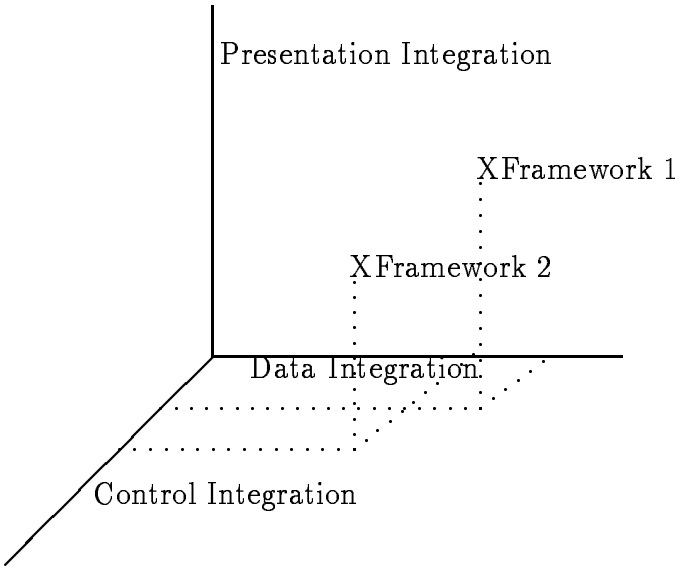


Figure 4: Integration mechanisms.

Using a figure developed by Wasserman[88], we may represent several of the integration areas as discrete points bounded by three integration axes (see figure 4). Every tool uses features to handle each integration dimension (e.g., command lines, tiled windows, Motif for presentation integration; shared files, common repository, common object base for data integration; shell scripts, triggers, messages for control integration). The goal of standardizing frameworks is to provide a small number of points to which tool vendors and environment builders may target their efforts. From the user's perspective it would enhance the presentation integration across tools, and from the developer's perspective it would enhance portability and interoperability across different environment frameworks.

While some of these aspects of integration are covered by some of the services in Section II of this report, additional study is needed before a more complete integration reference model is developed on top of this SEE framework RM.

3.2 Process Support

Early SEE framework design viewed the data repository as the central mechanism for achieving environment integration. However, it is an emerging view that the role of an environment is to support use of an *effective* software process, where *process* is defined as *a set of steps (potentially partially ordered) intended to reach a goal.* Process means software development in the context of this document.

This view is gaining support because software development is now recognized as a complex, labor-intensive, intellectual activity with high potential for quality and productivity improvements based on discipline, management, and the assistance of SEE and other computer technologies. Many organizations have trouble defining and performing the steps that transform user needs into a software product in ways that are repeatable, measurable with respect to their impact on quality goals, and adaptable or improvable. Hence, any SEE support for installing a defined process into a development project's setting could provide substantial near-term benefit. The goal of the services described in the process management clause (5) is to contribute individually or collectively to this effective support of software processes by providing end-user-oriented facilities for defining and using processes that could replace undisciplined, difficult-to-control, or tedious invocation of individual tools.

These services could also allow for architecting an increase of "intelligence" within SEE frameworks that increases automation in the sense that process-centered SEEs could be designed more easily and effectively – which would be an instance of *process improvement*.

The term *process* is generally used hierarchically in the sense that *complete processes* (meaning life cycle-wide completeness) may be decomposed into processes which span activities less than an entire life cycle. While it is proper to use *process* to also refer to these decomposed parts of complete life cycle processes, the terms *process element*, *process fragment*, *process step*, and *subprocess* are often used to clarify usage from life cycle completeness. Examples of process elements include configuration management processes, design processes, change processes, review processes, emergency processes, and maintenance processes. A design process may be an example of a non-atomic process element which contains a smaller process element, for example a review process (which might be an atomic process step, meaning it has no externally visible sub-structure from a process perspective). Other terms sometimes used synonymously with this meaning of *process* are *task* and *activity*, both of which are also overloaded words; also, a *project* may be regarded as an instance of a process that is being enacted (executed). The term *process* is used consistently in this document instead of these synonyms, and process-related terminology is used as defined in [38]. This usage of *process* should not be confused with usages of the word in different contexts (e.g., *operating system processes*).

SEEs that provide substantial process services are sometimes referred to as "process-centered," "process-managed," or "process-oriented" SEEs. The term *process engineering* is used to describe activities of a *process engineer* – one who specifies an organization's or a project's life cycle process or process elements and, in particular, the interaction between process users (project members) and an SEE. The terms *process programming* [66] and *process modeling* refer to potential activities within process engineering, which may also include tailoring, adapting, validating, tracking, measuring, analyzing, and improving (evolving) processes.

The process management services described in this reference model are based on an emerging view that software processes may enjoy life cycles very analogous to software life cycles:

- processes may be specified, designed, implemented, enacted (executed), analyzed, measured, evolved, and improved;
- processes may have *process architectures* which are conceptual frameworks for incorporating, relating, and tailoring process elements in consistent ways, including the ability to indicate whether a process element is or is not compatible with the architecture; process architectures may specify interfaces between (sub)processes, guidelines for composition, and communication mechanisms between process elements; process architectures may exhibit properties specific to application domains or specific to project objectives such as reuse support, prototyping, evolutionary development, or incremental delivery;
- *process assets* such as complete processes, process elements, and process models, process architectures, or process designs may be organized into libraries and be reused.

A SEE could provide services for process development life cycles and software development life cycles, or it could provide only services for one or the other. Typically, SEEs have aspects of both.

Some SEEs provide some process management services (e.g., process development) by the same facilities that fulfill other services in this reference model. Object management facilities are prime candidates for this duality because process developments are complexes of "information," and because processes might also operate on project information (as well as process state representations) during enactment in manners so consistent with SEE object management activities that one unified set of object management facilities could also provide certain process management services. The dimension "Related Services" is used in the Process Management clause to note this potential for an SEE's facilities to provide process services simultaneously with other services.

The field of process management is relatively new compared to other aspects of SEE technology such as object management and tools. For example, the point is made in clause 5 that most services in 5.3 and beyond might be construed as extensions of process definition (5.1) or aspects of process enactment (5.2), but they are

treated independently in this reference model because of the field's immaturity and the likelihood that many current SEEs may not deal with all these services in an integrated manner. As the subject of *process* and its impact on SEEs becomes better understood, the reference model will be improved to better accommodate description of these services and their potential overlap with other service categories.

Section II. Framework Service Descriptions

4 Object Management Services

The general purpose of the objects and object management service grouping is the definition, storage, maintenance, management, and access of data entities or objects and the relationships among them. Object managers manage "things" (i.e., data or objects, and possibly processes) which support the activities of the life cycle. In this document we call these things "objects." The following clauses describe each of the services in more detail.

4.1 Metadata Service

The Metadata service provides definition, control and maintenance of metadata (e.g., schemas), typically according to a supported data model.

4.1.1 Conceptual

Metadata is data about the structure and constraints of data and objects in the object manager. Its services may be specific or general in support of the many existing data models currently in use (e.g., E-R, relational, Object Oriented (O-O)). The study of data modeling in general, and its application to software engineering environment frameworks in particular, has a substantial background of research and development. Some books and surveys of the field include [10] [24] [86] [68] [48]. Brown[11] specifically addresses database support for software engineering.

The Metadata service is used to define schemas (i.e., specific types, attributes, or relationships) to support access to objects held in the object manager. The Metadata service may be used to develop, for example, data dictionaries, schema definition languages, catalogs, and class hierarchies. Metadata may be manipulated by the object management services. As such, we may apply other services to metadata, such as versioning and archiving of metadata.

The Metadata service provides control and maintenance of metadata. The study of metadata and its maintenance is ongoing [78]. Updating types and other forms of data model and schema evolution is a difficult problem. For example, what happens to existing instances of an altered entity type? As new data models incorporating complex objects and versions are developed, so metadata research develops. Depending on the data model adopted, creating new types may be done similarly or in a different form than creating objects (of certain types). It is for this reason that metadata updating is typically done using different functions from those provided by the Data Storage service. For example, instead of creating a new type "attribute," there may be a "create-attribute" function. Object-oriented database management systems (e.g., IRIS [40]), however, tend to be more uniform in treating both metadata and data storage operations.

4.1.2 Operations

Typical operations of the Metadata service are: create, update, and query schema information in the supported data model. Other operations which may be available may deal with changing a type by adding or removing attributes.

A Metadata service offers the opportunity for generic tools to be written which operate according to the structure of the objects in a particular environment, rather than relying on particular data structures to exist (apart from the metadata structures).

There are other data models which support a more dynamic form of operation at run time (see the Function Attachment service (4.14)). These are polymorphic systems, and they are one of the fundamental distinguishing characteristics of object-oriented systems.

4.1.3 Rules

The Metadata service creates the data model that allows the definition of rules to be associated with the various objects (e.g., constraints associated with domains in relational models, rules to be associated with a type, in object-oriented systems). For this purpose, it may provide its own rule definition language.

4.1.4 Types

A data model may or may not have the notion of "type" (or class). Example types include "relationship types," "object types," "attribute types," and "value types." The existence of a type does not necessarily depend on the existence of instances of that type. Instances of the same type have the same set of attributes. Examples of types include software, documents, problem reports, and a person. Examples of instances of types are Preliminary Design Document for System A, and System A.

Types may be related to one another (see 4.3). One way this is often done is by a "subtype hierarchy" in which types are related by a subtyping relationship. The exact form of this relationship may be defined in a variety of ways, but all are related to the basic form which states that all objects (or instances) of a given type T1, which is a subtype of type T2, have all the attributes which objects of type T2 have. A term used in this context is "attribute inheritance."

Typing is not the only useful grouping of objects. Another way of grouping objects is by aggregate types such as sets.

4.1.5 External

The schema definition capabilities may be presented in a different way from the object definition capabilities or in the same way. The latter case is called a self-referential mode, i.e., uses its own data model to define the schemas (e.g., relational and object-oriented models). In this case, the external capabilities may be provided in the same way as the "data storage" service external dimension. The benefits of this are that the user or application has fewer concepts to deal with, a consistent interface (e.g., the Query service may now return information about objects and metadata), and a place for general unification of ideas.

Examples of forms presented for external use are: Data-Definition Languages (DDL), procedural interfaces, and type definitions in persistent programming languages.

4.1.6 Internal

Typically a Metadata service uses the file system to store information about schemas; however, in other cases it may use other object management services (such as the Data Storage) to store metadata.

An issue with self-referential systems is that they may constrain the methods of implementation which may lead to some rather tricky problems (e.g., what are the implications of creating a new version of an entity type?).

4.1.7 Related Services

The services related to the Metadata service are many of the object management services including the Data Subsetting service (4.20). In particular, since the structure of the objects in an environment evolves, versions

of metadata may be present. Metadata may also interact with Mandatory and Discretionary Access Control services (9.3 and 9.4).

The use of DDL for data exchange relates to the Data Interchange service.

Note that data about other services (e.g., processes, rules, or tools) is not necessarily metadata.

4.1.8 Examples

Examples of data models from real systems include the node model of PCTE [85] [36] and CAIS-A [30]; the E-R model of Aspect [44] [32] [12]; the class hierarchy of ATIS [26]; and elements of the following models and systems: Damokles [28], the relational model [19], Taxis [63], SDM(+) [77], and Iris [40].

Examples of Metadata services are: data dictionaries, schema definition languages, catalogs, and class hierarchies.

4.2 Data Storage and Persistence Service

The Data Storage service provides definition, control and maintenance of data typically according to previously defined schemas and type definitions. It provides the means for persistence of environment related objects, i.e., it allows objects to live beyond the process that created them.

4.2.1 Conceptual

The Data Storage service provides a means for the creation and storage of objects or references to these objects. Creation brings into existence an object with an identity distinct from all other objects. The storage aspect of the service means that the object lives beyond the lifetime of the process that created it and may be accessed until its deletion.

Generally, it makes sense to distinguish between two levels of granularity of objects:

- coarse-grained data, corresponding to major sections of documents, e.g., source programs, diagrams, figures, chapters of a report; and
- fine-grained data corresponding to single single syntactic items, e.g., tokens or variables in a source program, lines or symbols in a diagram.

While the basic data modelling facilities can be the same for both levels, it is often difficult to achieve satisfactory performance in a single OMS for both levels at the same time.

4.2.2 Operations

The typical operations for storing objects are create, read, update, and delete. An important point about update is that it typically may not be simulated by a sequence of create and delete operations. This is because of the unique existence of objects. The update operation may change values of attributes. Update and delete may be replaced by a "new-version" operation (see 4.16).

Some operations may be subject to the basic constraints of the object model definition. Triggering operations (see 4.19) may be executed as a consequence of data changes or the execution of data storage operations.

Data administration operations may also be associated with this service. These are operations to map the data model into internal structures such as lower-level data storage concepts (e.g., indexing and garbage collection).

4.2.3 Rules

This service may allow rules or properties to be associated with the operations (e.g., pre- or post-conditions) for creating or modifying objects.

4.2.4 Types

The Data Storage service may provide information (data) about itself such that the data repository is self-defining. There may be metadata for handling internal house cleaning, such as maintaining efficient use of physical storage devices. There may be metadata for navigating through the database, i.e., relationships. There may be metadata for state descriptions such as when a state changed, when the last back-up was done and where it is located, whether objects are on-line or off-line. There may be metadata for users, services, and tools to access for obtaining a description of their data, such as size of the object, graphical route (path) to where the object is located, last modified date, etc.

4.2.5 External

The capabilities for accessing and managing objects may be provided by means of procedural interfaces or query languages.

4.2.6 Internal

Implementation strategies of the Data Storage service may range across the ways traditional database management systems are implemented, using operating system services completely, or skipping parts of the operating system to get direct access to hardware such as disk storage. There may be other implementation strategies adopted which are more suitable for certain kinds of systems, e.g., object-oriented database management systems. It may be that the service interface allows applications to set some implementation-related parameters (e.g., store an entity near its type definition). A term used in this context is "object clustering." Also, the actual repository may be distributed throughout the environment, yet appear local to the user.

The Data Storage service may be dynamically extensible, permitting expansion in physical size or functionality without disruption of the work of users.

4.2.7 Related Services

Examples of services that may be related to this service are: the Version service, which also affects the control, maintenance, and persistence of objects in the OMS, and State Monitoring and Triggering, which may cause actions as a result of data or state changes.

The Data Storage service is directly related to most services described in the reference model, especially the object management ones.

4.2.8 Examples

Many modern frameworks provide this service. A few examples are the object or entity management components of PCTE [36], CAIS-A [30] and SLCSE [80]. ECLIPSE [55] is a framework providing both coarse and fine-grain data objects based upon PCTE.

4.3 Relationship Service

A Relationship service provides the capability for defining and maintaining relationships between objects in the object management system. It may be an intrinsic part of the data model supporting relationships either at the type level or at the instance level, or it may be a separate service.

The objects in an SEE do not exist in isolation from other objects. Life-cycle schemas like the PMDB [69] and SLCSE [80] have suggested that there exist many relationships among life cycle objects.

4.3.1 Conceptual

The Relationship service allows the definition and maintenance of "relationships" (types and instances) among objects and object types. A simple form of a relationship may be a single (directional) "link" between two objects. Relationships may involve more than two objects. Relationships may be intrinsic to the data model, e.g., E-R model, or they may be supported by other models, i.e., relationships may be a new type in O-O models.

A relationship may be a type in its own right. If that is the case, then the same set of terminology, rules, and operations applied to types and objects may be applied to them.

Here are typical candidates for relationships: (a) There is a particular dependency between types of objects, e.g., object code is compiled from source code; (b) A recorded relationship exists between two or more objects, e.g., members of a project or author of a document; or (c) A constructional relationship is useful to record, e.g., parts of a design or test cases that test certain code.

Relationships may be used as the basis for providing derivation services.

4.3.2 Operations

If relationships are an intrinsic part of the data model, the relationship operations may be a subset of the metadata operations. Typical relationship operations may be: create, update, and delete relationship types and instances.

When relationships are themselves first-class objects, then operations across multiple sets of data may be supported, e.g., "remove all tuples involving Jim" or "assign all maintainers of M_1 as maintainers of M_2."

There may be navigation operations which allow access to objects via the relationships.

4.3.3 Rules

Rules or constraints may be associated with relationship types. An example is a rule preserving the referential integrity property.

4.3.4 Types

As implied by 4.3.1, relationships are often similar to types in the data model; in this case, relationships may be typed and have attributes. But the relationship concept may add further constraints such as "cardinality" (i.e., the number of objects allowed to participate in a relationship). Different kinds of relationships may be defined which offer differing semantics to the delete operation (and add constraints to the create and update operations). That is, a relationship may not exist unless the involved objects exist. This offers alternatives of rejecting a delete request or "cascading" the delete to delete other objects in a relationship if one object is deleted.

4.3.5 External

This dimension may be provided together with the external dimension of the Metadata or Data Storage services if relationships are part of the data model or defined as part of the schema. Or it may be provided as a service (e.g., new procedures) beyond these.

4.3.6 Internal

This dimension may be provided together with the internal dimension of the Metadata or Data Storage services. It may have its own implementation.

4.3.7 Related Services

The Relationship service is generally considered to be an intrinsic part of a data model. It may be considered as being well understood and is treated separately here since it may be provided without a complex Data Storage service (e.g., ATIS [59]).

Relationships offer a way to find objects by navigation without the need of a query language (see 4.18). Foreign keys play the role of links in the relational model, and links are not explicit in some object-oriented systems. It may also be used as a way to make composite objects (see 4.17).

Relationships may also be the (partial) basis for carving out part of an object base for interchange.

4.3.8 Examples

The entity relationship model [16] treats objects and relationships separately while RM/T [20] treats relationships as objects.

4.4 Name Service

The Name service supports naming objects and associated objects and maintains relationships between surrogates and names.

When people communicate with a computer-based environment there has to be an agreed means of identifying objects in the environment. Computers may use unique, arbitrary identifiers, while people often need to use textual identifiers called "names."

4.4.1 Conceptual

A "name" may be a string of characters associated with an object. The Name service maintains the relationships between surrogates and names. An object is allowed to have more than one name.

The term "surrogate," introduced by Hall, Owlett, and Todd [45], captures the idea that every object of the outside world is associated with a surrogate which stands for that object in the model. In other words, a surrogate is a unique, system-generated identifier which is never re-used. Other terms used in this context are "Object Identifier" (OID) or "Logical Identifier" (LID).

Surrogates allow the system itself to have control of surrogate values, which are defined over a single domain, and by allocating surrogates when the object is introduced to the system. It is therefore always true that two objects are the same if and only if their surrogate values are identical. Surrogates are never re-used to ensure that events such as restoring objects from backup storage (see 4.10) or the Archive service (see 4.9) cause no conflicts.

The Name service may also support the additional concept of a "namespace." Names may belong to specific domains (e.g., the set user names, the set of external file names). Any surrogate may then be associated with a name which is unique within the context of a namespace.

4.4.2 Operations

Typical operations are: translate a name to a surrogate, translate a surrogate to a name, and give the name of an object.

4.4.3 Rules

There may exist many rules (e.g., constraints) which apply to the particular naming conventions of a system.

4.4.4 Types

The type hierarchy, if any, may be used to enforce a naming policy. For example, nameable objects may be distinguished from those which may not be named or those which have to be named.

Names and namespaces may also be defined as types.

4.4.5 External

This dimension deals with the way users view and use surrogates. The surrogate concept does not necessarily imply that surrogates should be either totally visible nor totally invisible to the external user (e.g., project user). There may usefully be degrees of surrogate visibility:

- All objects are named, and whenever a project user would see a surrogate value, a name is presented instead;
- Project users are aware of an attribute recording surrogate values which may be taken advantage of in queries, but actual values are always hidden;
- Surrogates are completely visible and may be accessed by the project user, but the user may not, by definition, update, or in any way control, the values presented.

The first option is generally the most useful from a project perspective. However the third may be useful to the SEE framework developers or tool-writers while new elements are being developed or debugged. And the second is appropriate more especially for tools which do not want to have to generate names where it would be more appropriate to identify an object by its properties. An example might be the symbol table generated in a compilation. The use of surrogates for engineering databases in general is investigated by Meier and Lorie [62].

4.4.6 Internal

The translation of a name to a surrogate often requires a table lookup, hash table search, character transformation or a navigation among a complex data structure in the OMS. Since this may be a relatively complex operation and since name–surrogate translations may occur frequently in a framework, care must be taken that the implementation of this service does not become a system bottleneck.

4.4.7 Related Services

Name objects are created by the Data Storage service (see 4.2). The Name service does not necessarily provide access to objects (and is thus not intimately related to the Location service although it may use the Location service to locate part of its mapping). That is the purpose of the Query service (see 4.18). Where the name of an object is clearly distinguished from the means of accessing it, object independence is not compromised and it is clear whether or not two different names refer to the same object. But the two services may be very closely related so that a Name service reflects the access path to objects (i.e., a navigational approach). The PCTE approach is an example of this [36].

In general, the Name service may be used by all the other services which present an external interface to the user. Otherwise, objects may be referenced by the services as surrogates whenever object identity rather than identification is important.

4.4.8 Examples

The naming capabilities of PCTE [36], CAIS-A [30] and most modern operating systems are typical examples.

4.5 Distribution and Location Service

This service provides capabilities which support management and access of distributed objects and metadata. For example, the Location service may provide a logical and a physical model of OMS components and the means of maintaining the mapping between the logical and physical models.

Distributed software development support is firmly established as a requirement for SEE frameworks. Data (e.g., objects, resources, processors) and possibly services of the SEE framework may be distributed among a collection of processors and storage devices.

There are many aspects of distribution which SEE frameworks share in common with other subject areas also studying the problems of distribution, (e.g., office automation, distributed database management systems, and control automation systems). The necessarily brief coverage here of distribution disguises the very large topic it is in the computing world in general.

4.5.1 Conceptual

The Location service solves the problem stated above by maintaining a model of the physical components on which the environment framework services run and communicate (e.g., devices, channels, volumes, or peripherals), their attributes (e.g., live or connected), and relationships between them (e.g., the peripherals attached to a device). In addition to the physical model, the Location service may maintain a logical model which abstracts some of the physical details so that, for example, collections of devices may be considered at one location. The Location service maintains a mapping between the logical and physical models. Within the logical model there is the concept of a "Unit of Distribution." This defines the smallest unit of information upon which location operations may be invoked. The unit of distribution may be a whole, a set of, or part of, an object or service.

It is important to be clear that the Location service does not address all distribution issues. For instance, it does not deal with the ways in which a piece of work is distributed between individuals (see 4.20) nor what protocols exist for communication between services and tools (see clause 6).

4.5.2 Operations

In addition to the usual add and delete operations on elements of the distribution model, the Location service may provide move, copy, attach, detach, and replicate operations. Attach and detach deal with (dis)connecting elements (from)to the network. Replicates are copies held on more than one machine, for example, to safeguard against system failure or to provide guarantees about accessibility.

There may be special query operations to discover the contents of the distribution models. Special care may be taken concerning error handling. The failure of an operation, for example, because of network failure, may leave a wide variety of choice for actions to be taken.

