E C M A

EUROPEAN COMPUTER MANUFACTURERS ASSOCIATION

REFERENCE MODEL FOR FRAMEWORKS OF SOFTWARE ENGINEERING ENVIRONMENTS

ECMA TR/55

NIST Special Publication 500-201

This publication is a revision of Technical Report ECMA TR/55, published in December 1990, and has been prepared jointly by the ECMA/TC33 Task Group on the Reference Model and the ISEE Working Group of the National Institute of Standards and Technology (NIST) of the United States Department of Commerce. This report is also published as NIST Special Publication 500-201.

2nd Edition December 1991

Brief History

Work in a reference model for Software Engineering Environments (SEEs) has been in progress for the past years both in the United States and Europe. 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 (TC33/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. The reference model was adopted as an ECMA technical report [33] at the ECMA General Assembly of 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 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.

A minor extension to the ECMA RM was used by NIST ISEE to map several existing products to the RM in order to test the effectiveness and coverage of the services specified (CAIS-A [27], ECMA PCTE [32], SLCSE [67], and AD/cycle [44]).

This current document represents the results of discussions held at various NIST workshops and comments provided by the participants of these workshops, as well as a review by the members of TC33/TGRM. This report is expected to undergo further changes and an update will be published jointly as a NIST and ECMA report during 1992.

The current RM is still focused on SEE frameworks; however, it is expected that future versions of this model will be enhanced in support of a full SEE RM.

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 December 1991.

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[64]. *Software Engineering Environment* means the system[13] 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, denoted *Tools* (see 8), and a set of (relatively) fixed infrastructure capabilities which provide support for processes, objects, or user interfaces, denoted *SEE Frameworks*. 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., 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, secretaries use the customized environment to build target systems according to the customers' software requirements.

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 definers* 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). Its main purposes are to describe existing SEEs using the common reference terms and structures provided by the RM and to provide a basis for determining interfaces between environment components. Two of the objectives of the identification of the SEE frameworks RM are to support the identification of areas in SEE architectures for developing or improving standards and to provide a common reference for describing existing standards.

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.

Standard Identification and Relationship

- The RM 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 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 damaging 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	Software 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
DSEE	Domain Software Engineering Environment
EAST	European Advanced Software Technology
ECMA	European Computer Manufacturers Association
E-R	Entity Relationship (data model)
ESF	EUREKA Software Factory
I/O	Input/Output
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
LID	Logical Identifier
MIL	Module Interconnection Language
MIT	Massachusetts Institute of Technology
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	Project Data Exchange with STEP (Standard for Exchange of Product Model Data)
PHIGS	Programmer's Hierarchical Interactive Graphical System
PMDB	Project Master Data Base
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 Security Evaluation Criteria
TGRM	(ECMA) Task Group on the Reference Model
UA	User Assistance
UI	User Interface
UIMS	User Interface Management System

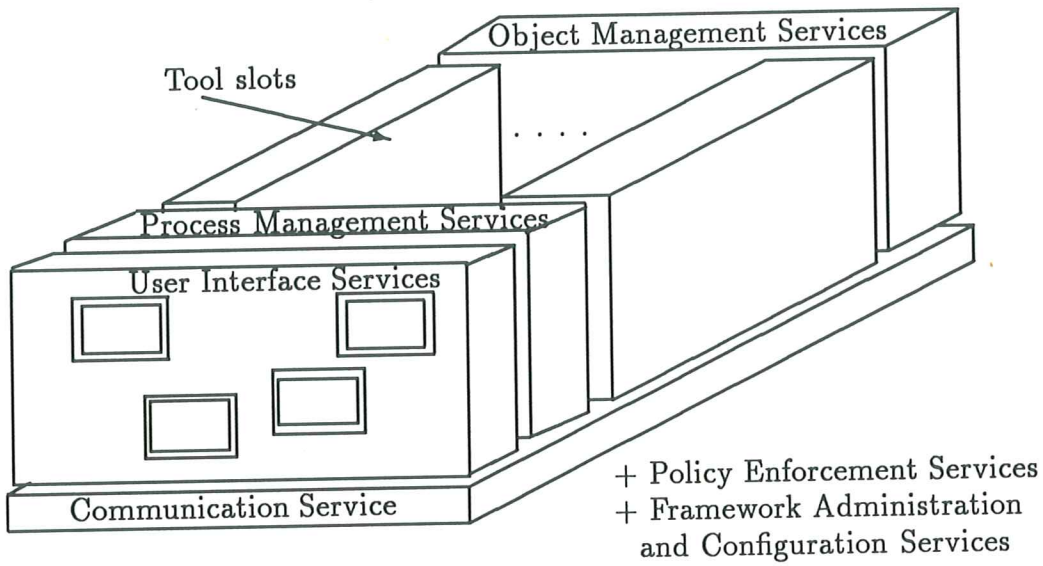

Figure 1: Reference model organization.