4.5.3 Rules

Rules for this service may deal with such things as the unavailability of some components in the distributed system.

4.5.4 Types

Information about the distribution of objects and processes in an environment framework may itself be distributed and maintained by the framework's Data Storage service. If this is the case, it has to be carefully ensured that recursive operations and queries terminate.

4.5.5 External

The choice between transparent distribution versus resource management is an important one for environment framework designers. In the former case, the Location service is hidden from view because it is only used by other services. In the other case, location operations may be made available for tools and users to make greater use of the networked resources according to particular applications.

4.5.6 Internal

Implementation may make use of a network. Even for transparent distributions, it may be necessary to make use of such techniques as replication. Another implementation concern is how to accomodate the ability of a resource that holds part of the repository to go off-line and return.

4.5.7 Related Services

The reason for choosing a particular unit of distribution may lie in relationships the Location service has with other services in the framework. If this is the case, then the logical model may explicitly include these related concepts.

The Name service (see 4.4) may well be closely related to the Location service as the naming of distributed objects contains many problems. The Query service, Data Storage service, and Relationship service are also particularly closely related to the Location service.

Clearly, the OS Process Support service (4.8) is related to the Location service when it comes to the distribution of processes over a network. The Communication service (clause 6) may use the Location service extensively. For handling distributed object management systems, the Replication and Synchronization service needs to ensure integrity of the OMS under network failure.

4.5.8 Examples

CAIS-A [30] and PCTE [36] are examples of systems that accomodate both transparent and explicit OMS distribution.

4.6 Data Transaction Service

This service provides capabilities to define and enact transactions.

4.6.1 Conceptual

A "transaction" is a unit of work and a unit of recovery made up from a sequence of atomic operations (and transactions, if nested transactions are allowed). The transaction either succeeds in carrying out all the specified operations or restores the object upon which it was acting to a state as though the transaction never happened. Note that some operations may be irreversible. For example, once messages are sent they may not be "unsent"; other examples are "print" commands. In general, any external communication breaks atomicity.

Examples of transactions in an SEE are adding a new programmer to a project team or having a complete set of user manuals processed by a document processor. In both these cases many simple operations have to succeed to achieve the final aim. If any fail, the SEE should not be left in a state with a half processed manual or a half-employed programmer. The Data Transaction service is not intended to provide the support for process transactions, where failure does not necessarily imply rollback of updates.

A system failure has to be coped with by this service. In this case recovery is more complex, and it is very useful to have a "checkpoint" procedure which is designed to support recovery in the event of a system failure. This does not cover the possibility of online media failure. Such a case has to be dealt with by the Backup service (see 4.10).

4.6.2 Operations

Typical operations include begin and end transaction. Planned successful termination of a transaction may be achieved by the commit operation, while unsuccessful termination may use the rollback operation. Shadowing is another technique which may be employed. If the transaction is terminated before commit or rollback, e.g., by a memory violation or arithmetic overflow, the transaction has to have rollback invoked for it automatically.

There may be operations to show what state transactions are in; these operations may also be part of the query capability.

4.6.3 Rules

Transactions are subject to rules governing rollback and recovery. Nested transactions require rules regarding scoping and visibility of objects and levels of nesting (if any) involved in the transaction during its lifetime.

4.6.4 Types

The main types of concern to transactions are the objects that are involved in a given transaction. They are also concerned with representations of the state of the OMS to support rollback.

4.6.5 External

It is considered beneficial that users and applications of this service are unaware of the complexity of recovery.

4.6.6 Internal

The ways of implementing the Data Transaction service may be many and varied. The traditional database community have developed algorithms and techniques which serve their applications well and may be suitable for adoption with or without modification within SEE frameworks. It should be noted that a distributed system may have significant effect on what may be considered workable solutions. The problem is well-understood for some applications, but that does not imply that it is understood as well for distributed SEE frameworks.

A Data Transaction service in a multi-process environment may likely utilize "locks" (possibly provided by the Concurrency service) to prohibit concurrent update and to isolate a fixed state of relevant objects that are read during a transaction.

4.6.7 Related Services

The services particularly related to the Data Transaction service are the Concurrency service (4.7), the Data Subsetting service (4.20), and the Process State service (5.1).

The relationship with process transactions is one of the most interesting. The sort of service described here is not suitable for the kinds of long, nested transactions which are found in an SEE. It may be the case that environment frameworks which support process transactions across subsets do not need many of the services described here. This topic is very much a research issue.

4.6.8 Examples

Marvel [7], CAIS-A [30], and PCTE [36] provide data transactions, as do many modern data management systems.

4.7 Concurrency Service

This service provides capabilities which ensure reliable concurrent access (by users or processes) to the object management system.

4.7.1 Conceptual

The Concurrency service addresses concurrent access by users of processes. The description of the Data Transaction service is independent of whether the system is being used by a single user or process or more than one. The Data Transaction service is required to cope with the simple fact that things may not be guaranteed to work all the time, independent of the fact that resources may be shared by more than one transaction and transactions may happen concurrently.

It is important to note that the Concurrency service may be omitted from a framework for one of two reasons. First, an environment may be single-user and not allow concurrent processes; the second is where an environment is partitioned into sub-environments (see Sub-Environment service 10.4) and single-role transactions take place only within the context of a sub-environment. But even in this case, system metadata (see Metadata service 4.1) has to be protected during concurrent access and update.

A traditional example, the lost update, should serve to explain the concepts involved. Suppose a designer starts a transaction to refine a design. Suppose another designer also decides to change the same design. Both read the same original design and then commit their respective changes. Unfortunately, the designer who commits some changes first will have these changes overwritten by the second designer's commit. One of the updates is lost.

The purpose of the Concurrency service is to prevent this situation and problems akin to it. One way of describing the service is to say that it "serializes" transactions. That is, it ensures that the effect of a set of concurrent transactions is as if the transactions were run in an arbitrary sequence. Clearly, the actual sequence has an impact on the final result. However, it is not the purpose of the Concurrency service to decide a particular order for "concurrent" operations for which it is providing support. If order is so important, it is controlled by another service (e.g., process management or a tool).

4.7.2 Operations

Examples of operations are acquire and release locks. The semantics of the acquire lock and release lock operations depend on the type of locks, of which there may be several in use in one system. The detailed constraints on the get and release lock operations depend to a great extent on the type of locks defined.

There may be operations to show what locks are held on which objects. These operations may be available as special cases of the general query facility if this information about transactions and locks is held in the SEE framework's (conceptually) central database.

4.7.3 Rules

Examples of rules that may hold for certain types of locks:

- a lock cannot be acquired on a locked object; and
- a lock cannot be released until it is acquired.

4.7.4 Types

The objects associated with this service are transactions and the locks on objects. These determine which process has access to specific object management objects.

4.7.5 External

The degree of visibility of the Concurrency service varies according to the technique actually used. The operations a user of this service may see are acquire lock and release lock and perhaps restart transaction if a deadlock occurs (i.e., multiple transactions waiting to acquire locks that others who are waiting already hold).

On the other hand, it may be considered that a desirable feature is to have locks managed transparently to the tool or user so that the right locks are acquired depending on what operations have been invoked.

It may be difficult to automatically restart a transaction if communication has occurred with resources outside the control of the transaction.

4.7.6 Internal

A traditional approach to achieving serialized transactions is to "lock" objects or their attributes. A lock puts constraints on operations which may be carried out on the locked item until the lock is released. There

are many techniques for providing concurrency control with locks and for avoiding or solving the problems which locks themselves introduce. The techniques have significantly varying degrees of impact on performance, depending on factors such as patterns of object usage, granularity, and what is locked. Applying good or bad methods may have serious impacts on the degree of concurrency of transactions and the overall performance of the system.

4.7.7 Related Services

There are services related to the Concurrency service. Clearly, support for concurrent processes is essential. This may be provided by the OS Process Support (see 4.8) service. The Data Transaction service (see 4.6) is obviously closely related and the Process Enactment service (see 5.2) may use or simplify this service extensively. The Replication and Synchronization service relies on the Concurrency service to assist in maintaining the integrity of a distributed OMS.

4.7.8 Examples

CAIS-A [30] provides locks in support of concurrency.

4.8 OS Process Support Service

This service provides the ability to define OS processes (i.e., active objects) and access them using the same mechanisms used for objects, i.e., integration of process and object management. This is distinguished from life cycle process support which is the topic of clause 5.

4.8.1 Conceptual

This service is **not** trying to capture OS processes per se; it is designed to capture systems where OS process abstractions are provided together with object management systems. The service provides the basic support mechanisms for enacting and controlling active objects.

The term "static context" is used for a program in a static form that may be run by a process, either through direct execution or indirectly through an interpreter. Useful information which may be stored about a static context includes the type of locations at which it may run and whether or not it is interpreted.

Useful dynamic information that may be maintained about a process which is executing is, for example, relationships to other processes (e.g., parent processes or the process' interpreter); relationships to concepts in other framework services such as associated schema, processes, or execution site; and priority and status of the process (e.g., ready, waiting, running, terminated).

The OS Process Support service provides mechanisms to access and monitor abstractions of executing processes and to handle the properties of static contexts.

4.8.2 Operations

Typical operations of the process support service include: create, start (asynchronously or synchronously), terminate, suspend, and resume. There is also a set of operations to transfer, control, and monitor foreign processes.

Both static and dynamic information about processes may be managed by the Data Storage and Relationship services. If this is not the case, then a special set of query operations may be made available to allow access to information about processes.

4.8.3 Rules

Rules for OS Process Support may cover how dynamically the OMS reflects changes in the state of the executing OS process. Such OS process objects are also subject to all of the rules regarding the OMS, such as typing and access constraints.

4.8.4 Types

The data model may provide "process" as an object type, or it may provide the capability to identify different types of processes.

4.8.5 External

The external interface is similar to the interface provided for other objects with the possible addition of extra procedures dealing with process specific information.

4.8.6 Internal

Implementations must be concerned with maintaining the correlation between the executing OS process and its representation in the OMS.

4.8.7 Related Services

Access to information (static and dynamic) about processes may be provided by the Data Storage or Query services.

4.8.8 Examples

Process nodes in CAIS-A [30] and PCTE [36] are examples of this.

4.9 Archive Service

This service supports the transfer of information to an off-line medium and vice-versa.

The amount of data which is associated with a project in an SEE may be more than may be held online. There are requirements that once software is developed, it may have to be maintained for tens of years. It would not be feasible to keep all development data online for such periods of time.

4.9.1 Conceptual

The Archive service carries out a mapping between the online storage and offline storage of objects. A placeholder may represent the object in online storage, while the object is archived offline.

4.9.2 Operations

The basic set of operations deals with: storage of sets of objects; the identification of disc volumes after movement; and the restoration procedures.

4.9.3 Rules

Integrity of both online and offline information both at archive time and later at restoration time is a major issue.

4.9.4 Types

The information about what is archived is vital, otherwise the archived data might be lost. Also, the relationships among the archived objects need to be kept to be used for efficient navigation. Information about each archiving instance is important so that the user knows when the archive was done, by whom, what was archived, where it may be found, and where it came from. The metadata may be kept with each archive instance and collectively in the master archive directory on-line.

4.9.5 External

Standard external formats for the archived information are important. Control over what is archived and when ranges from being carried out completely automatically by the environment framework to being completely under the control of users.

4.9.6 Internal

To a large extent, OMS archiving may be implemented using archiving capabilities available in systems today.

4.9.7 Related Services

Services related to the Archive service are the Data Transaction service, the Backup service, and, to some extent, many of the other object management services. In particular, the Version service could offer a pointer to the most convenient units to archive. Also, the Query service (see 4.18) has an interest in how archived objects are marked and may be retrieved.

4.9.8 Examples

Examples of archiving capabilities are found in most modern operating systems. ARCHIVE-SAVE and ARCHIVE-RESTORE operations in PCTE provide this service [36].

4.10 Backup Service

The purpose of the Backup service is to restore the development environment to a consistent state after media failure.

4.10.1 Conceptual

There is often a need to recover a system after a system failure or a user error. The Backup service restores the environment to a prior state. The Backup service has much in common with the Data Transaction service in that it keeps copies of database states and may use the "transaction log" to redo transactions which happened since the last dump was taken. There may be logs in the dump itself.

4.10.2 Operations

There are likely to be explicit operations for causing the dumps to be made. Backups may be made by location (e.g., all objects in some directory), by time (e.g., all objects created since a certain time), by role (e.g., all objects of a certain user), by triggering action (e.g., all objects modified since yesterday) or by other events. Recovery operations may have similar attributes. During recovery, operations may be available for determining what recovery actions are necessary and for taking those actions.

4.10.3 Rules

Rules may govern how frequently backup dumps are made and what information is to be dumped. A rule might, for example, stipulate that only items that are new or have been changed since the last dump will be included in a subsequent dump.

4.10.4 Types

The Backup service may work more effectively if the concepts in environment framework services and tools have representations as objects. If this is the case, then backup strategies only have to be developed for objects, rather than a strategy for each concept.

4.10.5 External

The Backup service is likely to be visible only to the environment administrator.

4.10.6 Internal

The most important aspect of the Backup service is the time the dumps are taken. In general, they are taken immediately after any significant input or reorganization of data. Incremental dumping, in which only data items which have changed since the last dump are dumped, is another tactic which may be employed.

4.10.7 Related Services

Services related to the Backup service are the Data Transaction service, the Archive service, and, to some extent, many of the object management services.

The State Monitoring and Triggering Service (4.21) and Process State service (5.1) may help to identify the appropriate times to take dumps.

There may be particular difficulties in taking dumps of physically distributed data. The coordinated use of the Location service is important in this case (see 4.5).

4.10.8 Examples

All modern systems provide backup services.

4.11 Derivation Service

The Derivation service supports definition and enactment of derivation rules among objects, relationships or values (e.g., computed attributes, derived objects, inherited objects).

4.11.1 Conceptual

An environment may include objects that are derived from other objects; for example, object code is generated from source code by some compilation process. If a system provides ways for specifying and enacting such relationships, these should be considered derivation services. Another example is computed attributes whose values are derived from other values in the system and may be computed either when the original value is modified or when the computed value is accessed.

The means of specifying derivation may be expressed in many different ways. OMS objects in different classes or types may be related together (e.g., the executable is derived from the source by a version of the compiler). If the OMS native relationships are so used, the instance objects form a dependency graph, presumably acyclic. This form of dependency graph may then be browsed graphically or textually. In general, derived relationships may not be directly modified, but must be determined by the derivation mechanism. In some cases, semantics may be assigned to direct update of derived relationships. If the dependency relationships are not stored using relationships native to the OMS, they may be stored in declarative or imperative form. An example of declarative form is the rules expressed in a Makefile [39] by Unix. An example of imperative form is the use of a shell script to explicitly represent the order of derivation activities. Certain idiomatic forms of data derivation have been found to be of widespread use. Transitive closures provide a declarative means for specifying a stereotypical form of search and may compensate for the absence of general recursion in a data model. Sum, min, max, and other reduction operators are frequently useful in domains with suitable algebras.

The framework may have mechanisms for inheriting properties of existing data descriptions, allowing for reuse of existing objects. Inherited properties may include attributes, relationships, methods, operations, and states. The inheritance mechanism may allow for full or partial derivation of properties from the parent object. For example, the user should be allowed to modify a method of an inherited object; it should be possible to also add properties to extend subtypes. If the framework permits multiple inheritance, then the framework must also define the order of inheritance of properties from ancestors.

4.11.2 Operations

The operations of the Derivation service include (1) operations to build a current set of derivative objects (e.g., Unix Make [39]) and (2) operations to maintain the specification of the derivation for general classes of objects (e.g., how Ada or C++ is compiled).

Many different objects are derivable from a given (set of) basis object(s). Which derivations are to be done autonomously may vary over time and be dependent on the states of processes or the OMS.

Examples of operations in support of derivation specifications are the UNIX Make rules, which may have many variations and alternatives (e.g., recompile just the changed things which need recompiling, recompile everything, just do a report on what needs recompiling, estimate the time it takes to recompile). Other examples are the APPL/A [81] derived relations.

Operations to maintain the specification of dependencies take different forms based on the type of specification. Use of relationships in the OMS may make use of a graphical or textual browser of OMS links and attribute values to change dependency links and specifications. Use of ASCII text files for declarative (makefile) and imperative (shell script) forms may use structured or general text editors. These operations may be post-conditioned to trigger reports, metrics, and automatic rebuilds.

Some Derivation service implementations may provide separate facilities for debugging of dependency specifications. This may take the form of "compilers," "rule sequencers," or builders run in alternate modes.

4.11.3 Rules

Rules may require that derivation graphs be acyclic and that derivation relationships can only be changed by repeating the derivation rather than by direct modification of the derived objects.

4.11.4 Types

Derivation services are heavily concerned with relationships and attributes.

4.11.5 External

External issues concern how the service is made available for use, i.e., by means of a language, procedural abstraction, and tools. Examples of issues are: (1) the visualization and reporting on the dependency enactment process itself, (2) the forms of editing, browsing, and reporting a specification of dependency, (3) the means of externalizing a dependency specification so it may be exchanged with other systems, other platforms, and various media of transmission along with or separate from the objects of the dependency relationships, and (4) mechanisms for controlling the autonomous (forward chaining) versus in-request (backward chaining) production of derived objects.

4.11.6 Internal

Derivation services are heavily influenced by the basis of the OMS implementation. For example, derivation in an ER OMS may be achieved quite differently from that in an O-O OMS.

4.11.7 Related Services

The Query service is a related one. The Derivation service deals with temporal issues, such as how to create a certain class of object from other classes (e.g., compiling source objects into executable objects). The Relationship service is more concerned with static relationships (e.g., what is the relationship between a source object and its author object? between a source object and its test data object?).

4.11.8 Examples

Examples of the specification of dependencies are: type dependencies in ODIN [17]; EAST [75] [84] and its graphical builder process; derived relations in APPL/A [81]; Unix Make tool [39]; and inheritance in Smalltalk [42].

4.12 Replication and Synchronization Service

This service provides for the explicit replication of objects in a distributed environment and the management of the consistency of redundant copies.

4.12.1 Conceptual

This service manages the concepts of ownership and checkout of multiple objects inherent in a distributed software environment. For example, replication or synchronization would deal with ways a piece of work is distributed between individuals. It also allows synchronization of read-only objects with an object at the

owner's workstation. Replication may be persistent by design (i.e., multiple databases) or temporary (i.e., portions of the database are checked out for work in a distributed manner.)

The Replication and Synchronization service covers many items that are not thought to be included in services such as the Location service, Version service, Data Subsetting service, Concurrency service, and Access Control and Security service. One main purpose in providing this service is that it provides a mechanism for many services to synchronize with each other.

4.12.2 Operations

The basic operations of this service are to provide synchronization of multiple objects in a distributed environment and manage replicated objects so that ownership is not hindered. These operations are, for example, check out, check in, perform consistency checks, update latest changes, replicate, and merge.

4.12.3 Rules

Replication rules will govern how to optimize the use of replicates. Rules are also likely that are intended to keep the replicates consistent.

4.12.4 Types

Replications deal primarily with subsets of the object base.

4.12.5 External

Due to its many relationships with other services in the reference model, the Replication and Synchronization service may be provided by components that also support other services.

4.12.6 Internal

There are significant implication issues involving optimization and synchronization of replicates. Implementations need to make the right replicates available conveniently while dealing with the problems of unavailability of distributed resources.

4.12.7 Related Services

Services related to the Replication and Synchronization service are the Location service, the Version service, the Data Subsetting service, the Concurrency service, and the Access Control and Security service.

4.12.8 Examples

PCTE explicitly addresses replication of portions of the object base. Objects may be normal, master or copy, depending upon their use [36].

4.13 Access Control and Security Service

The SEE objects managed by an OMS are the core of any SEE since they capture all of the information about the products (e.g., requirements, design, code, configurations, documentation), the project (e.g., plans,

milestones, project personnel, tasks), and the SEE itself (e.g., tools, users, roles). Access to this information may be controlled at multiple levels of granularity (e.g., schema, subschema, object or relationship type, object or relationship instance, and instance attribute) and may be based on multiple criteria (e.g., user identification, user role, current tool, current project phase, heuristics).

The Policy Enforcement services (clause 9), especially the Mandatory and Discretionary Access Control services, provide the mechanisms for access control to the OMS.

4.14 Function Attachment Service

This service provides for the attachment or relation of functions or operations to object types, as well as the attachment and relation of operations to individual instances of objects.

These capabilities may be provided as part of the OM data model (e.g., O-O model), supported by the data model (e.g., relationships in PCTE), or as a set of capabilities built outside of the OM.

4.14.1 Conceptual

Attachment of operations to object types and instances depends on the nature of the operations and the approaches for causing the operations to be invoked. In object-oriented data models, for example, operations (i.e., methods) are typically defined by explicitly associating them with the types, or instances in some cases. In some entity-relationship models such as CAIS-A, functions or processes may be defined as entities and attachment is achieved by means of relationships used to map operations to types (e.g., entity types) or data.

Management of the attachment differs depending on whether the attachment is inherent in or supported by the data model, whether it is provided within or outside the OMS.

The scope of an operation governs whether it applies to a specific object type or instance. Sharing may possibly relate separate OMS objects to common operations. Sharing may be by articulated paths as with relationships in an E-R OMS, or by implicit means, as in inheritance by position in class and type hierarchy structures. Shared operations may be inheritable by children of a class defining an operation, or an operation may be localized so it is not known elsewhere.

The means of activating those operations may be specified. The degree of polymorphism and the means for determining the choice of polymorphic operations is an issue related to attachment. The time of binding could be specified (e.g., early binding, such as compile time of the environment, or late binding, such as when installed during running of a program). The extensibility of the system without recompiling binary operations needs to be specified.

The connection to archiving and exporting services may be specified. Given that objects may be exported, then an operation associated with an object to be exported also may need to be exported. If the operation is performed by a tool that changes often, or exists on the system in different versions or branches, and the object's operation is dependent on a specific version or branch, again the tool (or equivalent) may need to be archived and exported whenever the object contents are exported.

Also, this service could be provided beyond object management and applied to the full SEE as a way to associate tools to data. If that is the case and if the tool is OMS- and framework-aware, a multifunction tool is very likely to present the multiple operations it may perform to the framework. Where operations are implemented as tools not written to directly use OMS objects, a capsule allows these tools to become part of the attachment process. The capsule also specifies any side effects within the OMS as a result of the tool operation execution. Capsules may take the form of highly structured specialized imperative programs or specialized languages for menu structure and interactive dialog presentation.

4.14.2 Operations

Operations for attachment may be embedded in the data model or made explicit. However, operations to manage attachment and relationships of operations may include browsing and editing of the attachments or relationships, updating and debugging of the attachments and relationships, and building, update, and synchronization of the attachments and relationships.

4.14.3 Rules

Rules for function attachment may address who is permitted to perform attachment operations or what sorts of function modifications are allowed.

4.14.4 Types

In an ER-based OMS, function attachment often deals with relationships. In an O-O-based system, it deals with methods.

4.14.5 External

If the attachment is inherent or supported by the data model, this dimension may equate to the data storage aspects of the external dimension in the Metadata service. Otherwise, there may be other ways of providing these services to users, including programmatic interfaces.

4.14.6 Internal

Implementation of function attachment is highly dependent on the OMS approach and data model. Function attachment is most natural to an O-O approach.

4.14.7 Related Services

Related services may be the services associated with the Metadata and Data Storage services, Common Schema, and Query. These are related because the same aspects of the OMS may be attachable to operations. Also, triggering may be related since operations may be executed based on changes to states of data; this could be another form of attachment. Versioning may control the management of multiple similar operations descriptions.

Derivation services may control the building and updating of the operations specifications, particularly when composed of multiple dependent object parts. The Registration service may be directly related to these capabilities.

4.14.8 Examples

Typically every object-oriented OMS provides the inherent capability of attaching methods to object types. Examples of attachment going beyond OM are as follows. Attachment descriptions are maintained in Microsoft Windows in a WIN.INI file, in a section which matches file extensions (files are desktop objects) to tool invocations. Equivalent capabilities are available on the Macintosh under MultiFinder.

4.15 Common Schema Service

This capability provides a common schema of the objects and (possibly) process descriptions in the database, in support of tool integration. This means that tools may use the common schema to describe and access the data they manipulate.

4.15.1 Conceptual

Having tools using the same DBMS is not sufficient for integration; it is necessary that these tools agree on a common (logical) definition of the objects (and operations) these tools may share. A common schema serves this purpose. It may also provide a set of services for accessing data in that schema; these services then translate the common schema to the underlying data models and data storage services.

The specific information model of the schema must be made available to tool integrators. This model should describe the higher-level schema and must additionally describe lower-level schemas by which tools may be reasonably expected to share data or control information. Lower-level schemas used by the framework- supplied tools may need to be accessible also to third-party tools for integration.

4.15.2 Operations

The schema needs to be created, browsed, edited, and navigated. Also, objects are created and accessed by means of that schema. Those capabilities are either provided by the underlying data model related operations, or they are built separately in which case the access may vary.

The ability and facilities to update, synchronize, distribute, and install the schema, in whole or part, may be needed.

4.15.3 Rules

The form of the common schemata needs to be such that it can be utilized by a variety of data models. Rules may also govern installation and effect of the common schema.

4.15.4 Types

Common schemata are likely to deal with object, relationship and attribute types, as well as attached functions and possibly messages.

4.15.5 External

If the common schema is defined using the selected OM data model, the schema definition uses the external capabilities of the Metadata service.

Export of the schema to accompany data archives and transmissions is vital because when the archive is brought online or the transmission received, the data structure must be known.

The support of common external forms and standards for exchange must be described. CAIS-A [30] and CDIF [37] have examples of common external form specifications.

4.15.6 Internal

Implementation may involve translation of the common schema into the data model of a particular OMS and vice versa.

4.15.7 Related Services

Related services are: Metadata, Data Storage, Relationship, Name, Access Control, Data Subsetting, Function Attachment, Version, and Derivation.

4.15.8 Examples

Examples of global schema are EAST SDS collection [9], the Project Master Database (PMDB) model [69], the PACT System services [82], and the SLCSE database-schema [80].

4.16 Version Service

The Version service provides capabilities to create, access, and relate versions of objects and configurations.

One of the distinguishing features of engineering environments (for software or otherwise) is that recording and maintaining information from previous states of a system is not only interesting but a definite requirement. The service concerned with managing data from earlier states is the Version service. Change throughout development has to be managed in an SEE and the inclusion of versioning is one of the means of achieving this.

4.16.1 Conceptual

Versioning is based on a simple concept: interest in aspects of an item's former state (including relationships with other, perhaps versioned, objects). The variations on this simple theme may generate many complex scenarios.

As items (e.g., objects) in the OMS evolve, there may be a need to replace them with newer versions and to keep histories of them. A "version" provides the ability to reference the newest instance of such an item or one or more of its predecessors. The terms "variant" or "revision" are sometimes also used in this context, either as synonyms or having slightly different semantic meanings. The term "variant" is commonly used to refer to a version that has a parallel existence. Thus there may be variants of a piece of software, each implemented differently but which all fulfill the same set of requirements, or there may be variants of the software for different platforms, or again there may be variants incorporating different amounts of comment. The term "revision" on the other hand is commonly used to refer to a version superceding a previous version. Revisions thus occur in a single line of development, whereas variants are formed when the development branches and versions exist in parallel.

There have been several presentations of generalized version models [27, 56]. Just some of the choices available are described here.

A version may be treated as a single object which has a set of attribute values. Creating a new version of the object identifies something new. The version may or may not have the status of an object, but it is something distinguishable with an associated set of values. The reason for which another state of the same logical object is identified may determine whether the new thing is a revision or a variant. For example, a module with a bug fix is the same module. Both must continue to exist if someone is using the earlier module. One needs to know that the second module is a correction of the first. The distinction between revision and version is subject to the possibly differing perceptions of people.

Adding to this complexity, the management of relationships between objects and their versions has to be achieved.

A further matter to take into account is that an enterprise or project may want to use its own version model or different models for different types of work. For example, one hundred people on an operating system development project require support significantly different from six people writing an accounting package. SEE frameworks may have version facilities that are parameterized or flexible enough to allow this (e.g., environments which come as a kit of parts).

4.16.2 Operations

Just as there is typically a data model associated with a data storage service, the Version service may be presented as a version model. The major operation is normally create version (sometimes referred to as an update of the previous version), since delete and update operations are often considered unnecessary and are constrained by the structural rules of the version model.

A locking operation on a version is sometimes known as reserve. A merge operation may support more than one source for the creation of a merged version.

4.16.3 Rules

A version service may include rules regarding access to older versions, length of retention for superceded versions, or deletion of superceded versions.

4.16.4 Types

Versioning can be applied to objects of any type. It may result in new objects with attributes recording who created the new version and when and who its predecessors are.

4.16.5 External

With potentially very large numbers of versions, and complex relationships between them, their identification and interrogation in useful ways is essential. A graph representation is a method often employed.

The version model may be utilized to provide sophisticated data transaction-like mechanisms.

4.16.6 Internal

As hinted above, the version model may be implemented as an intrinsic part of the data model. In this case optimization techniques may be used which save time and space in regenerating previous or current versions.

4.16.7 Related Services

Services closely related to the version model may be the Data Storage and Relationship services (the version model may be an intrinsic part of the data model or expressed using concepts in the latter), the Composite Object service, which may allow versions of composite objects or compositions of different versions. Other closely related services are the Name and Query services.

4.16.8 Examples

The set of systems which address versioning includes: PCTE [36]; DSEE [58]; ATIS [26]; NSE [1]; Aspect [44]; and Damokles [28].

4.17 Composite Object Service

This service creates, manages, accesses, and deletes composite objects, i.e., objects composed of other objects. This may be used to form configurations. It may be an intrinsic part of the data model or a separate service.

4.17.1 Conceptual

One often wants to look at and operate on a "thing" as a single item but at other times consider it as a structured or composite collection of other things. The Composite Object service offers the facilities to enable this. Terms used for the collective whole are "compound objects" or "composite objects."

One particular example of this kind of service is often found in software development environments. This is where a software system, composed of many source code modules, is to be treated as one object, often called a configuration. Generally, a configuration consists of a collection of objects intended for a particular application. A configuration may refer to an explicit version of each of its components or may simply require the newest version of each.

A general case of the above is a composite object that is not only defined by its constituents and their relationships, but also in terms of transformations of other (possibly composite) objects.

Composite objects can be used to realize more than just software configurations. For example, composite objects may also be used to look at documents or tests.

The description of how composite objects may be defined and operated upon may be found in "configuration models." General models for configuration management are described in Heimbigner [46] and Dillistone [27]. Many of the statements applicable to the Version service (see 4.16) may be applicable here, and the reader is referred to that clause.

4.17.2 Operations

Operations on composite objects may be the same as operations on the data model if composites are part of the data model. They may typically correspond to the same or a composition of the existing operations of objects (e.g., assign, copy, backup).

4.17.3 Rules

Rules for composite objects may govern what sorts of groups of objects may be included in a composite object (e.g., a random set or a graph). There may be rules regarding one or more objects being distinguished (for example, main program) within the composite object.

4.17.4 Types

The data model may allow a composite object to be treated as a type which may be instantiated like other types. The composite object service may enable definitions of composite object types which may later be instantiated. There may be several instantiations of the same composite object type.

4.17.5 External

There may be a procedural interface to create and manage composite objects by which the members of the composite object are identified.

4.17.6 Internal

One way of representing composite objects is by (directed) composition relationships between parts. The whole is then the transitive closure of the parts over the composition links.

4.17.7 Related Services

The Composite Object service may be related to the Version service since composite objects may have versions. It may also directly relate to the Metadata service if the data model allows treatment of composite objects as a type. The Relationship service provides links among composite objects.

4.17.8 Examples

PCTE [36] provides support for composite objects [36]. Most configuration management systems provide a limited notion of composite objects in the context of configurations.

4.18 Query Service

The Query service is an extension to the Data Storage service's read operation. It provides capabilities to retrieve and present sets of objects according to defined properties or values.

4.18.1 Conceptual

The values returned by the Query service may be those directly associated with the set of objects. The way the set of objects is determined may be expressed differently. It may be achieved through the names of the objects, by matching values associated with objects, by following links or relationships with other objects (navigation), or by combinations of these. In some systems the object being queried may be treated in semantic or logical ways and "inference," "goal oriented," or "knowledge base" techniques employed.

The study of query services, query languages, and query optimization is an ongoing research topic. Some environment frameworks use well-understood techniques (e.g., SQL) while others may use experimental languages.

4.18.2 Operations

A typical operation is query, which maps a set of objects and relationships among them into values derived from other objects (e.g., predicates, functions, properties). Other transformations on data access are also possible. Operations, such as sort or merge, are examples of these transformations.

4.18.3 Rules

Rules may govern the complexity of query that is allowed and the response received to a query on an object for which the user does not have appropriate access rights.

4.18.4 Types

Queries themselves may be stored as values. These may be retrieved and executed as part of a high-level query.

4.18.5 External

An environment framework may support more than one query language (e.g., to have more suitable languages to suit a particular type of purpose or user).

The Query service may be made available as a set of interface routines or as a query language (or both). In some cases the same functionality may be provided through different languages (e.g., relational algebra and relational calculus).

4.18.6 Internal

The implementation of the Query service and query languages is closely related to the implementation of other object management services. In many cases simple implementation of query resolution results in a very inefficient service (particularly in the distributed case) and optimizing parts of the process may be vital. The more information available to the Query service concerning the expected and actual use and location of objects, the more sophisticated optimization techniques may be used.

The implementation of this service must recognise the fact that a significant amount of data may be archived or that a machine in the network may be inaccessible. A query may also access data with different security values and often is implemented such that common queries are processed rapidly whereas other queries involving secure or archived data may be quite inefficient.

4.18.7 Related Services

Many parts of an environment framework may use the Query service to provide information as part of their service. The degree to which this happens depends on the degree to which the basic objects of other services are stored by the Data Storage service.

There are cases where applications or users understand the specific data model being used to implement a service and use the Query service to make their own specialist queries. In fact, a service may simply make its specific data model available and expect queries to be performed by use of the Query service. Note that the availability of a Metadata service (see 4.1) makes this a very feasible proposition.

The Query service is intimately connected with nearly all of the object management services. Apart from its general usefulness for enabling querying for other services, it is of importance to the Data Subsetting service.

4.18.8 Examples

Examples from real systems include: Aspect's relational algebra [44] and the PACT query service (DQMCS) built on top of PCTE [84] and ModifyER which provides graphical navigation and update of the SLCSE schema [80].

4.19 State Monitoring and Triggering Service

The State Monitoring and Triggering service enables the definition, specification, and enactment of database states and state transformations and the actions to be taken should these states occur or persist.

State monitors and triggers enable the coordinated use of tools that were independently designed. For example, monitors enable a data-consuming tool to enforce requirements on a data-producing tool's output; triggers enable a data-consuming tool to be notified when relevant data becomes available. In effect, the OMS may become an inter-tool signalling channel. Those services also enable project users or organizations to tailor environments to their own preferences and needs. They may also be used internally to a tool to simplify implementation by centralizing statements of processing that must be performed at numerous places in the software.

This remains a relatively experimental area of OMS. There is not widespread agreement on terminology, capabilities, or operations.

4.19.1 Conceptual

The definition part of the service may be separated into the definition of particular states and the definition of state transformations. Examples of these might be: when the status of all test results becomes "passed" (a static example); or when the number of lines of code written in a day exceeds 100 (a transform example). There may also be support for logical combinations of conditions. An example of a state persisting is "while there are outstanding bugs (send a monthly notification)."

The second part of the service, i.e., carrying out an action when a state (or transformation) is detected, may take several forms. It may be a notification to the person or role or process that defined the state or to some other process or person described in the definition. It may, alternatively, prevent a state or transformation from being reached.

It is important to note that these actions are started by the system (semi-)automatically from the point of view of the tools or the users of the environment framework.

Monitors are used to enforce some kind of consistency on OMS objects. The consistency may be of a form required for correct operation of software (e.g., module hierarchies are acyclic) or may represent organizational policy (e.g., at least 50% of the members of a design review team must not have been responsible for the design under review).

These services may not necessarily react to inconsistent states by rejecting them; they may augment an action or transaction (e.g., by supplying a default value for a required attribute) or even alter that action or transaction (e.g., replace one of the proposed members of the review team to satisfy the above-stated policy). When multiple monitors or triggers apply to a single proposed or achieved transaction, their scheduling is a semantic issue.

4.19.2 Operations

Typical operations are: create, update, delete, query, attach, detach, arm, and disarm a monitor or trigger. Attach and detach operations (dis-)associate a state with a set of actions. Arm and disarm are considered low-cost operations which may be applied to a monitor or trigger in a running program. They may have global and process-specific variants. Attach and detach are thought of as more expensive operations, perhaps requiring recompilation of code. Operations may also be available for creating groups of monitors or triggers that may be armed or disarmed and applied collectively.

4.19.3 Rules

Rules for state monitoring may address the range of OMS state changes that can be detected and thus result in triggering.

4.19.4 Types

State monitoring is potentially applicable to all objects, relationships and attributes in the object base. A trigger object may be a distinguished type.

4.19.5 External

Systems typically provide a declarative sub-language for characterizing the situations to which these services apply. This language is typically related to the system's query or object navigation language, but provides for: 1) parameterization (quantification) and 2) ability to refer to "change" or to differentially query two or more states.

The declaration of some highly idiomatic forms of monitors (e.g., cardinality restrictions, range restrictions, default attributes) may be made implicit in the syntax for defining a data schema.

4.19.6 Internal

Both monitors and triggers must characterize the situations in which they apply and their reaction to that situation. Implementations vary in the power of situation characterization; there also is variability in the allowed reactions, although for triggers arbitrary procedures may be allowed.

In general this service is parameterized by object classes or types. These parameters are bound in the process of unifying a transaction or action with a situation description. The bindings are then more available to the reaction. The reaction may or may not also have access to the transaction itself, in the old and new or proposed states of the OMS.

Important considerations in the implementation of these services are: 1) the power of the language for characterizing situations and 2) whether the complete set of monitors or triggers is known at the time code that performs transactions is compiled.

4.19.7 Related Services

The State Monitoring and Triggering service is related to many other services, both to help and support the definition of states and their transformations and to ensure that appropriate actions are carried out. These services are sometimes used to achieve capabilities for which the derivation service is intended. The semantics of monitors or triggers may be bound to some notion of transaction.

4.19.8 Examples

Examples of the State Monitoring and Triggering service may be found in ATIS [59] and PCTE [36]. The Vbase object-oriented DBMS [3] provides triggers to be associated with changes of values of attributes or the invocation of methods.

4.20 Data Subsetting Service

The Data Subsetting service 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 (e.g., actual objects). Since a full life cycle process is not conveniently or usefully understood as a single complex system, the SEE framework may provide support for selecting parts of the OMS necessary to carry out specific subprocesses of the overall project. The basis of division may vary; for example, it may be according to ownership, to security

clearance, or to relevance of an installed process or particular subprocesses. The SEE framework may provide all, few, or even none of these.

4.20.1 Conceptual

The Data Subsetting service deals with ways to access, define and manipulate a subset of the OM model and data (e.g., types, relationships, objects, and operations). Data or operations may be allowed to be in more than one subset at the same time. The most general form of definition is an intentional one which defines some of the properties of the objects and operations to be included in a particular OM subset. Other forms include derivations, which are defined in terms of the operations provided by the OM services, and user-written transformations in some general-purpose programming language.

Hierarchies of subsets may exist, in which case the underlying objects and operations may themselves be derived. Some notion of self-consistency of a subset (e.g., all the types referred to by objects and operations in the subset have to be available) may be imposed by the Data Subsetting service.

Subsetting of the OMS may be part of a larger concept of creating sub-environments where different users have access to specific subsets of data, specific processes to execute and limited availability of certain tools. A Data Subsetting service may have a major impact on how integrity and consistency is dealt with by an environment. While objects may be visible only within an OM subset, they must not be inconsistent with objects within and without that subset. Overall consistency may be provided by control of the visibility of such "inconsistent" data.

4.20.2 Operations

The set of operations supported by the Data Subsetting service could include the following: create, update, delete, query subset definitions, instantiate a subset (to cause the bindings to be made), and possibly operations to access subset elements (e.g., types, objects).

4.20.3 Rules

It may be a rule that a subset must be self-consistent. In particular it may be necessary that all relevant types are included in the subset.

4.20.4 Types

The complete definition of a subset is a complex item and it is likely that some definitions could be re-usable and treated as types.

4.20.5 External

The Data Subsetting service is one which should be visible to SEE framework users. This may be provided by the metadata interface, by sub-environment definition languages, or programmatic interfaces. Significant reuse of subsets could be envisaged. Whatever external representations are chosen may take this into account.

4.20.6 Internal

Subsets may be implemented by a two-way transformation between the types, objects, and operations in the underlying environment and a new set of derived objects and operations of which the subsets actually consist. The transformation has to be a two-way mapping so that changes to the objects caused by operations at

the sub-environment level may be reflected in the underlying data. This transformation may be in terms of the internal operations provided by the OM services or user-written transformations in some general-purpose programming language.

There are binding issues concerning whether subset binding happens at definition time or dynamically at access time.

Environment subdivision may provide for performance gain which may be achieved through simplifications of the distribution strategy (in fact making it mirror more closely the way people typically organize their work).

In some systems such internal details are expressed as though they are an essential element of the conceptual model (e.g., the Workshop system [18]). It is important that the conceptual and internal aspects of subsets are clearly separated in descriptions and discussions.

An interesting issue is how the boundaries imposed by the Data Subsetting service impact upon boundaries of other services.

4.20.7 Related Services

These services may be subsumed by the Metadata service. The Query and Derivation services may be used in connection with these services. The decisions about when to do the various bindings associated with the Data Subsetting service may be related to the models of work supported by Process Management and Sub-Environment services, in particular to the Process Visibility service (see 5.3) and Sub-Environment service (see 10.4). For example, a particular process (or process type) definition may be bound to a particular sub-environment definition, and when the process starts the particular bindings in the OM subset may be made.

There are also clear relationships with the Policy Enforcement services (clause 9) in the environment framework.

A Data Subsetting service may also be designed to relate well with the Data Transaction and Concurrency services, in some cases simplifying them considerably as concurrent access may not be necessary within a sub-environment.

4.20.8 Examples

Examples applicable to OM and SEEs from real systems include: the Schema Definition Sets (SDS's) and working schema of PCTE [36]; the "role databases" of ISTAR [25]; the "workshop" and "studio" processes of the Workshop System [18]; Perspective's "domains" [23]; a "view" in CAIS-A [30]; and the role-based access to tools and subschemas of SLCSE [80].

4.21 Data Interchange Service

The Data Interchange service offers two-way translation between data repositories in different SEEs. The object management services handle objects within an SEE framework, but there is a need to be able to exchange objects among environment frameworks. This may arise, for example, when software developed within one project's environment is to be enhanced by a different project in a different environment. In addition to the software itself, all design and development information has to be transferred. Another example is the release of new measurement tables to be used for checking quality. In all such cases the objects have to be represented in a form suitable for transfer across SEEs.

4.21.1 Conceptual

This service is to support translation between data repositories created by the same software system in different hardware platforms or repositories created by different software systems. The problem lies in reaching agreement on standard formats for all types of objects: text, graphics, audio, and video. It is important to note that no guarantee of accuracy can be given where two different environments represent things differently (e.g., floating point numbers).

4.21.2 Operations

The basic user-visible operations provided by the Data Interchange service permit the user to send and receive data from other environments. This may also include operations to compress or expand data using data compression techniques to provide for more efficient transmission of data or more compact storage requirements. The invisible operations handle the translation process. If desirable, the user is able to view the invisible process, for example, to verify the translation process.

4.21.3 Rules

There is a need to preserve object identity under data interchange.

4.21.4 Types

Data interchange may involve all types in the object base. There may also be distinguished objects that represent subsets of the object base in a specified interchange format.

4.21.5 External

This service is likely to have two external forms. One is the details of an interchange format. The other is procedural calls for creating and retrieving objects in a specific interchange form.

4.21.6 Internal

The difficulties of implementing data interchange are involved in effective translations to interchange formats and efficient identification of objects to be included in an interchange.

4.21.7 Related Services

The Data Interchange service may have to translate the outgoing data to a standard form, and it may have to translate incoming data to a form readable by the environment. The Data Transaction service may be involved in this process. Also, other services may be employed, such as Name, Data Storage, Relationship, Location, and Access Control.

4.21.8 Examples

PDES and Semantic Transfer Language (STL) [50] are example standards in this area. The Sendmail Transfer Protocol (SMTP) and File Transfer Protocol (FTP) as part of the TCP/IP (Transfer Control Program/ Internet Protocol) networking standards provide common formats for transferring information among sites in the worldwide internet computer network. CDIF [37] is an emerging standard in this area.

5 Process Management Services

The general purposes of the process management services in an SEE are the unambiguous definition and the computer-assisted performance of software development activities across total software life cycles. In addition to technical development activities, these potentially include management, documentation, evaluation, assessment, policy-enforcement, business control, maintenance, and other activities.

Execution, or enactment, of these activities is via processes, which undergo changes of state as certain events occur. The process state encompasses a wide variety of information. Some of it is process specific. This portion includes both the process' current enactment state (e.g., which processes are executing, which agents are enacting them, and what resources are being used by the processes) and the instantiation state of the process definition, for example, the particular choices made for the various classes (such as process assets and resources and other parameters) used in the process definition or the restrictions placed on these choices (e.g., JSD used as the design method, only senior Ada programmers code critical modules).

Other aspects of the process state involve the work products being produced, modified, or used by the process. The state of the work products and their relationship to the process and its resources are also part of the process state (and are an important aspect to capture in the process history).