2 SEE Frameworks Reference Model

The reference model for SEE frameworks divides an environment framework into 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 SEE Reference Model Structure

The SEE framework reference model consists of a set of services which correspond to SEE framework functional operational capabilities. For clarity and manageability purposes, these services are grouped together. Those groupings correspond to key characteristics of SEEs: the objects that are manipulated by the SEE (data), the tasks that an SEE must perform (processes), interactions with the user (user interface), integration of the various components together, and administration of the entire SEE.

For SEE frameworks, the grouping that has become convenient is represented by figure 1. With such groupings, the important interfaces between existing and forthcoming standards may be identified. Some of these groupings also enable various kinds of integration to be discussed: presentation integration (user interface services); control integration (process management services plus communication service); and data integration (data integration services).

It is very important to realize that the diagram must not be interpreted as a set of layers (see 2.2 for a discussion of the relationships between the services). The tool slots reserve a place for extending the facilities provided by the environment framework with integrated tool sets. The tools (or services) that extend the facilities may vary across different incarnations, as database schema and process programs may be different. The presentation aspects of a tool are deliberately separated from its semantic behavior as, in general, these are best discussed individually. The classification of tools has not been the primary focus of the reference model, but it is a natural extension of the work.

The purpose of grouping the services is to view the RM from perspectives which are closer to users' understandings of SEEs and to support the conjecture that they may help the identification of candidate standards. The groupings of services are briefly described in this clause with complete descriptions later in this document.

2.1.1 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.

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

Data Storage and Persistence Service The Data Storage Service provides definition, control, and maintenance of objects, typically according to previously defined schemas and type definitions.

Relationship Service The 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 or it may be a separate service.

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

Distribution and Location Service This service provides capabilities that support management and access of distributed objects.

Data Transaction Service This service provides capabilities to define and enact transactions.

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.

Archive Service The Archive Service allows on-line information to be transferred to off-line media and vice-versa.

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

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

Common and Canonical Schema Service This service provides mechanisms for integrating tools into an SEE by providing a means to create common (logical) definitions of the objects (and operations) these tools may share from the underlying objects in the OMS.

Version Service This service provides capabilities for managing data from earlier states of objects in the OMS. 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 The Query 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 The State Monitoring and Triggering Service enables the specification and enaction of database states, state transformations, and actions to be taken should these states occur or persist.

Sub-Environment (Views) Service The Sub-Environment 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).

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

2.1.2 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 lifecycles. 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 Definition Service It is expected that an organization may have a library (repository) of process assets, each of which may be a complete process, a (sub)process (or process element), or a process architecture. An SEE may provide facilities to define new process assets.

Process Enactment Service A process definition may be *enacted* by process agents that may be humans or machines.

Process Visibility and Scoping Service In general, several enacting process elements may cooperate to achieve the goals of a larger process. Logically, the extent of such cooperation is part of the definition of processes and may be provided by integrated visibility and scoping features with the process definition service.

Process State Service During enactment, a process has an "enactment state" that changes. Certain changes in the enactment state of a process may be defined as "events" and may act as conditions or constraints affecting other processes.

Process Control Service A process being enacted by an SEE may be recorded, measured, controlled, managed, or constrained.

Process Resource Management Service Process agents (e.g., tools or user roles or individual users) may be assigned to enact various processes and process elements, and this is typically done under constraints of time, budget, manpower assignments, equipment suites, and process definition technology (e.g., the formality or completeness of the installed process description language may be insufficiently unambiguous for totally automated enactment).

2.1.3 Communication Service

This service (clause 6) provides a standard communication mechanism which may be used for inter-tool and inter-service communication. The services depend upon the form of communication mechanism provided: messages, process invocation and remote procedure call, or data sharing.

2.1.4 User Interface Service

The subject of user interfaces is an extremely complex issue which is far more general than integration frameworks. Nevertheless, a consistent User Interface Service may be adopted for a complete framework. The importance of separating the presentation of functionality from its provision is noted in clause 7. The set of services is:

User Interface Metadata Service This service provides for describing the objects used by the User Interface Services.

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

Security Service This service provides the security constraints needed by the UI.

Profile Service This service provides the tool-to-session transformations needed to run multiple tools on multiple UI devices.