An SEE may also provide process simulation, analysis, or behavior modeling services that provide information about potential process enactment, but do not actually serve the role of performing a project development process and transforming project objects into project end-item outputs. These services may provide useful information to verify and validate a process' performance or accuracy before it is prepared for enactment. While these process-related services do place demands on lower level services, they are not provided directly by current SEE frameworks, nor is it anticipated that they will be in the foreseeable future. They are generally regarded as beyond the scope of SEE frameworks and will not be dealt with further in this reference model. They are in the scope of reference models for full SEEs, e.g., [70].

5.1 Process Development Service

It is expected that an organization may have a library (repository) of process assets, each of which may be a complete process definition, a process element (subprocess), a process model, a process architecture, or a process design. Also, an SEE may provide facilities to develop new process assets. These are assembled and tailored to form *process definitions* (by process engineers in an activity called *process development*). A complete process definition might be *instantiated* (by combining it with a process plan consisting of assigned enactment agents and resources) to form an instance of an *enactable process*. Process definitions may cover a spectrum, including the following:

- the activities of an individual, a project user role, or a machine agent that performs some creative activity (e.g., design)
- the decomposition of processes into process elements (subprocesses) and atomic process steps and their assignment to project roles or role types
- the decomposition of a complete process definition and assignment at specific group or project levels
- the institutionalized corporate business and technical processes and policies of the company in which the software is being developed.

As with other service categories, an SEE may have support facilities that are well-suited to only a subset of this spectrum.

5.1.1 Conceptual

The definition of a software development activity (a process) may involve a number of different types of information, for example:

- pre-conditions for enactment
- post-conditions (validation) for completion of enactment
- constraints or policies to be checked or enforced during enactment
- project data operated upon, both input and generated
- pre- and post-enactment of other processes or subprocesses
- allowable concurrency and synchronization (if any) with other processes or subprocesses
- process events potentially of interest for signaling or measuring
- specification of events that are to be known by other processes or the SEE's enactment facilities
- the degrees of freedom for process change (e.g., only during development, during initiation of enactment, during enactment)
- process state information operated upon (in the case of process elements that provide process monitoring or control services, see 5.4)
- enactment agents and their mappings to process elements (this is often not part of a process definition before instantiation)
- methods or algorithms or other SEE services for transforming object inputs to outputs
- information constituting process enactment state
- product, project management, and process metrics to be collected
- combinations of process elements that should be regarded as "atomic" in the sense that all must complete according to specified completion criteria or all affected project objects are restored to their input states (such an atomic combination defines a *process transaction*)
- in the case of higher level processes or complete process definitions a process architecture; overall specification of sequencing and other relationships (e.g., control activities such as monitoring, history recording, auditing) between constituent process elements, including specification of a start point and termination criteria (if appropriate); capabilities for tailoring and adapting process architectures and their instantiations with process assets; and management-oriented project plans reflecting certain information from the process definition.

An SEE may provide independent services for defining some of the above, or it may provide a unified facility for process development.

Sometimes a process definition is quite informal (e.g., English or some graphical languages). However, to be enactable, a process definition might need a significant degree of formality. Sometimes some of the above information is organized into so-called "models" that may be defined in the SEE. For example, a project data model and an SEE resource model are commonly available in an SEE and are "givens" for a process developer (i.e., the process engineer); a "user" or "user role" model may also be a given for a project or for an organization, or they may be objects that the process developer may build into the model. Information dealing with relationships between project objects and activities or tools may be represented in an "activity" or "workflow" model. A process definition (or a *complete process definition* [38]) may sometimes be thought

of as a "cross product" or aggregate perspective of the information contained in some or all of these other models; as such, a process definition may only be a set of models, or it may be a structure incorporating and possibly adding information to the other models.

In general, the granularity of a process may range from "do a whole project" (a complete process) to those at the level of "fix this bug." For this reason, a process may be manageable as a whole unit, and the process developer may wish to refine the process definition to describe subprocesses. Just as subroutines in programming languages share almost all the properties of programs, so subprocesses may share all the properties of processes. The relationships supported between parent and child processes are very much at the discretion of an SEE designer, although rules may be expected, such as that a child process' visible project data subset must be a subset of its parent's visible data subset or that a parent process cannot complete until all its child processes have.

5.1.2 Operations

Process development operations include:

- create instances of process definitions
- update instances of process definitions
- delete instances of process definitions.

There might also be operations for users to define types in the SEE for varying forms of process definitions, e.g., architectures, plans, models, and policies. In this case, a create operation is a type instantiation operation.

There might be a pre-enactment or process instantiation service to prepare transformations of a process definition into a form upon which the SEE's Process Enactment service may operate.

An SEE might also provide a Process Exchange service analogous to the Data Interchange service that converts a process definition in a form supported by the SEE to another form (either for exchange between SEEs or for translation into an alternate form supported by the SEE).

5.1.3 Rules

An SEE may have constraints concerning how and when a process definition may be changed. An SEE may have limitations in visibility and scoping between processes.

5.1.4 Types

An SEE may support process definitions as basically untyped objects in its object management system, it may provide a single type for process definitions, it may provide multiple types for varying forms of process definition information (e.g., process assets, process architectures, plans, models, policies), or it may provide type definition facilities for process developers to create types appropriate for their process development approach.

5.1.5 External

The operations of this service may be provided as procedure call interfaces.

However, beyond that is the representation of semantics of a developed process. If any formality in process definition (and hence the potential for process enactment) is supported by the SEE, logically the external dimension is providing some manner for users (e.g., process engineers) to represent process definitions or input

them to the SEE. An SEE may support one or more different representation approaches. Possibilities include supporting one or more of the following:

- free-form English language descriptions (sometimes referred to as a "process script," but may be a document)
- semi-formal, structured English
- a process programming language (much like a programming language)
- a graphical notation (that might be equivalent in semantics to a process programming language)
- a process modeling language (that might be any of the above or a form of them tailored for abstraction)
- a set of declarative rules (e.g., a knowledge base)
- a forms-filling paradigm
- an interactive user dialog with an SEE service.

The features (analogous to software programming language features) in any of these representation approaches provide the various types of information involved with process definitions itemized in the Conceptual dimension.

With the exception of English, semantically equivalent formal representations could exist in all of the above representational approaches. On the other hand, different approaches may be used to represent varying levels of process definition (and hence not be fully equivalent); or an SEE may support multiple representational approaches of the same style that convey different aspects of process definition information (e.g., one process programming language for workflow and a different process programming language for resource allocation, or one for defining top-level organizational views of processes and another for specifying complete process definitions).

An effective user interface is a crucial factor in the effective incorporation of process development (and all Process Management) services into an SEE.

5.1.6 Internal

Process definitions could be stored and managed by employing the SEE's Object Management services, or an SEE may provide an independent set of services unique to creation, storage, and management of processes; in this latter case, a separate SEE object store may exist for process definitions and be accessible by other services and tools, or all process information may be encapsulated with a closed object store. Process definitions may also employ services (e.g., resource models) provided by Framework Administration services.

5.1.7 Related Services

Object Management services and Framework Administration services will often interact with the Process Development service, if not actually provide some or all aspects of process development.

All other categories of Process Management services listed in this clause may have significant interaction with and dependence upon the Process Development service. The Process Enactment service requires the existence of at least some of the services and capabilities described for Process Development, and the other categories of Process Management services might be regarded as extensions of the Process Development service.

5.1.8 Examples

IPSE2.5's PML [79], Arcadia [52], EAST [9], ESF [73], EFA [47], Enterprise II [60], COHESION [14], Marvel [7], and SLCSE [80] all provide some process development services.

5.2 Process Enactment Service

A process definition may be *enacted* by *process agents* that may be humans (project user roles or individuals) or machines.[3] An SEE may support either or both methods of process enactment (humans or machines as process agents). Note that enactment is only one possible usage of process definitions; others could include analysis, education, and communication.

5.2.1 Conceptual

The Process Enactment service provides facilities necessary to control and support the enactment of processes that have been installed in the SEE. An SEE's process enactment capability may range from a shallow level (e.g., a help facility or simple "shell" invocations) to higher levels (e.g., knowledge-based process understanding or processes encoded in a process programming language). The development of one process may be underway while other processes are being enacted. The change of a process' definition during enactment may be supported (and is recognized to be a reality on typical software development projects). A process program may be allowed to evolve during its enactment.

The Process Enactment service provides ways to access up-to-date information on the state of enacting processes. Such information revealing the progress of an enacting process (that is a project) may be important to all members of the development team, but benefits managers in particular. A variety of tools might be written that operate on process state information.

5.2.2 Operations

Process enactment operations might include:

- instantiate a process definition, by assigning project resources (e.g., staff, schedule, equipment) to process elements
- bind process elements together, or process elements to process architectures (analogous to software linking)
- enact an instantiated process definition
- suspend and restart an enacting process
- abort an enacting process
- trace an enacting process (e.g., record and store snapshots of resource utilization, enactment times, error occurrences, or other desired parameters at specified time intervals or on the occurrence of specified events in SEE activity)
- checkpoint and rollback an enacting process
- change (dynamic).an enacting instance of a process definition

[3] *"Enact" is the currently preferred verb (over "execute") by the software process community because "execute" has too many mechanistic connotations.*

5.2.3 Rules

The semantics, possible inter-dependencies, and possible parameters associated with each enactment operation may be very similar to those for execution of software programs. The extent to which dynamic process change is supported is a rule. There may be rules about the effect of checkpoint, rollback, abort, etc., especially in the context of a hierarchy of processes.

5.2.4 Types

For an SEE to support process enactment, it needs representations of enactable process definitions with sufficient semantics and details for the level of enactment desired.

A type of information representing the state of enacting processes typically also is known in the SEE. This may be a combination of many types of information (e.g., metrics, "process program counters," event lists, condition triggers, process transaction status).

Depending on the semantics of the process definition form enactable in the SEE, other distinct types may also be required.

5.2.5 External

Access to process enactment operations may be by command language syntax, such as in the UNIX "shell," or by specific procedure calls on process enactment services.

5.2.6 Internal

Processes may be enacted by basic SEE framework services, tools, and humans; more likely, process enactment is a combination of all. Depending on the formality of the process definition and the extent of control effected in process enactment, there may be an SEE service or component (sometimes called a "process manager") that provides the overall control, coordination, synchronization, and communication of process enactment for a process larger than a single tool; this central process enactment facility may be embedded in the SEE's framework, may be provided by a tool installed in the SEE, or even by human user interaction with the SEE, or, again, by a combination of these options.

Accommodation for *process evolution* (including dynamic process change during enactment, i.e., during a project) may be provided by an SEE. If provided, this interaction of process definition and process enactment is probably supported by a more complex implementation approach than if just static process change is provided.

Process enactment facilities employ object management facilities because the essence of a process is to transform or create project data. The enactable representation of a process definition may be stored in the SEE's object management facility or independently. (See also 5.1.6.)

5.2.7 Related Services

The Process Development service is closely related, as it provides the representations that are enacted, including user-specification capabilities corresponding to the semantics of process enactment supported by the SEE.

All other Process Management services (if not provided by the Process Development service) may extend the enactment semantics of a SEE's process definition representation approach or utilize process state information maintained by the Process Enactment service.

Object Management services may be related as Process Enactment interacts with them, at least for operation upon project objects and sometimes for storage of process definition representations and process enactment state information.

Processes are typically the mechanism used to enact tools supporting some method in an SEE.

5.2.8 Examples

IPSE2.5 [79], EAST [9], Arcadia [52], EFA [47], Enterprise II [60], COHESION [14], SLCSE [80], and EIS [57] provide some support for process enactment.

5.3 Process Visibility Service

This service provides facilities for the definition and maintenance of visibility and scoping information associated with enacting processes. Several enacting processes may cooperate to achieve the goal of a higher level process or a complete life cycle process. The extent of such cooperation is typically part of the definition of processes and may be provided by integrated visibility features in a particular process definition representation. Alternatively, independent services may be provided to deal with interprocess interactions such as visibility of common data, common events, and propagation of information; these are the subject of this service.

5.3.1 Conceptual

The Process Visibility service defines which portions of an enacting process' state (including the work products it is producing or modifying) may be visible to other enacting processes and provide control over when and where these states are visible. An example is process interaction, an interchange between two cooperating, enacting processes. The interchange may be for the purpose of interchange of data or for the purpose of coordination (interchange of control). Visibility and scoping may provide capabilities for the dynamic and evolutionary aspects of processes that cannot be adequately addressed by traditional statically scoped process definitions. For example, a well-defined subprocess to take standard emergency corrective action may need to be enacted within the context of any of a number of other enacting processes if and when the emergency situation arises (e.g., funding cutback, unexpected management review).

Visibility pertains to product information as well as to process information. That is, an enacting process may likely need access to product data, history, or state in order to accomplish its goal.

In the case of process support, visibility requirements for an enacting process may be a function of time and allow access to intermediate results in a manner dissimilar to the atomic process transaction model typical in database systems. While it might be appropriate to provide full visibility to process state or product information of an enacting process to other enacting processes as of the completion of the last successful "process transaction," there are situations when an enacting process may need earlier visibility to "work in progress" information that is being generated by another enacting process. Therefore, services to define regions within an enacting process when it allows or obtains visibility to or from another enacting process may be provided.

Also, this type of service could be provided beyond process management and applied to the full SEE as a way of subsetting other classes of objects in the SEE, e.g., tools and data.

5.3.2 Operations

Visibility and scoping operations might include the following:

- yield visibility of specified information to a designated entity(s)

- preclude visibility outside of the defining entity
- define the scoped region
- make the specified scoped region visible
- control access rights associated with the designated operation (e.g., read-only versus modify).

5.3.3 Rules

In a sense, this service effects rules affecting the interaction of process definitions with the Process Enactment service.

5.3.4 Types

It is possible that a framework has fine-grained types for representing the linguistic aspects of process definition such as scopes and regions. These types may be related to types of the Data Subsetting (4.20) or Sub-Environment (10.4) services.

5.3.5 External

If any formality in process definition (and hence the potential for process enactment) is supported by the SEE, the external dimension provides some manner for users (e.g., process engineers) to represent process definitions including visibility and scoping information, or input them to the SEE. An SEE may support one or more different representation approaches, as described in the External dimension of the Process Development service.

Alternatively, this service may be provided by a procedure call interface.

5.3.6 Internal

The Process Visibility service could be supported by the same mechanisms that support process enactment (basic SEE framework services (especially, OM services), tools, and humans) and that support process development (especially open or closed object stores for specifications). This service could be supported by the Data Subsetting and Sub-Environment services.

5.3.7 Related Services

Process Visibility may use the operations within the Data Subsetting (4.20) or Sub-Environment (10.4) services to provide scoping properties.

The Process Development service is closely related, as it provides the representations that are enacted, including user-specification capabilities corresponding to the semantics of process visibility and scoping supported by the SEE.

All other Process Management services (if not provided by the Process Development service) may extend the enactment semantics of a SEE's process definition representation approach or utilize process state information maintained by the Process Enactment service.

Object Management services may be related as Process Enactment and its visibility and scoping semantics interact with Object Management services, at least for operation upon project objects and sometimes for storage of process definition representations and process enactment state information.

5.3.8 Examples

EFA [47], Enterprise II [60], SLCSE [80], EAST [9], ESF [73], Arcadia [52], and IPSE 2.5 [79] all provide some process visibility and scoping services.

5.4 Process Monitoring Service

The Process Monitoring service observes the evolving enactment state of processes, detects the occurrence of specified process events, and enacts other processes to respond to these events. This service (or part of it) is sometimes referred to as "Event Management." The "other" processes enacted as a consequence of observed process states and events are sometimes referred to as "control processes."

5.4.1 Conceptual

Process monitoring observes the evolving enactment state of a process (or set of processes) and its history, and may take actions on the basis of these observations (such as by ensuring notifications and scheduling activities).

Conceptually, events are any change in process state, but the event detection mechanism may place restrictions on what subset of process events it can detect and therefore what kinds of event specification it accepts. This service supports the definition of "events" and actions to be taken (or services, roles, or tools to be notified) should an event happen. The exact definition of what constitutes an event is a decision that might be constrained by the particular SEE framework designer. However, one would expect the granularity of events that may be defined for, and detected by, this service to be at least at the level of initiating, interrupting, or completing enactable process elements. Event definitions may be specific, such as, "when designer C finishes the next specification," or more generic (specifying a type of event), such as, "each time a document is approved by the quality assurance department." The ways an event definition may be expressed are dependent on the external presentation of processes made available by the Process Development service (see 5.1).

5.4.2 Operations

Operations for this service may include:

- create an event definition
- update an event definition
- delete an event definition
- query an event definition
- set and unset (operations that control if the checks are to be applied or not)
- manipulate the control process definitions associated with an event
- attach and detach (operations that (dis-)associate sets of actions with events)
- query an enacting process' state

5.4.3 Rules

All Process Monitoring operations are subject to constraints in their effect as defined by visibility operations.

5.4.4 Types

It is possible that a framework has fine-grained types for representing the linguistic aspects of process definition such as events, process transactions, and monitors.

5.4.5 External

If any formality in process definition (and hence the potential for process enactment) is supported by the SEE, the external dimension provides some manner for users (e.g., process engineers) to represent process definitions including identification of process events, or input them to the SEE. An SEE may support one or more different representation approaches, as described in the External dimension of the Process Development service.

Alternatively, this service may be provided by a procedure call interface. Often the specification of process monitoring for a project (an instance of an enactment of a process definition) is independent of (and later than) process development; in these cases, this procedure call alternative is the likely approach for a user (e.g., a process engineer) to specify such binding.

5.4.6 Internal

Process monitoring may be supported by the same mechanisms that support process enactment: basic SEE framework services, tools, and humans. Also, the efficient detection of events that match any of those defined is an interesting part of the internal aspects of this service.

5.4.7 Related Services

The Process Development service is closely related, as it provides the representations that are enacted, including user-specification capabilities corresponding to the semantics of process monitoring supported by the SEE.

Object Management services may be related as process monitoring and event management operations may interact with them, at least for operations upon project objects and sometimes for storage of process definition representations and process enactment state information.

5.4.8 Examples

IPSE2.5 [79], Arcadia [81], EAST [9], ESF [73], EFA [47], Enterprise II [60], COHESION [14], EIS [57], and SLCSE [80] all provide some process monitoring services.

5.5 Process Transaction Service

This service provides 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.

5.5.1 Conceptual

A "transaction" in the general case is a unit of work and a unit of recovery made up from a sequence of atomic operations (and transactions, if nested transactions are allowed). The transaction either succeeds in carrying

out all the specified operations or restores the object(s) upon which it was acting to a state as though the transaction never happened. In the case of processes (as compared to object management), some troublesome differences with traditional concepts of data transactions (see 4.6) have been noted in the literature. The problems are specifically addressed by a number of researchers (e.g., [53], [6], [41], [31]). A conventional data transaction is defined as a sequence of primitive operations that are carried out atomically, i.e., either all the operations succeed and change the database state, or at least one operation fails and the database is restored to the state it was in before the data transaction started. The concept may be extended to a nested set of process transactions. The process transaction idea is needed in an SEE that supports process enactment. For instance, a tool may be thought of as a sequence of either primitive operations or nested tools that must all successfully complete or leave the project's development database (object store) unchanged. But there is a more important situation to model. That is, there is a process that, when appropriate, should be carried out to achieve some development within the SEE. It might not be possible beforehand to prescribe exactly how that development should be carried out in terms of a sequence of operations, but there may be a way to describe when the aim has been achieved. Such processes may well have to be broken down into more manageable ones, again implying a nestable structure. But the timescale of hours, weeks or months for carrying out processes has four implications for a framework's Process Transaction service:

- Work performed by an enacting process should not necessarily be discarded if a failure occurs.
- It is unreasonable to lock the information used by an enacting process for its duration, making it unavailable to other parts of the project for such long periods.
- State information needs to be maintained by long transactions.
- During the enactment of a process or subprocesses, the project's database (object store) may have to go through what would be considered externally as an inconsistent state in order to satisfy a post-condition.

5.5.2 Operations

Operations may:

- create, initiate, abort, delete, and modify transactions
- commit completion of transaction
- checkpoint and rollback of process state
- "login" and "logout" of long-lived transactions

They may also query the identity of what is included in such transactions.

5.5.3 Rules

Possible rules might be prohibitions or restrictions on how process transactions may be nested or redefined dynamically, or on the validity for partial completion.

5.5.4 Types

It is possible that a framework has fine-grained types for representing the linguistic aspects of process definition such as events, process transactions, and monitors.

5.5.5 External

If any formality in process definition (and hence the potential for process enactment) is supported by the SEE, the external dimension provides some manner for users (e.g., process engineers) to represent process definitions including the definition of process transactions, or input them to the SEE. An SEE may support one or more different representation approaches, as described in the External dimension of the Process Development service.

Alternatively, this service may be provided by a procedure call interface.

5.5.6 Internal

Process Transaction operations may be supported by the same mechanisms that support process enactment: basic SEE framework services (especially OM services), tools, and humans.

5.5.7 Related Services

This service may use the operations of the Process Monitoring service. Process Transactions may use OMS features of the State Monitoring and triggering service.

5.5.8 Examples

IPSE2.5 [79], Arcadia [81], EAST [9], ESF [73], EFA [47], Enterprise II [60], COHESION [14], EIS [57], and SLCSE [80] all provide some process transaction services.

5.6 Process Resource Service

Management and allocation of resources is a necessary aspect of enacting a defined process. Process agents (e.g., tools or user roles or individual users) may be assigned to enact various processes and subprocesses, and this is typically done under constraints of time, budget, manpower assignments, equipment suites, and the process definition technology (e.g., insufficient formality may be used for totally automated enactment). These allocations are often provided by services independent of those for process development. This service is typically used by project management and its inputs and results are captured in a process plan.

5.6.1 Conceptual

Process resource management and allocation includes: (a) the definition of process resource types (e.g., user roles, equipment suites, software suites); (b) the registration of resources through the instantiation of process resource types and the assignment of values to the resultant instances; and (c) the interrogation of process resource knowledge by other process and framework services.

Resource definition permits resource types to be defined, including enactment agent types and enactment support mechanisms. For example, enactment agent types include:

- human enactment agent types, such as a project manager, process engineer, or a software engineer, modeled by a "user role model" that may be (partially) mapped to a "resource model"
- machine enactment agent types, such as process program, tool, or shell script, modeled and mapped by a "resource model."

Resource redefinition is required since user classes may be added to and deleted from SEEs, their roles and capabilities may change, and machines and the underlying software are upgraded, replaced, and modified.

In a software development project, the people involved typically have job titles or "user roles" that they are employed to play throughout the project (e.g., as "systems analyst," "technical architect," or "programmer"). In general, the mapping between individuals and roles may be many to many and may (and typically will) change throughout the project's lifespan. The Sub-Environment service also supports handling information about people and roles and the relationships among them. There are more possible relationships than "person X plays role A." For instance, there could be a "person Y is capable of playing role B" relationship that is constrained by attributes such (e.g., experience; age; or ability) of the roles and people. Relationships may exist between roles such as: "all people carrying out quality assurance may carry out the programming role."

Definitions for process resource types could include attributes, operations attached to individual process resource types, and objects required to support their operations.

Resource registration (mapping) permits the instantiation of resources onto defined process resource types, e.g., enactment agent types and enactment support mechanisms. Collectively, a set of process resource types and a project's instances of those types, are sometimes referred to as a "process resource model."

Resource interrogation permits users, processes, and processors to interrogate properties of a process resource model to satisfy the needs of processes being enacted or processors enacting processes.

5.6.2 Operations

Process resource operations might include:

- define, create, modify, and delete a process resource type or instance of a process resource
- register a project's resources as instances of the process resource types
- map enactment agents to processes or subprocesses (this operation may be a specialized variant of a register operation)
- remap (change) the assignment of enactment agents to processes or subprocesses
- interrogate properties of a project's process resource model

5.6.3 Rules

Process Resource Management operations might be constrained by framework access controls and may typically be available only to privileged users. Other constraints might be imposed by institutional policies and doctrines, and by project or customer directives.

Indirectly, these services define constraints on the Process Enactment service.

5.6.4 Types

The models managed by the Process Resource service may be unique types in the framework (potentially aggregates of other types), or they may be basically untyped objects operated upon only by services (or tools) with knowledge of their internal structure.

Some roles may have many instances, e.g., the programmer role. There would therefore probably be role-types, information about their definitions, and rules and operations to do with their instantiation.

5.6.5 External

Typically, simple formalisms will be provided by the SEE for users to specify the resource-related information needed by this service. If any formality in process definition (and hence the potential for process enactment) is

supported by the SEE (as described in the External dimension of the Process Development service), a subset of that formalism or a related language or notation may be used to specify the information needed by this service.

Alternatively, this service may be provided by a procedure call interface.

5.6.6 Internal

One way of implementing most Process Resource operations is by using the Object Management services to model the resource information and the mappings. This enables general tools to be easily provided (or even be available by default) to provide the Process Monitoring service. However, some frameworks implement a Process Resource service independently of Object Management services. This service could be supported by the Sub-Environment service or the Data Subsetting service.

5.6.7 Related Services

Object Management services may be related as the Process Enactment service interacts with them (as described in the Internal dimension).

User roles interact with the Policy Enforcement services (clause 9), especially the Identification and Authentication service, and the Sub-Environment service (10.4). The Process Resource service also is related to (and sometimes even at least partially redundant with) other Framework Administration services (clause 10). It may also be related to the Data Subsetting service (4.20).

5.6.8 Examples

IPSE2.5 [79], ISTAR [25], EAST [9], the Atherton Backplane [67], EFA [47], Enterprise II [60], COHESION [14], EIS [57], and SLCSE [80] have a role database or otherwise provide the Process Resource service.

6 Communication Services

There is a need for explicit communication among the components of an SEE and between components of different SEEs. Tools have to communicate and exchange information with each other and with the set of framework services; framework services need to communicate with each other; and frameworks may require services from other frameworks.

Some of these communication needs are met by services described elsewhere in the model. These are briefly described by the Data Sharing and Interprocess Communication services that follow. However, explicit communication needs also exist in frameworks and are described by the services in this clause.

6.1 Data Sharing Service

Data sharing through the OMS or memory is provided by data manipulation services. This includes access to OMS objects via shared schema of the Metadata service (4.1) or storage and retrieval of the Data Storage service (4.2) and the Data Interchange service (4.21), inheritance via the Function Attachment service (4.14) and various file and communication operations of the Operating System services (clause 7). 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.

6.2 Interprocess Communication Service

Interprocess communication (IPC) (including process invocation (clause 7) and remote procedure call (RPC) (6.3)) is another form and may be provided by operating system (clause 7) or network (6.3) services. The operations for this type of service include connect, disconnect, create channel, and delete channel, as well as send and receive. OS process execution also includes operations such as exec, fork, invoke, and spawn and includes the means to pass appropriate parameters to the executing process as well as means to determine process status after execution.

6.3 Network Service

Most SEEs today consist of collections of processors. Communication among these processors or among OS processes executing on these processors is a vital part of framework design.

6.3.1 Conceptual

The Network service provides for low level communication primitives between collections of computers. The Message service (6.4) provides for communication between any two processes within an SEE. If the processes are executing on different computers, then this Network service must be invoked to cause the actual transfer of data between machines.

6.3.2 Operations

The Network service may include the following operations:

- Operations for establishing simple communication between processes in a network (e.g., Remote Procedure Calls (RPCs)) including name resolution, establishing and destroying connections, data and file transfer, error handling, and connectionless operations such as waits, timeouts and event notification.
- Operations to query network status, control routing of messages through the network, select network protocols, and determine and set capabilities.
- Operations to identify distributed resources and dynamically use distributed resources.
- Operations to manage, monitor and log network usage, alarms, events, load, availability and performance.

6.3.3 Rules

Since no one processor is generally "in charge" and various parts of the network may fail, communication requires explicit protocols for proper sequencing of messages in order to keep the network functioning (e.g., telnet for remote login, smtp for electronic mail).

6.3.4 Types

Communication across a network is generally in the form of messages. Messages are often broken down into smaller sequentially numbered packets, although the packet structure is often hidden from the user of this service.

6.3.5 External

In many networks, the distributed nature of the system is often invisible to the user. Sending messages to tools executing on another machine is the same as sending messages to a tool executing locally. On the other hand, in some networks, explicit network addressing must be part of any network message.

6.3.6 Internal

Long messages are often broken down into smaller packets in order to permit other shorter length messages to pass to avoid jamming the network. Communication can be point-to-point (e.g., "twisted wire" communication) or broadcast (e.g., ethernet or token ring).

6.3.7 Related Services

The Message service may use this service when transmitting across a network. The OMS may be used to map physical devices into named objects and the Distribution and Location service may be used to identify the location of such objects.

6.3.8 Examples

Sockets provide communication paths between processes. GOSIP and TCP/IP are two protocols used by the Internet to perform network communication.

6.4 Message Service

There is a requirement for a service that supports managed communication among a large number of elements of a populated environment framework.

6.4.1 Conceptual

The Message service provides explicit communication through the managed exchange of messages. It supplements the other forms of communication services; for example, tools might share data via messages as well as through the repository. The Message service provides communication over a distributed (in both the logical and physical senses) collection of services and tools. (It does not provide a distribution service per se; for that the reader is referred to the Network services (6.3.)) Although important in the development of SEEs across multiple frameworks, this service is relevant even in an SEE that does not involve distribution or a network. The Message service may be used for two-way inter-tool, inter-framework, tool-to-service, and framework-to-framework communication.

There are three fundamental methods or approaches for exchanging messages:

Point to point. The sender identifies the recipient(s).

Broadcast. The message goes to all tools and framework services.

Multicast. Some selection mechanism identifies the tools and services that receive the message.

These, however, constitute different paradigms for the use of messages and are not separate services.

6.4.2 Operations

The exact set of operations depends on the message protocol or approach being used. Apart from create, delete, and update, there may be send, receive, acknowledge, reply, and ignore.

In the case of a multicast delivery approach, there may also be registration and de-registration operations by which a service indicates in which kinds of messages it is or is not interested. The interest may be expressed in properties of the message itself (e.g., all messages requesting services of the event monitor), in the data contained in the message (e.g., all messages referring to my objects), or various combinations of these.

Error recovery capabilities are typically reflected in the set of operations.

6.4.3 Rules

In the case of multi-cast delivery, it may not be possible to provide a requested service if the tool or framework service has not been registered (10.1).

6.4.4 Types

Messages may be categorized in a number of ways. For example, there may be "request" messages, "notify" messages, and "failure" messages. Each type of message may contain fields for different types of information.

6.4.5 External

The Message service is most likely made available through procedure call interfaces.

There are four primary types of two-way communication which may be provided, any of which may be synchronous or asynchronous:

1. tool to tool,
2. service to service within a single framework,
3. tool to framework, and
4. framework to framework.

The intent where appropriate is to make the requested service available such that it can be requested without reference to the identity of the service provider and it can be provided without reference to the identity of the service requester.

6.4.6 Internal

All of the message approaches can be implemented in very different ways. Depending on the demands of distribution, they may make use of network services. Their implementation also depends on whether the service being offered is synchronous or asynchronous.

6.4.7 Related Services

All services may communicate via the Message service. The information contained in all communications in a given environment framework may depend to a large extent on the models used by the other framework

services. The proper functioning of a multi-cast Message service may be depend on the Registration service (10.1). If the target service to receive the message is realized on another computer within a network of computers, then the Network service may be invoked to enact the communication.

6.4.8 Examples

An example of a Message service would be a discriminating message delivery service which allows control of the destination of messages without the sender having to be aware of desirable recipients.

Some systems that provide various forms of Message services are: ESF [73], ATIS [59], the HP SoftBench Broadcast Message Server (BMS), and the OMG Object Request Broker.

6.5 Event Notification Service

This service supports the delivery of messages based upon certain triggering conditions.

6.5.1 Conceptual

Conceptual integrity of SEE interfaces is enhanced by a consistent format for messages generated by services in an SEE. This service provides a consistent way for various services to report information. This may be in the form of an asynchronous event, a "return code" upon exiting a program, or standard error messages reported to the user.

6.5.2 Operations

The Event Notification service may provide the operations of: post a warning, post an error (of a certain severity), enact event, or inquire as to previously sent messages.

6.5.3 Rules

Messages usually have predefined severity codes.

6.5.4 Types

An event is generally recorded as a number or text with a severity code.

6.5.5 External

This service may appear as a file, as textual output to the user, as a flag which can be interrogated, or as an asynchronous event.

6.5.6 Internal

This may be implemented as either an event requiring operating system services or a default use of specified resources (e.g., a standard error file name).

6.5.7 Related Services

This service may use the asynchronous event notification operation of the Operating System services. It may use the Process Management Enactment service or the Message service to trigger an action. It may also interact with the Presentation or the Dialog services of the user interface.

6.5.8 Examples

The return-code on various operating systems provides a standard way for tools to pass information on to other tools or to the user. The file *stderr* is the standard file for error messages in POSIX [51].

7 Operating System Services

The realization of an SEE requires the implementation of framework services on physical hardware. Accessing the physical resources and providing for the security and integrity of all processes within the SEE is the function of the operating system. There is also a need for services to interact. This may involve communication between two services or in one service accessing the services provided by another.

Since an SEE is usually realized as an extension to an existing operating system, these services describe the functionality generally ascribed to operating systems, although the realization of a service may or may not involve the native operating system.

These services, often realized as the kernel of an operating system, provide the primitive set of services needed for execution of software components in an SEE. This includes the basic operations for reading and writing of files, for managing storage (both real and virtual) of a program, for communicating with other programs and the user, for maintaining security and integrity of the system, and for the general processing of events and scheduling of components to be executed. The set of operating system services provides the base upon which most other framework services are built.

The major object managed by these services is the operating system process. Services provided by an SEE are generally the result of execution of these processes.

7.1 Operating System Process Management

This service provides the ability of an SEE to create processes and to schedule each independently for execution.

7.1.1 Conceptual

The ability of an SEE to create processes and to schedule each independently for execution is a basic functionality upon which many SEE services are based. Each process may execute in its own virtual address space and typically is provided with security against being damaged by another process in the system.

7.1.2 Operations

Operations may: create or spawn; schedule, fork, exec or invoke; destroy; or query low-level operating system processes.

7.1.3 Rules

This service is often hidden from the set of end user services in order to allow for transportability of tools to other frameworks.

7.1.4 Types

The process is the basic structure upon which all execution in an SEE is based.

7.1.5 External

This service is often implemented as a set of procedure calls. The semantics of the procedure calls are often very system specific. For this reason, this set of operations is usually hidden from the set of end user services in order to permit tools to be transportable and to be moved among alternative frameworks.

7.1.6 Internal

Process creation may allow two processes to use the same address space or may require new addresses for the new process. A new process entry is made in the scheduling table for this new process, which now competes for processor time with the creating process.

Each processor executing an SEE executes processes sequentially. Two processes accessing the same data must use various lockout mechanisms to serialize their access to critical sections.

7.1.7 Related Services

This service may interact with the OS Process Support service (4.8) as well as all other operating system services.

7.1.8 Examples

POSIX 1003.1 functions of fork and exec create and execute new processes. Chapters 3.1 and 3.2 of the POSIX 1003.1 Standard provide a summary of process operations ([51]). CAIS-A [30] uses spawn and invoke for these operations.

7.2 Operating System Environment Service

This service provides for passing of information between operating system processes.

7.2.1 Conceptual

Each process may access data previously set by another process. This other process may be unique to an individual user or may be global to the entire SEE. Such information as user security constraints, local time and date, and resource limits are all examples of information needed by processes.

7.2.2 Operations

Operations may set, change, and query various environmental factors, such as operating system process size, priority or identification, time, date, or other environment variables.

7.2.3 Rules

Processes must have sufficient privileges for accessing certain information to avoid security and integrity problems. This service is often hidden from the set of end user services in order to allow for transportability of tools to other frameworks.

7.2.4 Types

Environment variables handle such types as process size, names, time, and dates.

7.2.5 External

This service is often implemented as a set of procedure calls. The semantics of the procedure calls are often very system specific. For this reason, this set of operations is usually hidden from the set of end user services in order to permit tools to be transportable and to be moved among alternative frameworks.

7.2.6 Internal

Implementation of some of these operations often is a simple table lookup of data for efficiency. Network timing functions (i.e., making sure that the time is the same among all frameworks in a distributed SEE) is an extremely difficult task.

7.2.7 Related Services

This may use the Network service in a distributed SEE.

7.2.8 Examples

The getenv function of POSIX provides a mechanism for one process to access information set by another process. POSIX 1003.1 Section 4 provides other examples of environment operations [51]. PCTE provides similar functionality using resources, locks, or pipes [36].

7.3 Operating System Synchronization Service

This service provides for appropriate synchronization of the execution among all operating system processes in an SEE.

7.3.1 Conceptual

The ability for one process to interrogate the status of another process with respect to data access is necessary for implementing multiprocessing systems when multiple processes seemingly access the same data concurrently.

7.3.2 Operations

Operations may create, delete, check, set and reset semaphores, locks, signals and other synchronization objects.

7.3.3 Rules

Operating system processes may be synchronized through the use of segments of code that must be executed mutually exclusively, called critical sections. Executing a critical section within a process usually involves a strict protocol, such as: (1) Only one process may be in a critical section at one time, (2) Any number of read accesses may be allowed as long as no write access may be in a critical section, or (3) Any number of read accesses and only one write access to data within a critical section is allowed. This service is often hidden from the set of end user services in order to allow for transportability of tools to other frameworks.

7.3.4 Types

Locks or semaphores are primitive types. The basic operation is to be able to check the status of a lock or semaphore and set it without being interrupted by another process.

7.3.5 External

This service may have the appearance as a procedure call. It may also be a primitive operation in the hardware of the machine executing the framework. An RPC mechanism with messages may also be used.

7.3.6 Internal

The machine hardware often has a basic hardware instruction like "Test and set" for implementing locks. This service is often related to the use of signals (See Asynchronous Event service.)

7.3.7 Related Services

The Interprocess Communication service is often used for locks implemented in the form of messages.

7.3.8 Examples

The "Test and Set" instruction on IBM mainframe hardware is an example of a locking instruction. CAIS-A [30] "intents" cause locking and synchronization.

7.4 Generalized Input and Output

This service provides basic operations to transfer data between processes and input and output devices attached to an SEE.

7.4.1 Conceptual

The ability to transfer data between a process and some external device is necessary for an SEE to perform some task and either save the results for later use or to report these results to a user of the SEE. This service

provides for this basic operation.

7.4.2 Operations

The basic operations are read and write data. Depending upon the type of device, there may be additional operations such as: ready, online, offline, busy, and start and halt transmission.

7.4.3 Rules

The semantics of each character read or written depend upon the hardware characteristics of the device.

7.4.4 Types

The basic data transferred is the byte or character, sometimes called "raw I/O." This may be interpreted as a character to be printed (on printer or screen), a character to read, pixels for a graphics device, or sound, video or motion commands. The basic hardware device may be, but is not limited to, a keyboard, printer, screen, speaker, facsimile, scanner, disk, tape, or some sensor or control mechanism.

7.4.5 External

This service always operates through some external hardware interface between the SEE and the device.

7.4.6 Internal

Transmission may be asynchronous, character at a time transmission, as with a keyboard, or may be synchronous, block at a time transmission, as with a disk drive. Processes that are performing this service generally need to be locked into memory for transmission of data to succeed.

7.4.7 Related Services

A process often needs the Physical Device service to obtain the hardware address of some logically named device.

7.4.8 Examples

Every operating system includes primitives for performing this service. Sometimes such operations are also provided by programming languages.

7.5 File Storage Service

This service provides the basic operations to read and write files and to store and access them in directories.

7.5.1 Conceptual

This is the basic file system set of operations. Complex object management systems often use this service for basic file system interaction.

7.5.2 Operations

Operations may read, write, query, create, and destroy files in an external storage medium. This also includes operations to manage, create and destroy hierarchical file directories.

7.5.3 Rules

Many systems strongly type files. That is, files have a "type" attribute and may only be accessed if opened with the same type. This service is often hidden from the set of end user services in order to allow for transportability of tools to other frameworks.

7.5.4 Types

There generally are files and directories which contain the names of files. Often files are classified by type (e.g., text files, executable files, indexed files).

7.5.5 External

This service is often implemented as a set of procedure calls. The semantics of the procedure calls are often very system specific. For this reason, this set of operations is usually hidden from the set of end user services in order to permit tools to be transportable and to be moved among alternative frameworks.

7.5.6 Internal

Files are often sequences of characters or sequences of records.

7.5.7 Related Services

OMS services (clause 4) may be implemented using file operations.

7.5.8 Examples

POSIX 1003.1 sections 5 and 6 give examples of file operations [51].

7.6 Asynchronous Event Service

This service provides for the creation and sending of signals between operating system processes.

7.6.1 Conceptual

Processes need the ability to invoke other processes. Signals are one way to provide this mechanism.

7.6.2 Operations

Operations may set, receive and process asynchronous events. This includes external events such as I/O interrupts as well as internal events such as exceptions and program-generated signals. Error handling of unexpected events may also be provided.

7.6.3 Rules

Rules are necessary to accomodate receipt of multiple event signals simultaneously.

7.6.4 Types

Signals provide the basic communication type.

7.6.5 External

This service is often implemented as a set of procedure calls. The semantics of the procedure calls are often very system specific. For this reason, this set of operations is usually hidden from the set of end user services in order to permit tools to be transportable and to be moved among alternative frameworks.

7.6.6 Internal

Signals are closely related to locks and semaphores of the Operating System Synchronization service. A system may often implement one of these mechanisms.

7.6.7 Related Services

This service often uses an RPC or Network service for its implementation.

7.6.8 Examples

POSIX signals in Section 3.3 of the 1003.1 standard are examples of such events [51].

7.7 Interval Timing Service

This service provides for the ability to set and test timers on individual operating system processes.

7.7.1 Conceptual

Processes often schedule interval timers in order to invoke other processes at the conclusion of that time period. This service provides that capability.

7.7.2 Operations

Operations may set, check and cancel interval timers on processes.

7.7.3 Rules

One process often has control over the timed process. This service is often hidden from the set of end user services in order to allow for transportability of tools to other frameworks.

7.7.4 Types

Time periods are often expressed in units of process time (i.e., the time the process uses in the cpu) or real time (i.e., elapsed wall-clock time).

7.7.5 External

This service is often implemented as a set of procedure calls. The semantics of the procedure calls are often very system specific. For this reason, this set of operations is usually hidden from the set of end user services in order to permit tools to be transportable and to be moved among alternative frameworks.

7.7.6 Internal

Timing is often limited by the granularity of the hardware clock (e.g., 50 or 60 hz).

7.7.7 Related Services

This service may also use the Asynchronous Event service to manage the timed process.

7.7.8 Examples

The POSIX operations of alarm and wait are examples of interval timers [51].

7.8 Memory Management Service

This service provides the ability to manage the memory requirements of the SEE.

7.8.1 Conceptual

There may be a need to manage the memory requirements of processes within an SEE. This includes both the physical memory resources of the hardware as well as the logical virtual address space of a process.

7.8.2 Operations

Operations may allocate, free and manage both real and virtual memory. There are also operations to swap processes (and their memory allocation) into and out of secondary storage as processes compete for processor time.

7.8.3 Rules

Depending upon system and hardware design, virtual memory of a process may be larger than physical memory of the hardware.

7.8.4 Types

Memory address space is the major type used by this service. Often it is in terms of fixed length units (e.g., 16 bytes, 1024 bytes).

7.8.5 External

When memory management is not transparent, it is often made available as a set of procedure calls.

7.8.6 Internal

Memory allocation is strongly tied into efficiency constraints. Locking processes into physical memory is necessary at times, but at a cost of flexibility in managing this resource.

7.8.7 Related Services

Swapping of processes into secondary storage may be affected by the File Storage service.

7.8.8 Examples

Malloc in POSIX 1003.1 is the major memory allocation operation [51].

7.9 Physical Device Service

This service provides the ability to manage the physical devices attached to the framework.

7.9.1 Conceptual

There is a need to convert physical devices into logical names used by other services in order to access these devices. Accessing disks, tapes, monitors and serial lines is handled more effectively using logical names rather than physical addresses.

7.9.2 Operations

Operations manage a logical name service to translate between logical device names and physical devices in an environment.

7.9.3 Rules

Rules may be needed to accomodate multiple (interchangeable) physical devices per logical name. There are also likely to be rules regarding how quickly dynamic changes in the availability of physical devices are reflected by this service.

7.9.4 Types

Logical device names are defined by this service.

7.9.5 External

This service is often implemented as a set of procedure calls. The semantics of the procedure calls are often very system specific. For this reason, this set of operations is usually hidden from the set of end user services in order to permit tools to be transportable and to be moved among alternative frameworks.

7.9.6 Internal

For efficiency, internal names are often a pointer to a table of attributes giving the physical address and characteristics of the device.

7.9.7 Related Services

The File Storage service needs this service to translate between a user request for service and accessing the actual physical device. The OMS Name and Location service may be used for doing the actual mapping of device names.

7.9.8 Examples

POSIX uses the mount point /dev as entry points for physical devices, which appear to other processes as special files [51]. Functions like ctermid and ttyname provide mappings between terminal devices and internal file names.

7.10 Operating System Resource Management Service

This service provides the ability to allocate system resources among the various operating system processes in a framework.

7.10.1 Conceptual

There is the need to allocate resources to various processes (and users) of a system. Disk files, tape drives, memory, etc. may be allocated to various users and processes according to various constraints.

7.10.2 Operations

Operations manage the resources of the operating system, including system administration operations to allocate resources to users of the system. Setting and checking quotas on such resources also are involved.

7.10.3 Rules

Resources are typically allocated to single users. Multiple requests for the same resource may be queued or refused. There may also be rules that deal with detecting resource-request deadlock.

7.10.4 Types

Physical attributes, like maximum physical memory allocation, maximum file size, disk or tape drive access, monitor access, and external line access are all types of this service.