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 with various tools and various display devices.

Application Interface Service This service provides most of the data transfer capabilities into and out of the tools and environment to the end user.

Dialog Service This service provides for integrity constraints between the user and the framework.

Presentation Service This service provides for low-level manipulation of display devices by the user interface.

Internationalization Service This service provides capabilities concerned with different national interests.

User Assistance Service This service provides a consistent feedback from various tools to the user for help and error reporting.

2.1.5 Tool Services

A tool (clause 8) is any software that uses the services provided by the framework to support a particular application. The services described (in clause 8) are:

Tool Service This service provides any of the additional functionality needed to support any application, such as editing, testing, compiling, and analyzing.

2.1.6 Policy Enforcement Services

The reference model uses the term "policy enforcement" to cover the similar functionality of security enforcement, integrity monitoring, and various object management functions such as configuration management. The SEE reference model regards security as a service that crosses many of the boundaries of the reference model divisions. It is described in clause 9. The set of services is:

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

Discretionary Confidentiality Service Discretionary confidentiality policies are those established by a user concerning access to the information contained in an object and becomes largely a matter of personal privacy.

Mandatory Integrity Service Integrity provides assurance that a system object maintains (or at least tracks) the "purity" or "goodness" of an object by recording exactly what has been done to the object and how it was done.

Discretionary Integrity Service Discretionary integrity controls are implemented by all write, modify, and append permission functions defined for discretionary access controls.

Mandatory Conformity Service Conformity policies are the result of automation of operational models.

Discretionary Conformity Service Individual users would use conformity enforcement to structure their own work environment. Under the right conditions, it could turn out to be the equivalent of "canned procedures" or "command scripts."

2.1.7 Framework Administration and Configuration 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:

Tool Registration Service The Tool 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.

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

Metrication Service This service provides the ability to collect technical measurement information of importance to the administration of the framework.

User Administration Service This service provides the ability to add users to an environment, to characterize their modes of operation and roles (including security privileges), and to make available to them the resources which they require.

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

2.2 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 are 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.

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,

[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.*

[2] *The terms are borrowed from the database world (the IRDS work [54] is a reasonable source), but they are used more generally here.*

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. Once again, 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 RM, each clause corresponding to a service grouping. For each service, descriptions about the dimensions described above are provided. 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.

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 [32] and CAIS-A [27]. 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 lifecycle 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 schema 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 and Canonical Schema (see 4.15) are examples of services that provide integration mechanisms.
- **It is a different dimension of the services.** Having common services allows for but does not force integration. Having common OMS and schema permits but does not force the tool builder's use of it. Thus for any individual environment framework, the degree to which these common services 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[69]).

3.1.2 Integration Mechanisms

As defined above, integration is the relationship among (several) components in an SEE. This 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 [60]). 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?

Using a figure developed by Wasserman[75], we may represent several of the integration areas as discrete points bounded by three integration axes (see figure 2). 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* (software development in the context of this document). This view is gaining support because software

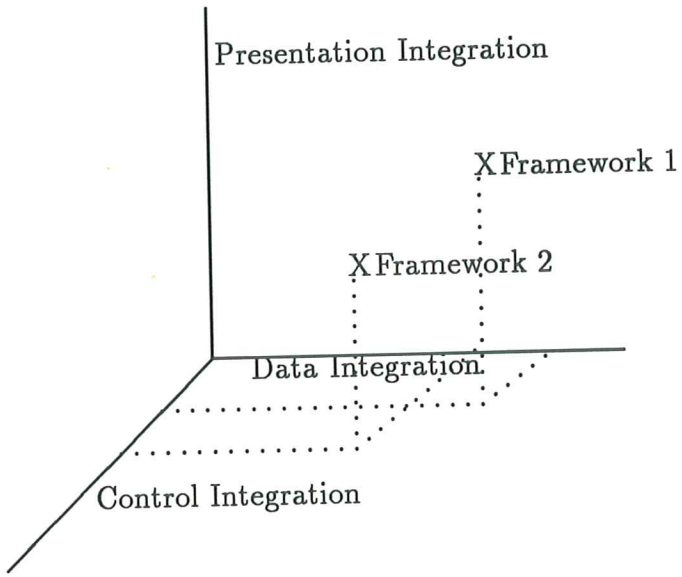

Figure 2: Integration mechanisms.