7.10.5 External

This service is often implemented as a set of procedure calls. The semantics of the procedure calls are often very system specific. For this reason, this set of operations is usually hidden from the set of end user services in order to permit tools to be transportable and to be moved among alternative frameworks.

7.10.6 Internal

Implementation of resource management must be dynamic, flexible and efficient. The implementation of this service must pay attention to security and integrity concerns of the framework.

7.10.7 Related Services

Operating System Security services are needed to insure appropriate privileges to access and modify resource data. This service implements many of the integrity and security constraints and interacts with those Policy Enforcement services.

7.10.8 Examples

POSIX device operations in section 7 of 1003.1 provide device operations [51].

8 User Interface Services

The User Interface services provide the main conduit for user involvement with the environment, and they provide the major data path between tools and users. Because of the desirability of inter-tool user interface (UI) consistency, a coherent set of UI services may be offered as part of the environment.

While the application program needs to read and write textual information, the user is interacting with the program using windows, menus, mice, sound, video and other modes of communication. Both application program and user interface are often defined independently, and there needs to be some linkage or translation mechanism so that application programs and users can communicate. Since 1982, the most influential user interface reference model has been the Seeheim model (Figure 5), defined at a workshop sponsored by Eurographics and IFIP in Seeheim, Germany [43].

This model defines the run-time user interface as consisting of three components: the application interface, the dialog control, and the presentation component. The application interface provides descriptions of the application data and routines accessible by the user interface. The presentation component is responsible for the physical appearance of the user interface including all device interactions. The dialog control component manages the dialog between the user (via the presentation component) and the application.

It is the dialog control component that provides the translation between the format of the data needed by the application program and the format of the data accessible to the user. For most current systems, the dialog component is processed by the application program with direct invocation of presentation operations; however, with the growth of User Interface Management Systems (UIMSs), there will probably be a growth in systems that implement this intermediate translation service.

Popular window system implementations have tended to honor the Seeheim model more in the breach than the observance. The X Window System from MIT [74], for example, does have separate display and application interface components, but the tokens exchanged via the X Protocol contain only presentation information, and there is no corresponding exchange of dialog tokens between a dialog component and the application. Other window systems have provided dialog control components, but have neglected the interfaces to the application and presentation components. User interface toolkits which are functionally separate from the higher-level presentation component require an elaboration of the Seeheim presentation model. This version of the reference model, for its user interface services, uses the basic Seeheim model, but has extended it in several areas, as the following services describe.

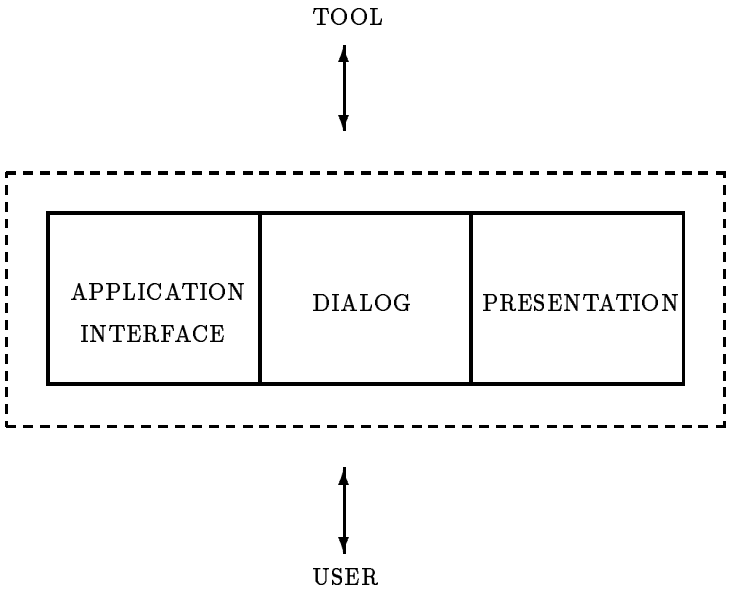


Figure 5: User interface model.

8.1 User Interface Metadata Service

The UI Metadata service provides for defining schemata used by other user interface services. The ability to define windows, menus, and icons and to reference them by name is the function of this service.

8.1.1 Conceptual

The UI Metadata service, like its OMS counterpart (4.1), provides definition, control, and maintenance of the schemata needed to support the user interface. This may include the description of display objects such as windows, menus, icons, and scroll bars.

8.1.2 Operations

Some of the necessary operations are: create, delete, query, and modify. Attributes on these operations control instantiation of these schemata, geometry management (e.g., size and location), resource management (e.g., color), and event management (e.g., sequencing and activation).

8.1.3 Rules

The schemata define the "look and feel" of the interface and are often dependent upon specific style guides for building display objects.

8.1.4 Types

Display object types like menus, windows, icons and scrollbars are defined by this service.

8.1.5 External

These operations may either appear as function calls to other services or as linguistic descriptions of such metadata.

8.1.6 Internal

Long-lived objects, such as the description of window structure and specific user characteristics, may be stored in the OMS using the OMS Metadata service. More active components, such as descriptions of currently active windows, may be stored in more volatile and efficiently accessed storage.

8.1.7 Related Services

This service may use the OMS Metadata service to store persistent objects. The Session service needs to access this metadata in establishing a session. The Presentation service uses this information in actually displaying objects.

8.1.8 Examples

The X Window System intrinsics library and Athena widget set enables application programs to define such presentation schemata [65].

8.2 Session Service

This service provides the functionality needed to initiate and monitor a session between the user and the SEE.

8.2.1 Conceptual

Each time the SEE is entered or invoked, a connection is established between the environment and the user. Depending on the user, the environment and the tools, it may be necessary to register and invoke particular UI facilities.

The UI may be set up in different ways, depending on the user, tool, session, or role. The user may set user preferences such as colors, prefered layout, personalized menus or icons, etc. The system administrator may set up particular collections of tools or repository views for various user roles.

8.2.2 Operations

The set of operations may include login, logout, query, modify a session, register user interface requirements with the session manager, and negotiate user interface requirements with the session manager.

Some operations which may be used to tailor a session for an individual user include: create and delete a user or role profile, copy or modify an existing user or role profile, query a user or role profile, and register a prototype user or role profile.

8.2.3 Rules

The Session service must be invoked before other UI services may be used between a tool and a user using the SEE.

8.2.4 Types

The Session service needs the concept of a user, which may be a separate object or can be a role for some user modelled by the Data Subsetting service (4.20). Each user may have one or more profiles managed by the SEE.

8.2.5 External

Internationalization (e.g., multiple-language character sets, formats for date and time) may be made visible to the user by the external view of the Session service. Formatting procedures may be made available for converting internal data into an appropriate external format.

8.2.6 Internal

Confidentiality and integrity issues are critical with this service in order to prevent unauthorized entry to an SEE.

8.2.7 Related Services

Managing the profiles for an environment is part of the responsibility of Framework Administration. The Session service interacts with the UI Metadata service in determining permitted characteristics for the desired profile. This service interacts with the UI Security service and the Identification and Authorization service for authentication of the user.

Logging and script operations may be related to the Dialog service.

8.2.8 Examples

Upon initialization, the X Window System allows users to format the display screen and invoke applications by putting them in the *.X11Startup* file [74]. The *.Xdefaults* file is used to set default resources for various applications. *Xmodmap* redefines key mappings for various national character sets.

8.3 Text Input Service

This is the service that manages character input for the framework.

8.3.1 Conceptual

This service 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.

8.3.2 Operations

UI Text Input service operations may include: read a character from the keyboard and map keyboard character (or string of characters) to internal sequence of characters. Operations may also remap keyboard keys into alternative sequences of characters. Line editing operations may also be included, such as delete previous character.

8.3.3 Rules

An SEE may have rules regulating maximum line length or the number of characters a user may type ahead of being processed. Some keys may be restricted to certain functions while others may be remapped by the user for various purposes.

8.3.4 Types

Complex data types (e.g., sound, video) may ultimately be reduced to text when processed by this service.

8.3.5 External

This service usually processes keys pressed on the keyboard attached to a terminal or workstation in a framework.

8.3.6 Internal

Some of these operations may be implemented directly by the hardware in the keyboard.

8.3.7 Related Services

Mouse and menu input may result in characters being processed by an application using translation operations of the Display Management service. This service directly communicates with the Dialog service.

8.3.8 Examples

The X Window System program *xmodmap* remaps keyboard characters into new sequences of characters. The stty program in UNIX, on the other hand, remaps how specific characters are to be handled by the operating system. Terminal emulator programs (e.g., kermit on PCs, xterm under the X Window System) provide for emulating a large number of different text input devices.

8.4 Dialog Service

The Dialog service provides the interface between the application program and physical devices. The function of this service is to seamlessly pass information between programs and users.

8.4.1 Conceptual

The Dialog service is the main data interface between the executing application and the user external to the SEE. Although specific implementations of UIs use different methods to exchange data with the user and the environment, some general operations are common to all UIs. A program reads and writes data to the interface, and a user uses a keyboard and mouse for input and views a program on a monitor. Users of an SEE also need the ability to monitor and control the resources over which they have control.

8.4.2 Operations

Dialog service operations may include the following:

- Input and output. Get and put data between the application program and the user interface.
- Translation. Translating data from an application program into a format useful for the user interface, e.g., translation from a value to a representation on a bar graph or dial, and vice versa. Also, a program may have graphical (or textual) output and the display device may only support textual (or graphical) output.
- Redirection. The actual input may be redirected from other sources, e.g., a file. An interface, like the POSIX shell, may provide scripts of text to simulate the keyboard for the application. This also includes the ability to redirect and save input and output streams in order to provide backup capabilities, auditing functions and script generation.
- Data control. Read next parameter from command line, read or write environment parameter, invoke new command interpreter (e.g., shell), and redirect input and output from keyboard and screen to another program (e.g., set a shell program between the application and the actual display devices) or a file.
- Process control. This operation may monitor the application environment (e.g., stop and start programs, query the environment as to what is executing, logged in, etc.) and enact process services to make changes in the environment.

8.4.3 Rules

It is important for the UI to adhere to SEE policy enforcement rules. Prohibiting covert channels and the passing of information among windows not authorized to obtain such information must be enforced.

8.4.4 Types

The service may be able to handle text, pictures, sound and video data, as well as files, physical devices and user profiles.

8.4.5 External

This service defines the interface that most users consider to be "the system." However, such shells actually provide a syntax that is independent of the underlying environment (e.g., POSIX functionality is defined by the kernel functionality, not the shell command language).

8.4.6 Internal

Shell programs may run as separate application programs which may permit different users to use alternative dialog capabilities within the same framework. This permits an application to be built independently from the interface mechanism that will ultimately be employed.

8.4.7 Related Services

This service may use the Presentation service for displaying output data and may use the Text Input service for accessing input data.

8.4.8 Examples

Simple programming read and write commands in languages like C, FORTRAN and Pascal invoke this service. Almost every system has a command line interface, from command.com of MS-DOS to the various UNIX shells (e.g., C, Bourne or Korn shells). User interface management systems like Serpent include complex translations between the output from an application program and what is ultimately displayed on the screen [72].

8.5 Display Management Service

This service provides for interaction among individual windows. It enforces various constraints on the user's interaction with the framework.

8.5.1 Conceptual

In general, each application writes to a specific set of windows. This service provides for the interaction among all such applications and their related windows. This includes interacting with other windows and enforcing integrity constraints between the user and the framework. Passing information between windows is under control of this service.

8.5.2 Operations

The Display Management service may include the following set of operations:

- Data exchange operations for cutting and pasting objects (e.g., text, pictures, graphs, sound) from one window to another window. This also includes translation of mouse and menu input into text.
- Transaction operations to support multiple applications accessing individual windows. Concurrency of UI services and operations in multi-tasking, multi-user environments may also be supported. Handling of window placement and movement may also be supported.
- Enactment operations for attaching various environment functions to user actions. For example, invoking a menu item via mouse or keyboard may cause another service to be enacted.
- State monitoring operations to monitor the state of the environment, e.g., what display objects are visible, what applications control them, and layout of windows on the screen. Operations to set, alter and query the status of UI environment variables.

8.5.3 Rules

The Display Management service enforces interactions between a window and other facets of the display while the Dialog service enforces the interactions between an application and its output windows.

8.5.4 Types

Display Management operations make use of objects such as windows, icons, scrollbars, menus and buttons.

8.5.5 External

These services are typically viewed as mouse and menu input.

8.5.6 Internal

Cut and paste operations are handled by the Display Management service of the UI and may not be part of any application program.

8.5.7 Related Services

This service is closely related to the UI Presentation service and the Dialog service.

8.5.8 Examples

The East UIMS supports a drop and drag mechanism that includes references to any represented PCTE object or link.

8.6 Presentation Service

This service provides capabilities to create and manage the physical interface between the user and the SEE. This includes the screen display area as well as sight, sound, touch or other sensory interfaces in the SEE.

8.6.1 Conceptual

System design has moved from simple character-based terminals to graphical devices managing multiple windows on a display. New technology is enhancing such graphical devices with animation. In addition, other sensory interfaces are emerging, such as real-time video, sound (music and voice) and "ink" (hand-written text) for handling "pen PCs," FAX input and output, and digitized pictures. How to organize, structure and present such information is the function of this service. The physical appearance of the presentation is determined by this service.

8.6.2 Operations

The operations for this service manage, change and delete display objects. This includes operations to create windows, scrollbars, and menus to add text or pictures to windows; to create bitmap display images for video and pictures; to produce phonemes, sound frequencies and tones for sound; and to provide resolution operations for processing hand-drawn or sound input. Operations for voice, video or ink tagging of documents may be provided.

8.6.3 Rules

Rules may govern associations between combined media, such as a sound with an action on the screen.

8.6.4 Types

The primitive types may include: text window for textual output; graphical window for pictures, video and animation; sound icons for aural output; and "ink" images for representing sound or hand-drawn input and output.

8.6.5 External

The objects created by this service may be processed by some output device: screen display items (windows, menus, scrollbars, icons, buttons, video, etc.), sound through a speaker, or digitized pictures for video or FAX.

8.6.6 Internal

Pictures and graphics can be stored as bitmap displays or as vector instructions to a graphics engine to reproduce the picture. Similarly, sound can also be represented as digitized sound waves or as sequences of phonemes to reproduce the sound.

Graphical input may have real-time constraints for processing video or animation objects. Sensory data may often have a resolution greater than the available resolution of the presentation device (e.g., graphical resolution may be greater than the resolution of the bitmap display).

8.6.7 Related Services

The UI Metadata service describes the creation of display objects. Manipulation of these objects between the user interface and the user is the role of the Dialog service. Competition between windows for display area is managed by the Display Management service.

8.6.8 Examples

The Xlib and Xtoolkit features of the X Window system contain examples of this service [65]. Interactive video and video email are starting to make their appearance. N-way conferencing with voice and video among users on several workstations are becoming feasible options.

8.7 UI Security Service

The UI Security service mediates between user views and actions and the security policies enforced by the framework security mechanisms (clause 9). Some of the UI-specific services required are:

- Authentication of the user to the environment, and vice versa
- Trusted Path for communications between the user and the environment
- Representation of the security policies to the user via UI behavior.

This service heavily depends upon the Policy Enforcement Identification and Authentication service (9.2).

8.8 User Interface Name and Location service

This service permits the framework to manage multi-user and multi-platform environments. It permits various sessions to communicate among various tools and various display devices.

8.8.1 Conceptual

In a distributed or multi-user environment, there often is a facility for specifying what, by whom, and where user interface services are being used. With the growth of client-server computing, the service performing the

application may be distinct from the service displaying the results. These may simply be different processes within a single computer or may be processes residing on different nodes of a network.

8.8.2 Operations

Some of the necessary operations are create a name or location reference, delete a name or location reference, query an existing name or location reference, modify an existing name or location reference, register an existing name or location reference, and interpret a name or location reference.

8.8.3 Rules

Names may be relative to certain objects, such as pathnames relative to a root path in a distributed environment. This relates to the namespace concept of the OMS Name service (see 4.4).

8.8.4 Types

Internal names and system-wide name spaces may be required.

8.8.5 External

The concept of surrogates, as in the OMS Name service (4.4), may be used here.

8.8.6 Internal

The translation of a name to a surrogate often requires a table lookup, hash table search, character transformation or a navigation among a complex data structure.

8.8.7 Related Services

UI Name and Location services may depend on the underlying OMS Name and Location services (see 4.4). There may be a need to couple this service with the UI Security service.

8.8.8 Examples

The X Window System provides the ability for servers to execute client applications elsewhere in a network using the surrogate name of the other resource in the network.

The East UI discovers UI data at the PCTE path which is a pathname relative to the local workstation object. However, since every workstation has such an object there may be many current sets of UI data (East configurations) being used simultaneously within a network.

8.9 Internationalization Service

Within the worldwide SEE community, there are various local conventions for accessing computer systems and referring to data. This service provides for those local capabilities.

8.9.1 Conceptual

Often local conventions affect the way data is stored and processed by an SEE. Such issues as character codes, collating sequences, formats for date and time, and other local standards affect input and output data. This service provides for those capabilities that are affected by national conventions and rules.

Some of the relevant issues are [61]:

- Collating Sequences
  - sorting. The position of blank, non-Roman characters, such as Å, Ø, ß, ö, and others are not uniformly defined and may have different interpretations in different countries.
  - case. Some languages like Japanese, Arabic, Hebrew and Thai have no upper-lower case distinction.
  - Scanning direction. Most languages read from left to right, but others exist (e.g., right to left, top to bottom)
- Country-specific date formats. (11/26/92 in U.S. is 26/11/92 in England is 26.11.92 in France is 26-XI-92 in Italy, etc.)
- Country-specific time formats. (e.g., 5:40 p.m. in U.S. is 17:40 in Japan is 17.40 in Germany is 17h40 in France)
- Time zones. While the general rule is one hour of change for each 15 degrees of longitude, it is more a guideline than reality. While time zones are generally an integral number of hours apart, some vary by 15 or 30 minutes. Time changes (e.g., Daylight savings time in U.S. and summer time in Europe) do not occur uniformly around the world. Translating local time into a worldwide standard time is non-trivial. In the southern hemisphere, the transformation for summer time is opposite that of the northern hemisphere.
- Ideographic systems. Some written languages are not based upon a small number of characters forming an alphabet, but instead use large numbers of ideographs (e.g., Japanese, Chinese, Korean).
- Currency. Representation of currency (e.g., \$, £, ¥) varies by country.
- Icons, symbols, and pointer shapes.

8.9.2 Operations

Internationalization operations may include the following:

- Date manipulation operations. There is a need to translate between time zones, 12-hour and 24-hour clocks, summer and standard time, and other local conventions into and out of framework standard time.
- Code-translation operations. While the 8-bit byte and 256-character code are generally sufficient for alphabetically-based languages, there are needs for more extensive coding schemes to handle character-based writing (e.g., Japanese and Chinese Kanji). This operation translates between an external representation of such characters and a common internal representation. There may also be a need for operations for handling collating sequences in such languages.
- Data-translation operations. Various languages provide for specific data formats. There is a need to convert such data into data consistent across framework services. Examples include: representing monetary units in British pounds, U.S. dollars, Japanese Yen; Representing date and time, etc.

8.9.3 Rules

An environment may be localized during installation and configuration. Some environments and tools may allow the administrator to set local conventions on a per-user or per-session basis. Some may allow the user to choose.

8.9.4 Types

External representation of such objects is often outside the control of the framework designer. Date formats, monetary units, language syntax, etc., are often dictated by national laws or conventions.

8.9.5 External

Internationalization objects most often are represented by character strings.

8.9.6 Internal

A framework may often keep one standard format for such objects and translate to the appropriate output format. Different collating sequences can be a problem.

8.9.7 Related Services

Internationalization affects all input and output services. It can affect the Text Input and Dialog services. Many OMS services, like Query and Data Storage, may need this service.

8.9.8 Examples

The UNIX ctime function converts the internal clock into a character string based upon local conventions. The POSIX 1003.1 standard provides for a setlocale function to provide information to the SEE on local standards [51].

8.10 User Assistance Service

This service provides consistent feedback from various SEE components to the user for help and error reporting. User Assistance (UA) (or Help) is an environment-wide, cross-tool concern.

8.10.1 Conceptual

Users of an SEE often need information regarding appropriate use of SEE resources or in the interpretation of SEE error messages. The ability to provide a consistent format for such information greatly aids in using the SEE and in allowing users to move among SEE components easily.

To offer context sensitive help, the tool (or UA service) may need to know where it is being used and what tools are being used with it. To provide a more consistent UI, it is desirable for tools to use common mechanisms to provide UA.

8.10.2 Operations

Some of the necessary operations are register a UA requirement, add a UA facility, remove a UA facility, query an existing UA facility, and modify an existing UA facility.

8.10.3 Rules

This service is often transparent to the service requesting input. The User Assistance service is often enacted independently from any other tools that are executing. That is, User Assistance commands are often processed in parallel to other commands being processed by the user interface.

8.10.4 Types

Types usually come in the form of text associated with a particular command, icon, or function topic.

8.10.5 External

This is often represented as a special menu entry or specific keyboard key to activate.

8.10.6 Internal

Various styles of UI may make various styles of UA necessary. Coordinating them all in a consistent fashion with consistent information may be difficult.

8.10.7 Related Services

Installing UA facilities for the various tools is part of the responsibility of the Framework Administration service.

8.10.8 Examples

The Help facility of many products is an example of this.

9 Policy Enforcement Services

The general purpose of the Policy Enforcement services is to provide support for confidentiality, availability, and integrity in an SEE, for both mandatory and discretionary security. *Confidentiality* is the prevention or monitoring of disclosure of information to unauthorized users. It is sometimes referred to as *access control*, although this term often also implies integrity and availability. *Availability* is the ability to access authorized resources within appropriate time constraints, and being able to prevent denial of service is an appropriate policy enforcement concern. *Integrity* is the prevention of unauthorized or uncontrolled modification of information. A framework may be capable of enforcing policies of confidentiality, availability or integrity.

Security policies divide naturally into two classifications, *mandatory* and *discretionary*. Mandatory rules are those whose access values for objects are created or imposed by a security officer, a person with system-wide authority with regard to the rules. Discretionary rules are those whose access values can be chosen by an individual user to govern access to those things of which that user is said to have ownership.

Policies are stated in terms of the *objects* and the *subjects* of the framework. The objects are the (mostly passive) things to which policies apply. Subjects are the active entities and encompass the complete set of *users*, sometimes collected into *groups* for convenience or named as *roles* that one or more users may take on. Particular users or users in particular roles are called *administrators* or *security officers* because they have the authority to manipulate particular policies. Subjects also include devices, programs, and other computer resources.

The following text draws heavily on the TCSEC (Trusted Computer System Evaluation Criteria) or "Orange Book" [29], as well as the ITSEC (International Trusted System Evaluation Criteria) document [22] produced by the international information security community.

9.1 Security Information Service

The Security Information service is responsible for the establishment of security information for use within the SEE.

9.1.1 Conceptual

The Security Information service 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 SEE. Each such level is often represented by a label.

There is often a need for users to access specific data within a general class of objects for which they do not have security privileges. For example, users need the ability to create files within a file system where they may not have the ability to freely allocate and delete file storage, or they may wish to grant other users access to their files, but they do not have the right to freely change access permissions on files. For these cases, there must be the concept of a "trusted program" which provides higher levels of security functions to users with lower security privileges. This service provides the basis for establishing such programs.

9.1.2 Operations

The operations of this service include:

- define roles and the associations between users and the roles they may assume
- define groups and the associations between users and the groups of which they may be members
- create, read, and change confidentiality and integrity levels of subjects and objects
- test the integrity level of an object.

Creation of objects may provide default security levels without explicit enactment of operations of this service.

9.1.3 Rules

There may be some roles that certain users are not allowed to assume. Rules may govern whether a single user may constitute a group and restrictions on which users can be members of certain groups. Rules govern how labels and their underlying levels are interpreted (e.g., users with "top secret" labels may be able to read information with label "confidential") and the circumstances under which labels and levels may be changed.

9.1.4 Types

One set of types used by this service is those that are necessary for the definition of roles. These may be, for example, the attributes or access rights that delineate the privileges (or restrictions) of those who adopt the role. Another set of types is those that are necessary for the definition of groups. These may be, for example, the attributes or access rights that delineate the privileges (or restrictions) of those in the group. The labels and corresponding levels are other data types of interest to this service. Labels are the basis for the mandatory security policies.

9.1.5 External

The operations provided by this service are often restricted to certain users (most likely to those in roles such as administrator), but assignment of discretionary access control values is normally available to many or all user roles (e.g., through a procedure call interface or a tool). Specifically, the other Policy Enforcement services operate based on information values established or assigned using this service. Labels are most often not visible at the interface to the service. However, there must be an external realization of the operations to change the labels.

9.1.6 Internal

One of the concerns in implementing this service is to ensure that it is itself subject to the restrictions of the Policy Enforcement services. The change label operation must be in the trusted part of a given implementation of this service, thus guaranteeing the integrity of the labels.

9.1.7 Related Services

This service is closely related to the other Policy Enforcement services. It may make use of Object Management services, and the Sub-Environment service (see 10.4) which may restrict access to certain objects. All services involved in mandatory security policies are related to the services involving labels.

9.1.8 Examples

The role capabilities are exemplified by the role services of CAIS-A [30]. The security group services of CAIS-A and POSIX are examples of the security group capabilities of this service. The label capabilities are exemplified by the security interfaces in CAIS-A and PCTE [36]. The *setuid* function of POSIX provides the ability to create trusted programs [51].

9.2 Identification and Authentication Service

The Identification and Authentication service 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.

9.2.1 Conceptual

There is a need for users to make themselves known to the SEE before being granted access to certain information. This service provides for this access authentication.

9.2.2 Operations

Operations for this service may include:

- query identification of user
- validate authentication information (See Rules, below)
- issue certified identity information to user

9.2.3 Rules

Rules for this service may address what information is required to identify a user and what rights are required to alter the authentication information. Authentication can be via something the user knows (e.g., passwords, personal history), something the user is (e.g., signature, fingerprints), or something the user has (e.g., key, identification card).

9.2.4 Types

This service deals with the authentication information, such as labels and permitted roles.

9.2.5 External

This service is most often made available through a login interface.

9.2.6 Internal

One of the concerns of implementing this service is to be sure that it is itself safe from security compromise. Viruses and other malicious software that can make it past the identification and authentication service will have free reign within the SEE.

9.2.7 Related Services

This service is often closely tied to the user interface Session service for establishing a connection between a user at a terminal or workstation and the SEE. This service is dependent upon the Security Information service for the basic information on which its operations are based. All other services that are dependent on being accessible only by authenticated users are indirectly related to the Identification and Authentication service.

9.2.8 Examples

Login services commonly provided by operating systems are an example of this service.

9.3 Mandatory Access Control Service

Mandatory access control policies are those governing the assignment of access values by a security officer to govern access to the information contained in an object.

9.3.1 Conceptual

There is often a need for uniform mandatory rules to be imposed upon all access by users of an SEE. These policies are established by a security officer concerning access to the information contained in an object. The most familiar such policies are those of "government classified information" involving several classification levels ("CONFIDENTIAL," "TOP SECRET," etc.) and "need-to-know" markings ("NOFORN," "EYES-ONLY," etc.).

9.3.2 Operations

The main operation of Mandatory Access Control is to deliver decisions on access control, granting or denying access to some system resource.

9.3.3 Rules

The rules on which the Mandatory Access Control service is based will constrain a subject's access to objects based on various pieces of information. Such rules embody a specific security policy on how to handle the classified information or other resource throughout its lifetime. An example of such a rule is that a subject with a classification label of "confidential" is not permitted to read an object with a higher classification, such as a label of "secret."

9.3.4 Types

Mandatory Access Control rules are based on labels attached to every subject and object in the system, or possibly expressions involving these labels. These particular labels are normally implemented representing the positions in zero or more hierarchies and the names of zero or more non-hierarchical tags. Other important types include the subject's clearance or authentication and the form of access being mediated. The labels representing mandatory access control values of each subject and object should appear as read-only attributes, with only the security officer able to change them.

9.3.5 External

In most secure systems this service is not explicitly invoked. It is a service that is provided by the system as an implicit part of every request for access to some system resource.

9.3.6 Internal

There should be nothing in the definition or implementation of the framework that would preclude the use of it in an environment requiring any level functionality of TCSEC security when it is hosted upon and integrated with a suitable trusted computer base. Note that the subtle nature of some TCSEC security breaches (covert channels, for instance) make this a much more widely influential requirement than it might appear; there must be nothing in the framework interface that provides or supports security "holes" of this nature. Meeting this requirement necessitates a very careful analysis of the entire framework for the security implications of every feature.

It is also important that subjects not be able to circumvent the Mandatory Access Control mechanisms and that (malicious) subjects not be able to starve other (legitimate) subjects through deletion of data, monopolization of processing resources, or other starvation for resources. While prevention of denial of service is an important component of trusted systems, it is not implemented as a distinct service, but rather as part of other services.

9.3.7 Related Services

In an implementation operating under mandatory access control, this service will be related to virtually every service that provides a subject with access to some object, in that it will be implicitly invoked before the requested service is allowed to be performed. This service may utilize features of the Object Management services to access the information it needs to function properly. This service is dependent upon the Security Information service for the basic information on which its operations are based.

9.3.8 Examples

Both CAIS-A [30] and PCTE [36] provide services for mandatory access control. In addition, some operating systems are now starting to be certified as implementations of these services.

9.4 Discretionary Access Control Service

Discretionary Access Control provides the ability to have personal privacy.

9.4.1 Conceptual

Discretionary Access Control gives users the ability to control (i.e., permit and deny) individual modes of access to objects that they own by individual users and all members of sets (groups) of users. Discretionary access control differs from mandatory access control in that it is based on need-to-know, as opposed to mandatory control which is driven by the classification or sensitivity designation of the information or resource.

9.4.2 Operations

As with Mandatory Access Control, the main operation of Discretionary Access Control is to deliver decisions on access control, granting or denying access to some system resource. However, Discretionary Access Control is not dependent on the values of the subject and object labels.

9.4.3 Rules

The rules on which the Discretionary Access Control service is based will constrain a subject's access to objects based on various information values associated with each object. Such values encode an object owner's selective discretionary security decisions on how to protect information and resources that are under his control according to the system's defined security policy. Examples of such rule values may grant all members of some group read-only access to one of the user's files or all users execution access to a file. In the extreme, the rule values could take the form of predicate expressions involving the subject's attributes, evaluating to a boolean value.

9.4.4 Types

Important types for Discretionary Access Control rules include the subject's clearance or authentication and the form of access being mediated. The information representing discretionary access control values of each subject and object should appear as read-only attributes, with only the owner and operating system able to change them.

9.4.5 External

As with Mandatory Access Control, in most systems the Discretionary Access Control service is not explicitly invoked. It is a service that is provided by the system as an implicit part of every request for access to some resource that is under the user's control.

9.4.6 Internal

As with Mandatory Access Control, the rules may normally be applied at the time that a subject initiates a given kind of access to an object. Because this happens very frequently, it is important that the rule checking be as fast as is possible. Also, the storage of rules could cause significant data store overhead if there is no way to group objects with the same rules. It is also important that subjects not be able to circumvent the Discretionary Access Control mechanisms.

9.4.7 Related Services

In an implementation operating under discretionary access control, this service will be related to virtually every service that provides a subject with access to some object, in that it will be implicitly invoked before the requested service is allowed to be performed. This service may utilize features of the object management services to access the information it needs to function properly. This service is dependent upon the Security Information service for the basic information on which its operations are based.

9.4.8 Examples

Discretionary Access Control is a common feature of most modern operating systems. The file permission bits (e.g., referring to self, group and world) are examples of discretionary access control in UNIX. In addition, PCTE [36] and CAIS-A [30] provide discretionary access control features.

9.5 Mandatory Integrity Service

The Mandatory Integrity service protects objects from unauthorized or unconstrained modification as determined by the SEE security officer.

9.5.1 Conceptual

In software development, we can never be completely sure that some part or feature of the product has not been maliciously corrupted. The Mandatory Integrity service provides assurance that an SEE maintains the "purity" or "goodness" of an object by preventing unauthorized or uncontrolled modification.

9.5.2 Operations

The main operation of the Mandatory Integrity service is to deliver decisions on whether some system resource has been corrupted by unauthorized or uncontrolled modification.

9.5.3 Rules

Integrity values of objects are those established by a security officer or other user, according to the system's defined security policy, to permit authorized modifications of objects. In the simplest form, the framework

may provide a means of defining some number of levels of integrity, assigning a level to every subject and object in the system and recording the level value of objects as the "low-water mark" of those of the subjects that have written or modified them. That is, the object's level is compared to that of any subject that is given write access to it and, if higher, is set down to the same as the subject's. Alternatively, the framework could prevent write or modify access of an object with a given integrity level by a subject with a lower integrity level. The service may also record the identifier of a subject that lowers an object's integrity level. Another rule may require that, when an object's integrity level is being recorded and its level is decreased, the user or security officer should be notified.

9.5.4 Types

The integrity level value of each subject and object should appear as read-only attributes, with only the security officer able to change them.

9.5.5 External

As with access control, in most systems the Mandatory Integrity service is not explicitly invoked. It is a service that is provided by the system as an implicit part of every request for access to some object that is under integrity control.

9.5.6 Internal

The most accurate possible implementation of integrity would involve invoking the rules when any kind of WRITE command is issued. This may be difficult from a design point of view, so a more normal implementation would invoke the rules when the object is opened with WRITE access, even though the writing may not really be done and the object not changed.

9.5.7 Related Services

In an implementation operating under integrity control, this service will be related to virtually every service that provides a subject with write or modification access to some object, in that it will be implicitly invoked before the requested service is allowed to be performed. This service may utilize features of the Object Management services to access the information it needs to function properly. This service is dependent upon the Security Information service for the basic information on which its operations are based.

9.5.8 Examples

PCTE [36] provides some Integrity services.

9.6 Discretionary Integrity Service

The Discretionary Integrity service allows users to protect objects from unauthorized or unconstrained modification.

9.6.1 Conceptual

In software development, we can never be completely sure that some part or feature of the product has not been maliciously corrupted. The Discretionary Integrity service provides assurance that a user may determine

that an SEE maintains the "purity" or "goodness" of an object owned by the user by preventing unauthorized or uncontrolled modification.

9.6.2 Operations

The main operation of the Discretionary Integrity service is to deliver decisions on whether some user resource has been corrupted by unauthorized or uncontrolled modification.

9.6.3 Rules

Integrity values of objects are those established by a user, according to that user's own security policy, to permit only authorized modifications of objects the user owns. In the simplest form, the framework may provide a means of defining some number of levels of integrity, assigning a level to every subject and object in the system and recording the level value of objects as the "low-water mark" of those of the subjects that have written or modified them. That is, the object's level is compared to that of any subject that is given write access to it and, if higher, is set down to the same as the subject's. Alternatively, the framework could prevent write or modify access of an object with a given integrity level by a subject with a lower integrity level. The service may also record the identifier of a subject that lowers an object's integrity level. Another rule may require that, when an object's integrity level is being recorded and its level is decreased, the user should be notified.

9.6.4 Types

The integrity level value of each object the user owns should appear as read-only attributes, with only the user (owner) able to change them.

9.6.5 External

As with access control, in most systems the Discretionary Integrity service is not explicitly invoked. It is a service that is provided by the system as an implicit part of every request for access to some object that is under integrity control.

9.6.6 Internal

The most accurate possible implementation of integrity would involve invoking the rules when any kind of WRITE command is issued. This may be difficult from a design point of view, so a more normal implementation would invoke the rules when the object is opened with WRITE access, even though the writing may not really be done and the object not changed.

9.6.7 Related Services

In an implementation operating under integrity control, this service will be related to virtually every service that provides a subject with write or modification access to some object, in that it will be implicitly invoked before the requested service is allowed to be performed. This service may utilize features of the Object Management services to access the information it needs to function properly. This service is dependent upon the Security Information service for the basic information on which its operations are based.

9.6.8 Examples

PCTE [36] provides some Integrity services.

9.7 Secure Object Exportation and Importation Service

A secure SEE must be able to export and import objects in a secure manner.

9.7.1 Conceptual

This service provides for the accurate and interpretable exportation and importation of information between SEEs under constraints imposed by security labels attached to each object exported from and imported to the SEE. It is important that the confidentiality and integrity labels and levels between SEEs be the same and that security policies be compatible.

9.7.2 Operations

Exportation operations make sure that exported labels and levels accurately and unambiguously represent the internal labels and levels, so that the recipient can handle the objects according to its own confidentiality and integrity policies. Importation operations utilize the labeling of import channels as either trusted or not and provide facilities to label each object imported from a non-trusted channel, translating the security labels and integrity levels of each imported object according to its own security policies.

Encryption and decryption of data to pass through non-trusted channels is an important operation of this service.

9.7.3 Rules

There may be rules restricting the set of labels and levels that are allowed to be exported over each export channel or imported from a non-trusted channel. There may also be rules stating that any change to such a set of labels or levels is auditable.

Determining existence of users with appropriate access control and integrity levels on both host and target systems must be part of this service.

9.7.4 Types

The types of most importance to these services are the labels and levels of both the objects and the channels.

9.7.5 External

As with Access Control, in most systems these services are not explicitly invoked. They generally are services that are provided by the system as an implicit part of any attempt to export from or import to a channel, although there may be an external interface for turning such control on and off.

9.7.6 Internal

The labeling of data from a non-trusted channel may be an automatic process using the highest confidentiality level and the lowest integrity level of the channel. Manual upgrading or downgrading of information may be

performed afterwards.

9.7.7 Related Services

The main services that are related to these are the ones that are involved in export and import across external channels. If the labels and levels are stored by the OMS, then those services are also related. This may involve the OMS Data Interchange service.

9.7.8 Examples

The Data Encryption Standard (DES) provides for 64-bit keys to encrypt data.

9.8 Audit Service

It is an essential part of security policies that it be possible to audit the actions that are taken that relate to security.

9.8.1 Conceptual

This service provides the ability to record information about calls of the SEE facilities in order to track and control security-related actions.

9.8.2 Operations

The main operation is to record audit information. It should also be possible to select the data that is to be audited and the conditions under which auditing will be performed.

9.8.3 Rules

It is a likely rule that the change of audit conditions and access to audit data will be limited to those in an authorized role. As an example of possible audit selection conditions, audit might be selectable on the basis of user, terminal or workstation, date and time, type of operation or function, data, or return code, or any logical combination of such criteria.

9.8.4 Types

Almost any information or action in a secure SEE may be subject to audit. Of most interest may be those things that can be related to development activities.

9.8.5 External

As with other security services, once selected, the auditing is not explicitly called each time but works constantly in the background. There will be external interfaces for control of the auditing, however.

9.8.6 Internal

Auditing is itself subject to confidentiality and integrity concerns. It is also likely that the appropriate level of abstraction of the audit will vary with different activities, so that careful choices must be made to make this service effective and useful.

9.8.7 Related Services

All other services that can be audited are related to this one.

9.8.8 Examples

The history file on UNIX provides a trace of all user commands. Various system files track user logins, accessing root (superuser) privileges and other security concerns.

10 Framework Administration Services

There is no doubt that an SEE has to be carefully administered, not least because its precise configuration may be constantly changing to meet the changing needs of the software development enterprise.

It is believed that configuring and administering an environment should require few additional services beyond those already described. Most of the administration and configuration manipulations required should be sufficiently supported by the set of operations within each of the services already described. Clearly some of these operations may only be available to those with system administration privileges. The point made is that most of these operations are provided as part of each of the services.

Examples of the use of framework services are: the management of users, tools, and hardware introduction and maintenance; the control of accounting, backups, and use of subenvironments; and security policies. There may be a particular set of tools, or subsets, defined for system administration or configuration. This is for the most part just a customized use of the framework services.

There are, however, a few specialized Framework Administration services. These include Registration, Resource Management, Metrication, and Sub-Environment.

10.1 Registration Service

The Registration service provides a means for incorporating new tools into an environment based on the framework in such a way that different framework components coordinate effectively with the new tool.

10.1.1 Conceptual

The complexity of a Registration service may vary widely. At one extreme it may consist of simply identifying a communication channel along which messages may pass. At the other extreme it could involve negotiation and agreement on the use of protocols, user interface session, profile and dialog services, data types and formats, and process management enforcement mechanisms.

The Registration service is also responsible for handling the process of tool modification, such as a tool being deleted or archived from the SEE. This may involve notifying (or even requesting permission from) the services and tools making use of the tool that is leaving.

The interaction among various tools to form an integrated whole may require the use of this service. There are two aspects to this registration: making the tool known to other framework or environment services (Object Management, Process Management, Communication, etc.), and having the control functions of these services extend to the operations of the tools. The Registration service deals only with the former, since the latter is handled by other services.

"Tool" registration may also be achieved at a finer granularity than the whole tool; this is particularly true for tools that have several entry points. Although this discussion is in terms of "tools," this should be understood to include individual tool entry points as necessary.

10.1.2 Operations

Example operations for the Registration service include register and unregister. There also needs to be a query operation which would allow a user or administrator to determine the whereabouts and status of a particular tool, as well as tool characteristics.

10.1.3 Rules

It may be a rule that a tool will have to have been installed before it can be registered and cannot be removed until it has been unregistered.

10.1.4 Types

A data structure for the information regarding tools may be necessary.

10.1.5 External

The service may be made available as procedure call interfaces.

10.1.6 Internal

Implementation of tool registration accommodates the registration of tools relying on proprietary mechanisms of the framework and tools supplied by a third-party vendor. Integration of a tool results in notification to the Object Management, Process Management, User Interface Management, or Communication service that a registration is being accomplished. Each such notification provides a different class of capabilities. Disconnecting either category of tool results in the Registration service accomplishing the actions necessary to both remove the tool (unregister) and notify all services and other tools previously aware of the tool.

Tool registration may be both static (e.g., a tool is available for use from the OMS) and dynamic (e.g., each enactment of a tool first requires the compilation of a data object "rule base"). The UIMS has both static and dynamic components.

In the case of registration of individual entry points within a tool, a mapping between entry points and executables may be maintained by this service.

10.1.7 Related Services

The Registration service has relationships with the other major framework components. Notification to the OMS of a tool registration has implications for special privileges enjoyed by tools and special aspects of considerations in accessing tools. There is likely to be a special part of the OMS oriented around the special

category of object represented by "tools." Notification to the Process Management service of a tool registration means that that tool is now available for inclusion in process management considerations and workplans. Notification to the Communication service of a tool registration means that the tool may be located for purposes of exchange of information with other tools and processes within the environment or from outside the environment.

There has to be a model of how a tool or service registers interest in messages. The interest may be expressed in properties of the message itself (e.g., all messages requesting the services of the event monitor), or the interest may be in the data contained in the message (e.g., all messages referring to specific objects), or the interest may be in various combinations of these.

A tool may consist of several subcomponents, each needing its own registration. The Process Enactment service may be used to provide for the invocation of the higher-level tool. The user interface requires notification for enactment of the tool by users.

10.1.8 Examples

HP SoftBench provides an example of this service with respect to the BMS.

10.2 Resource Management Service

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

10.2.1 Conceptual

Framework resources consist of machines, data servers, compute servers, and networks. The Resource Management service is concerned with framework component installation and release, including migration of components, tools, and objects.

10.2.2 Operations

The operations are those necessary for the establishment, manipulation, modification, inquiry, and removal of information bearing on the management of the physical resources of the framework, as well as the physical installation and removal of such resources. Operations may include the ability to run diagnostics in order to determine the proper behavior of the various components (both software and hardware) of the framework.

10.2.3 Rules

It may be a rule that a resource will have to have been installed before it can be registered and cannot be removed until it has been unregistered.

10.2.4 Types

The information managed reflects the physical resources incorporated into the framework and the environment. It may include types describing different physical resource characteristics as well as structures depicting the physical configuration and access paths.

10.2.5 External

This service may be made available through procedure calls, interactive languages, or dialogues, although they are most likely available only to those with framework administration privileges.

10.2.6 Internal

This service may be implemented in a very superficial or very sophisticated way. In either case, it will be important to balance the need for current information (implying a dynamic implementation) with the risk of impeding performance due to the unavailability of a resource that has unexpectedly gone off-line.

10.2.7 Related Services

This service may make use of the OM services to manage its information. It also makes information available to other services which may have a need to know about availability of other framework components.

10.2.8 Examples

The Resource Map of CAIS-A is an example of the kind of information structure required to support this service [30].

10.3 Metrication Service

The purpose of this service is to provide the means to determine the productivity, reliability, and effectiveness of a framework, of the environment built on it, and of the development processes that use that SEE.

10.3.1 Conceptual

This service provides the ability to collect technical measurement information of importance to the administration of the framework. This information consists of two classes of measurements: 1) Process metrics determined by the various users, roles, subenvironments, and products being developed and 2) Product metrics determined from objects in the OMS. The former are needed to evaluate and measure the software development process of an SEE. Since this field is still rather immature, these measurements are hard to define. In that case, a framework often provides for the collection of the latter product metrics. While easier to collect, they often do not indicate the information that is really needed.

A third class of measurements, environmental, are more concerned with operation of the SEE and are managed by the Process Monitoring service. These are the more typical auditing and accounting services.

The Metrication service should be supportive of the evaluation of progress of an individual project and comparison of productivity and quality among projects. Most important, it is necessary to collect such information in order to determine if the framework, as reflected in its support for the process, is improving over time; it provides the basis for optimizations to improve performance and support for environment needs. This differs from the Audit service (see 9.8) whose primary goal is to insure security and usage statistics within the environment.

10.3.2 Operations

This service may include the following operations:

- invoke the data collection service whenever certain events (user command as well as a specific trigger or time period) occur
- report the results of the collected data
- query the data collection data base
- initialize the metrication data

Various graphing, estimating and tabular programs may be built using these basic services.