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 lifecycle-wide completeness) may be decomposed into processes which span activities less than an entire lifecycle. While it is proper to use *process* to also refer to these decomposed parts of complete lifecycle processes, the terms *process element*, *process fragment*, *process step*, and *subprocess* are often used to clarify usage from lifecycle 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 [35]. 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 lifecycle process or process elements and, in particular, the interaction between process users (project members) and a SEE. The terms *process*

programming [56] 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 lifecycles very analogous to software lifecycles:

- 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 architectures may be organized into libraries and be reused.

A SEE could provide services for process development lifecycles and software development lifecycles, 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 definition) by the same facilities that fulfill other services in this reference model. Object management facilities are prime candidates for this duality because process definitions 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 a 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 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. Those services may be specific or general in its support for 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 [9] [21] [73] [58] [43]. Brown[10] specifically addresses database support for software engineering.

The Metadata Services are 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.

The Metadata Service provides control and maintenance of metadata. The study of metadata and its maintenance is ongoing [55]. 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 [37]), 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 Metadata Services use the file system to store information about schemas; however, in other cases they 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 Sub-Environment Service (4.20). In particular, since the structure of the objects in an environment evolves, versions of metadata may be present.

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 [72] [32] and CAIS-A [27]; the E-R model of Aspect [40] [29] [11]; the class hierarchy of ATIS [23]; and elements of the following models and systems: Damokles [25], the relational model [17], Taxis [53], SDM(+) [65], and Iris [37].

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.

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

To be discussed in a future edition of this publication.

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 [59] and SLCSE [67] have shown that there exists 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. For example, the entity relationship model [14] treats objects and relationships separately while RM/T [18] treats relationships as objects.

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

4.3.2 Operations

If relationships are part of the data model, then one should use the same operations provided to the data model. Typical relationship operations may be: create, update, and delete relationship types and instances.

When relationships are themselves first-class objects, then "bulk data" operations 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 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., ACTIS [50]).

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

Here are typical candidates for relationships:

1. There is a particular dependency between types of objects, e.g., an object code is compiled from source code.
2. A recorded relationship exists between two or more objects, e.g., members of a project or author of a document.
3. A constructional relationship is useful to record, e.g., parts of a design or test cases which test certain code.

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 [41], 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 [52].

4.4.6 Internal

The implementation of a Name Service has to take care that it 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 [32].

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

To be discussed in a future edition of this publication.

4.5 Distribution and Location Service

This service provides capabilities which support management and access of distributed objects. 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. The essence of the problem to be solved is to enable all data (e.g., objects, resources, processors) and possibly services of the SEE framework to be available over a distributed collection of (potentially heterogeneous) 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 at 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

To be discussed in a future edition of this publication.

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

To be discussed in a future edition of this publication.

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 (see 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

To be discussed in a future edition of this publication.

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" comments. 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

To be discussed in a future edition of this publication.

4.6.4 Types

To be discussed in a future edition of this publication.

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 Sub-Environment Service (4.20), and the Process State Service (5.4).

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 sub-environments do not need many of the services described here. This topic is very much a research issue.

4.6.8 Examples

To be discussed in a future edition of this publication.

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. The 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 4.20) 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

To be discussed in a future edition of this publication.

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 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's 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

To be discussed in a future edition of this publication.

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

To be discussed in a future edition of this publication.

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 [27] and PCTE [32] are examples of this.

4.9 Archive Service

The Archive Service allows on-line information to be transferred to an off-line medium 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.

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 is 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 be discussed in a future edition of this publication.

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

To be discussed in a future edition of this publication.

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

To be discussed in a future edition of this publication.

4.10.3 Rules

To be discussed in a future edition of this publication.

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 (5.4) 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

To be discussed in a future edition of this publication.

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 [36] 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 [36]) 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 [68] 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

To be discussed in a future edition of this publication.

4.11.4 Types

To be discussed in a future edition of this publication.

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

To be discussed in a future edition of this publication.

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 [15], EAST [63] [71] and its graphical builder process, derived relations in APPL/A [68], Unix Make tool [36].

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, Sub-Environment 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

To be discussed in a future edition of this publication.

4.12.4 Types

To be discussed in a future edition of this publication.

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

To be discussed in a future edition of this publication.

4.12.7 Related Services

Services related to the Replication and Synchronization Service are the Location Service, the Version Service, the Sub-Environment Service, the Concurrency Service, and the Access Control and Security Service.

4.12.8 Examples

To be discussed in a future edition of this publication.

4.13 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.

4.13.1 Conceptual

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, object or relationship type, and instance attribute) and may be based on multiple criteria (e.g., user identification, user role, current tool, current project phase, heuristics).