10.3.3 Rules

Proper data collection first requires an understanding of the complexity model that will be used to define the metrics computed from the collected data.

10.3.4 Types

The metric data collected has specific types used for storage of the information in the OMS, so the report and data collected should be modelled in the type and metadata structure of the OMS. The metric operations requires information on the types of the objects about which metrics are to be collected.

10.3.5 External

Externally, this service may be made available through service calls that are generally used by project management tools.

10.3.6 Internal

The interaction of the collection operation and the enactment of other processes in the environment requires careful synchronization of the OMS in order to preserve the integrity of the data collected.

10.3.7 Related Services

This service is closely related to many of the OMS services since the collected data may reside as objects in the OMS. It is related to the State Monitoring and Triggering service (4.19) since the occurrence of certain events (e.g., updating a module or running a test) could cause the data collection services to be invoked. Object locking from the Concurrency service (4.7) may be used to provide integrity of accessed data. It is related to the Policy Enforcement (clause 9) and Sub-Environment services (10.4) because the extent of the Metrication service needs to be controlled according to security policies and roles. It is related to the Registration service (10.1) since the data collection tools, the conditions under which the tools are to be invoked, and the kind of data to be collected should be included in tool registration information.

10.3.8 Examples

The Goals-Questions-Metrics (GQM) model of Basili [8] provides a mechanism for determining the appropriate metric to collect based upon the needs of the organization.

10.4 Sub-Environment Service

This service provides the capability to divide the SEE into separate sub-environments and to restrict the access of users to only a subset of those SEE resources that are available.

10.4.1 Conceptual

An environment may be considered as a full schema (defined in some data model), together with the set of objects, processes and operations on those objects. The Sub-Environment service deals with ways to access, define, and manipulate a subset of these independently of other objects in the environment. Data or operations may be allowed to be in more than one sub-environment at the same time. This allows creation of "views" of the environment in which a user may have access to only some of the OMS objects, tools, processes and other available facilities.

This service provides the ability to define and add users to an environment, to characterize their modes of operation, to define their roles (including security privileges), and to make available to them the resources they require.

10.4.2 Operations

The following operations may be available:

- define, create and delete users from the framework
- assign and modify roles for each user
- restrict the set of resources that may be made available
- delete the restriction of resources
- change the restriction
- query the framework to determine the set of resources that are available
- determine the current usage of resources and any limitations on using additional resources

10.4.3 Rules

It is necessary for the administrator to have the necessary privileges to perform the administration operations. In particular, in a secure environment it will be necessary for the administrator to have an appropriate security clearance before the operations can be performed. Also, there must be safeguards to ensure that the user is not placed in roles or given access to information for which he or she is not qualified.

10.4.4 Types

This service requires OM structures that identify the user and account for the various relationships (including applicable privileges) of each user with all other related aspects (e.g., other users, objects in the OMS, particular tools) of the environment.

10.4.5 External

This service may be made available externally through a procedure call interface, although it will most likely be restricted to those with appropriate privileges.

10.4.6 Internal

The implementation of this service is likely to be heavily influenced by the nature of the underlying operating system.

10.4.7 Related Services

Other framework services (e.g., Policy Enforcement, Project Management, Data Subsetting) provide some of the services that actually accomplish the granting of rights to the user, establishment of Process Management controls on the user's (future) actions, etc. Virtually all other services are dependent on the results of this service for the determination of actions that are allowed for a particular user.

10.4.8 Examples

SLCSE provides a version of this service. Few systems provide this service explicitly via a tool. However, most operating systems (e.g., VMS) support a script capability which can cover all of the necessary steps completely using a few user-specific parameters.

10.5 Self-Configuration Management Service

This service supports the existence of many simultaneous resident configurations of a framework implementation.

10.5.1 Conceptual

This service provides for the:

- configurations for several different machine types – in a heterogeneous network
- testing of updates of services on one machine before making them globally available
- development and testing of new tools on one machine before making them globally available
- ability to regress to previous configurations of services if new versions prove to be unreliable
- formalization of the modularity of the framework, the delivery of new components, etc.

10.5.2 Operations

Operations may include the following: configure and regress.

10.5.3 Rules

The SEE must emulate all framework and hardware functions in a transparent manner in order to allow multiple frameworks to execute without interfering with each other.

10.5.4 Types

This service must be able to deal with versions and with a variety of simultaneously valid configurations.

10.5.5 External

The operations in this may be provided by procedure call, most likely restricted to those processes having specific authority to perform such configuration management actions.

10.5.6 Internal

The internal aspects of this service will be very specific to a given implementation. They will be highly dependent on the configuration of the underlying machine(s), on the configuration of the network (if any), and on the nature of the cooperation between various projects sharing the overall resource.

10.5.7 Related Services

There are clear relations with OMS and other framework administration services.

10.5.8 Examples

Such self-configuration management of parts of environments is often needed to support UIMS, such as that of EAST [9].

10.6 License Management Service

Software, especially on a network, is often licensed for use by a specific subset of users of the SEE. This service provides this enforcement mechanism.

10.6.1 Conceptual

Licensed programs are often assets which have limitied availability or have monitored use within an SEE. A given tool may be restricted to only specific users, may only be used a certain number of times, may only be used for a specific period of time, or may be monitored for a "charge-per-use." This service provides for managing access to such licensed products.

10.6.2 Operations

The primary operations behave much like locking operations in the OMS. The operations allocate and assign a resource to a given user and free the resource back to the system. Success of the allocation may depend upon the number of other (simultaneous) uses of the resource, the number of previous allocations of the resource or the identity of the user requesting the resource.

A register (and deregister) operation makes the resource known to the licensing service. A query operation or a status operation may inform the user of the current status of the resource, how much it has been used over a given period, and whether it may be accessed at this time.

10.6.3 Rules

Modification of the allocation parameters may be outside of the control of the administrators of the SEE. This is necessary to provide necessary security and preservation of such assets.

10.6.4 Types

Licensing is generally provided on program components, but any "value added" resource may be licensed (e.g., a special tape drive or high quality laser printer).

10.6.5 External

This service is generally invoked by the program itself. The user, by invoking a tool, will either use the tool or get an error message stating that the tool is unavailable.

10.6.6 Internal

The enforcement functionality is often provided by encoded instructions in the program in order to prevent users from disabling the feature.

10.6.7 Related Services

The OMS Access Control service may be used to implement this service. The Registration service may be used by this service to store relevant information on the licensed tool. Encryption using the Secure Object Exportation and Importation service of the Security services may be needed.

Resource monitoring and accounting services may also be used.

10.6.8 Examples

Demonstration copies of software often have time limits after which they cease to work. Software licensed for local area networks (e.g., word processors, spreadsheets) may be installed so that only up to a fixed number of copies may be accessed at one time.

Annex A
(Informative)
BIBLIOGRAPHY

[1] Adariis E. et al. Object Management in a CASE Environment. *Proc. 11th ACM/IEEE International Conference on Software Engineering*, Pittsburgh, PA, May 1989.

[2] AJPO. Evaluation & Validation Reference Manual, version 3.0, 14 February 1991, Ada Joint Program Office, NTIS number AD-A236-697; and Evaluation & Validation Guidebook, version 3.0, 14 February 1991, Ada Joint Program Office NTIS number AD-A236-494.

[3] Andrews T. and C. Harris. Combining language and database advances in the object-oriented development environment. *Proc. Second ACM Conf. on Object Oriented Programming, Systems, Languages, and Applications*, *ACM SIGPLAN Notices* (22)12:430-440, 1987.

[4] ANSI. Interim Report of the ANSI/X3/SPARC Study Group on Data Base Management Systems. ACM SIGFIDET, 7(2):3-139, 1975.

[5] Auroy A. and A. Legait. L'environment de genie logiciel: Entreprise II, *Genie Logiciel et Systemes Experts* No. 22, Mars 91.

[6] Bancilhon F., W. Kim, and H. Korth. A Model of CAD Transactions. *Proc. 11th International Conference on Very Large Data Bases*, 1985.

[7] Barghouti N. S. Supporting Cooperation in the Marvel Process-Centered SDE. *Proc. of the ACM SIGSOFT Fifth Symposium on Software Development Environments*, McLean VA, *ACM SIGSoft Software Engineering Notes* 17(5):21-31, December 1992.

[8] Basili V. R. and H. D. Rombach. The TAME Project: Towards Improvement-Oriented Software Environments. *IEEE Trans. on Software Engineering*, 14(10):758-773, 1988.

[9] Bourguignon J-P. Structuring for Managing Complexity. *Managing Complexity in Software Engineering*, R J Mitchell (Ed.), Peter Peregrinus Ltd., 1990.

[10] Brodie M. L. *On the Development of Data Models*, Springer-Verlag, pp 19-47, 1984.

[11] Brown A. Database Support for Software Engineering, Chapman and Hall, 1990.

[12] Brown A. W. (Ed.). *Integrated Project Support Environments: The Aspect Project*, Academic Press, 1991.

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

[14] Bucken M. Digital's COHESION Taking on AD/Cycle, *Software Magazine* 11(4):71-72, March 1991.

[15] Checkland P. B. *Systems Thinking, Systems Practice*, Wiley and Sons, 1981.

[16] Chen P. P. The Entity-Relationship Model - Toward a Unified View of Data, *ACM Trans. on Database Systems*, 1(1):9-36, 1976.

[17] Clemm G. M. and L. J. Osterweil, A mechanism for environment integration, Department of Computer Science, University of Colorado Technical Report, CU-CS-323-86, 1986.

[18] Clemm G. M. The Workshop System – A Practical Knowledge-Based Software Environment, *Proc. of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, *ACM SIGSoft Software Engineering Notes* 13(5):55-64, November 1988.

[19] Codd E. F. A Relational Model of Data for Large Shared Data Banks, *Comm. of ACM*, 13(6):377-387, 1970.

[20] Codd E. F. Extending the Database Relational Model to Capture More Meaning, *ACM Trans. on Database Systems*, 4(4):397-434, 1979.

[21] Cole R. A Model for Security Standards in Distributed Systems, *Computers and Security* 9(4), June 1990.

[22] Commission of the European Communities. Information Technology System Evaluation Criteria (ITSEC): Provisional Harmonized Criteria, Luxembourg: Office for Official Publications of the European Communities, Version 1.2, 28 June, 1991.

[23] Cronshaw P. The Experimental Aircraft Programme Software Toolset, *Software Engineering Journal*, pp 236-247, November 1986.

[24] *Data Semantics* (DS-1), pp 7-11, January 1985.

[25] Dell P. W. Early Experience with an IPSE, *Software Engineering Journal*, 1(6):259-264, 1986.

[26] Digital Equipment Corporation. ANSI X3H4 Working Draft Information Resource Dictionary System ATIS, Technical Report ANSI X3H4/90-187, ANSI, February 1990.

[27] Dillistone B. R. VCMF – A Version and Configuration Modelling Formalism, *Proc. of Software Engineering 86*, D. Barnes and P. Brown (Ed.), pp 145-163, Peter Peregrinus, 1986.

[28] Dittrich K. R. The DAMOKLES Database System for Design Applications: Its Past, Its Present, and Its Future, pp 151-171. Ellis Horwood, 1989.

[29] DoD. Trusted Computer System Evaluation Criteria, ("Orange Book"), Department of Defense Standard 5200.28-STD, 1985.

[30] DoD. Common Ada Programming Support Environment (APSE) Interface Set (CAIS), MIL-STD-1838A, United States Department of Defense, April 1989.

[31] Dowson M. and B. Nejmeh, Nested Transactions and Visibility Domains, *ACM Workshop on Software CAD Databases*, 1989.

[32] Earl A. N. and R. P. Whittington, Capturing the Semantics of an IPSE Database – Problems, Solutions, and an Example, *Data Processing*, 27(9):33-43, November 1985.

[33] ECMA, Security in Open Systems: A Security Framework, ECMA TR/46, July 1988.

[34] ECMA, Security in Open Systems – Data Elements and Service Definitions, ECMA-138, December 1989.

[35] ECMA, Portable Common Tool Environment (PCTE) Abstract Specification, ECMA-149, December, 1990.

[36] ECMA, Portable Common Tool Environment (PCTE) Abstract Specification, ECMA-149, 2nd Edition, June, 1993.

[37] EIA, CDIF Organization and procedure manual, EIA/PN-2329, January 1990.

[38] Feiler, P.H. and W.S. Humphrey, Software Process Development and Enactment: Concept and Defintions, Software Engineering Institute Report, Carnegie Mellon University, Pittsburgh, PA 15213-3890, U.S.A., March, 1992.

[39] Feldman S. I. Make – a program for maintaining computer programs, *Software Practice and Experience* (9):255-265, 1979.

[40] Fishman D. et al. Iris: An Object-Oriented Database Management System, *ACM Trans. on Office Information Systems*, pp 48-69, January 1987.

[41] Garcia-Molina H. and K. Salem. Sagas, *Proc. ACM SIGMOD*, 1987.

[42] Goldberg A. and D. Robson. Smalltalk-80: The language and its implementation. Addison-Wesley, Reading, MA, 1983.

[43] Green M. The University of Alberta user interface management system, *Computer Graphics* 19(3), pp 205-213.

[44] Hall J. A., P. Hitchcock, and R. Took. An Overview of the ASPECT Architecture, *Integrated Project Support Environments*, J. McDermid (Ed.), pp 86-99, Peter Peregrinus Ltd., 1985.

[45] Hall P., J. Owlett, and S. Todd. *Relations and Entities*, pp 1-20, North Holland, 1976.

[46] Heimbigner D. and S. Krane, A Graph Transform Model for Configuration Management Environments, *ACM Software Engineering Notes* 13(5):217-225, 1988.

[47] Heyes, P. J. An Introduction to the Eurofighter IPSE, *Proc. 4th Int. Conf. on Software Engineering,* Toulouse, December 1991, 527.

[48] Hull R. and R. King, Semantic Database Modeling: Survey, Applications, and Research Issues, *ACM Computing Surveys*, 19(3):201-260, September 1987.

[49] IBM Corporation, *IBM Systems Journal* 29(2), 1990 (Entire issue devoted to AD/cycle).

[50] IEEE, A reference model for computing system tool interconnections (draft), P1175/D11, *IEEE Computer Society's Task Force on Professional Computing Tools*, May 1991.

[51] IEEE, IEEE Standard Portable Operating System Interface for Computer Environments, IEEE Standard 1003.1-1988.

[52] Kadia R. Issues Encountered in Building a Flexible Software Development Environment, *Proc. of the ACM SIGSOFT Fifth Symposium on Software Development Environments*, McLean VA, *ACM SIGSoft Software Engineering Notes* 17(5):169-180, December 1992.

[53] Kaiser G. E. A Flexible Transaction Model for Software Engineering, Technical Report, Computer Science, Columbia University, June 1989.

[54] Karrer, A, and M. Penedo, A Survey of Alternative SEE Architectural Approaches, Technical Report, Arcadia-TRW-90-004, TRW, December 1990.

[55] Kelter U. Supporting Fine-Grained Data in PCTE, ECMA/TC33/91/36, 1991.

[56] Klahold P., G. Schlageter, and W. Wilkes. A General Model for Version Management in Databases, *Proc. 12th International Conference on Very Large Data Bases*, pp 319-327, Kyoto, Japan, August 1986.

[57] Krueger J., T. King, J. Ward, and T. Peterson. Using Frameworks as a Foundation for Integrating Tools and Data, *Proc. 3rd National Symposium on Concurrent Engineering,* Washington DC, 1991, Sponsored by the Concurrent Engineering Research Center, Morgantown, VA.

[58] Leblang D. B. and R. P. Chase, Computer-Aided Software Engineering in a Distributed Workstation Environment, *ACM SIGPLAN Notices*, pp 104-112, May 1984.

[59] Leblang D. and D. Hare, CIS 89-008 Draft Proposal: Tool Integration Environment Program Interface, 6 June 1989.

[60] Legait A, Entreprise II: An Integrated and Open Software Engineering Environment, *Proc. of Milcomp 90.*

[61] Martin A. Internationalization Explored, Uniforum, 1992.

[62] Meier A. and R. A. Lorie, A Surrogate Concept for Engineering Databases, *Proc. International Conference on Very Large Data Bases*, 1983.

[63] Mylopoulos J. and H. Wong, Some Features of the Taxis Data Model, *Proc. 6th International Conference on Very Large Data Bases*, pp 399-410, ACM, 1980.

[64] NIST, Information Resource Dictionary System (IRDS), Federal Information Processing Standard 156, April 1989, (Also ANSI X3.138-1988).

[65] O'Reilly and Assoc, The X Window System. O'Reilly and Associates, Sebastopol, CA, 1990.

[66] Osterweil L. Software Processes Are Software Too, *Proc. 9th ACM/IEEE International Conference on Software Engineering*, pp 1-13, Monterey, CA, March 1987.

[67] Paseman, W. Architecture of an Integration and Portability Platform, *IEEE Compcon*, San Francisco CA, IEEE Catalog Number 88CH2539-5, ISBN 0-8186-0828-5 pp 254-258, 1988.

[68] Peckham J. and F. Maryanski, Semantic Data Models, *ACM Computing Surveys*, 20(2):153-189, September 1988.

[69] Penedo M. H. and E. D. Stuckle, PMDB – A Project Master Database for Software Engineering Environments, *Proc. 8th ACM/IEEE International Conference on Software Engineering*, London, England, August 1985.

[70] PSESWG, Reference Model for Project Support Environments, Version 1, NAWCADWAR-93023-70, February, 1993.

[71] Purtilo J. The Polylith Software Bus, *ACM Trans. on Prog. Lang. and Systems*, 1991.

[72] Seacord R. User interface management systems and application portability, *IEEE Computer* 23(10), 1990.

[73] Shafer W. and H. Weber, The ESF-profile, *Handbook of Computer Aided Software Engineering*, Van Nostrand, September 1988.

[74] Scheifler R. W. and J. Gettys, The X Window System, *ACM Trans. on Graphics* 5(2):79-109, 1986.

[75] Simmonds I. C development on PCTE, *International Workshop on Unix-Based Software Development Environments*, Dallas, January 1991.

[76] Simmons G. L. What is Software Engineering? Technical Report ISBN 0-85012-612-6, NCC Publications, 1987.

[77] Smith J. M. and D. C. P. Smith, Database Abstractions: Aggregation and Generalization. *ACM Trans. on Database Systems*, 2(2), June 1977.

[78] Snodgrass R. and K. Shannon, Fine Grained Data Management To Achieve Evolution Resilience in a Software Development Environment, *Proc. 4th ACM SIGSOFT Symposium on Software Development Environments*, Irvine, CA, December, 1990, 144–156.

[79] Snowdon R. A. A brief overview of the IPSE2.5 Project, *Ada User* 9(4):156-161, 1988.

[80] Strelich T. The Software Life Cycle Support Environment (SLCSE) – A Computer-Based Framework for Developing Software Systems, *Proc. of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, *ACM SIGSoft Software Engineering Notes* 13(5), November 1988.

[81] Sutton S. M., D. Heimbigner, L. J. Osterweil Language constructs for managing change in process-centered environments, *Proc. of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Software Development Environments*, *ACM SIGSoft Software Engineering Notes* 15(6):206-217, December 1990.

[82] Syntagma, PACT Tool Writer's Guide, Technical report, Syntagma, April 1988.

[83] Taylor R. N. et al. Foundations for the Arcadia Environment Architecture, *Proc. of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Software Development Environments,* November 1988.

[84] Tedjini M., I. Thomas, G. Benoliel, F. Gallo. A query service for a software engineering database system, *ACM SIGSoft* 4^{th} *Symp. on Software Development Environments*, Irvine, CA, December 1990.

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

[86] Tsichritzis D. C. and F. H. Lochovsky, *Data Models*, Prentice-Hall, 1982.

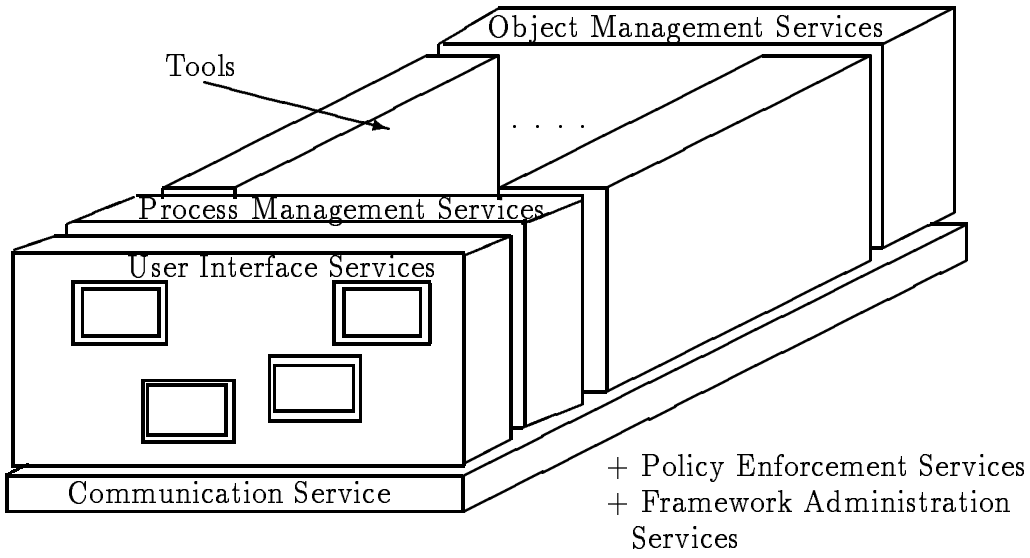
[87] Warboys B. The IPSE 2.5 Project: Process Modelling as the Basis for a Support Environment, *Proc. of Software Development Environments and Factories*, Berlin, May 1989.

[88] Wasserman A. I. Tool integration in software engineering environment, *Software Engineering Environments,* F. Long (Ed.), Lecture Notes in Computer Science 467, pp 137-149, Springer Verlag, 1989.

[89] Zelkowitz M. V. Use of an environment classification model, *Proc. ACM/IEEE* 15^{th} *International Conf. on Soft. Eng.*, pp 348-357, Baltimore, MD, May, 1993.

Annex B
(Informative)
REFERENCE MODEL STRUCTURE

It may be noticed that for this third edition of the environment frameworks reference model in 2.2 there is no figure giving the structure of the model. The following was used previously and was deliberately deleted in this new edition.



This figure does not represent the structure of the frameworks reference model

The reference model[4] is a catalog of services that typically represent environment frameworks. It is not the intent of this model to describe any architecture for this framework, and specifically, not to enforce *good* or *bad* framework designs. The goal is to be able to describe any existing and, it is hoped, any future framework.

However, the previously used figure has often been misinterpreted as an architecture for an environment, and technical papers and product announcements have been written describing products as meeting the "OMS interface," the "Policy enforcement interface," or other such interfaces of the frameworks reference model.

This is clearly a misuse of the figure. The groupings and the visual image were only intended as a way to remember the service categories in the model. How one designs an architecture for implementing some of the framework services is not part of the model and is outside of the scope of this document.

[4] **The figure shown is NOT the frameworks reference model. THIS REPORT is the model.**