4.13.2 Operations

Examples of operations provided by this service are: 1) operations supporting creation and modification of access control specifications (e.g., access control lists based on user name, role), and 2) operations supporting access control enforcement at runtime (i.e., dynamically granting or denying users or tools access to objects based on the pre-defined access control specifications).

4.13.3 Rules

To be discussed in a future edition of this publication.

4.13.4 Types

To be discussed in a future edition of this publication.

4.13.5 External

Of the two examples of operations provided by an access control service (i.e., specification and run-time support), the specification operations are generally visible to project users (usually a project manager or other authorized personnel). The enforcement operations are usually embedded in the OMS or its underlying implementation (e.g., access control lists for a relational DBMS, standard file protection mechanisms for a hierarchical file system) and are generally not visible to users. Access control enforcement is a dynamic capability exercised at runtime whenever a user or tool tries to access data objects. Access control specification may be both static and dynamic (i.e., access controls are specified before creation and population of an object base, but may be changed on the fly as users, roles, and other project characteristics change over time).

4.13.6 Internal

To be discussed in a future edition of this publication.

4.13.7 Related Services

The services related to access control may include: Data Storage, Communication Service, Metadata, Canonical Schema, and the State Monitoring and Triggering Services. This service is related to the confidentiality, integrity, and conformity services of the policy enforcement services (see clause 9).

4.13.8 Examples

To be discussed in a future edition of this publication.

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, 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 encapsulated "foreign" tools (ones not written to directly use OMS objects), the capsule itself may become part of the semantics 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

To be discussed in a future edition of this publication.

4.14.4 Types

To be discussed in a future edition of this publication.

4.14.5 External

If the attachment is inherent or supported by the data model, this dimension may equate to the subset 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

To be discussed in a future edition of this publication.

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 Tool Registration Service may be directly related to these services (capabilities).

4.14.8 Examples

Typically every object-oriented OMS provides the inherent capability of attaching methods to object types. OMF provides a library of function attachment services.

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 and Canonical Schema Service

This capability provides a canonical 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 canonical 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

To be discussed in a future edition of this publication.

4.15.4 Types

To be discussed in a future edition of this publication.

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 [27] and CDIF [34] have examples of common external form specifications.

4.15.6 Internal

To be discussed in a future edition of this publication.

4.15.7 Related Services

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

4.15.8 Examples

Examples of global schema are EAST SDS collection [8], the Project Master Database (PMDB) model [59], the PACT System Services [69], and the SLCSE database-schema [67].

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.

There is a wide range of terminology used in this area or at least many words used in differing ways, e.g.,"version," "variant," or "revision." The same words are often used in different ways.

There have been several presentations of generalized version models [24, 47]. 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 version 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 variant and version is subject to the possibly differing perception of people.

Adding to this complexity, the management of relationships between objects and their versions and variants 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. Operations are normally only create version or create variant (sometimes referred

to as an update of the previous version), since delete and update are often considered unnecessary, and are constrained by the structural rules of the version model.

The locking operation on a version or variant 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

To be discussed in a future edition of this publication.

4.16.4 Types

To be discussed in a future edition of this publication.

4.16.5 External

With potentially very large numbers of versions, variants, 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 are 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) and the Composite Object Service, which may then allow versions of composite objects and configurations of differing versions. Other closely related services are the Name and Query Services.

4.16.8 Examples

The set of systems which address versioning includes: PCTE [32]; DSEE [49]; ATIS [23]; NSE [1]; Aspect [40]; and Damokles [25].

4.17 Composite Object Service

This is the service to create, manage, access, and delete composite objects, i.e., objects composed of other objects.

4.17.1 Conceptual

The problem is that 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. 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 are not only found in SEEs in the form of software configurations. For example, documents or tests have components, too.

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 [42] and Dillistone [24]. 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

To be discussed in a future edition of this publication.

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

To be discussed in a future edition of this publication.

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 is related to the Version Service since composite objects may have versions. It may also directly relate 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

To be discussed in a future edition of this publication.

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).

4.18.3 Rules

To be discussed in a future edition of this publication.

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 some of the 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.

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 is 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 Sub-Environment Service.

4.18.8 Examples

Examples from real systems include: Aspect's relational algebra [40] and the PACT query service (DQMCS) built on top of PCTE [69] and ModifyER which provides graphical navigation and update of the SLCSE schema [67].

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. Notification and monitoring are needed to maintain UI consistency checking.

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

To be discussed in a future edition of this publication.

4.19.4 Types

To be discussed in a future edition of this publication.

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 ACTIS [50] and PCTE [32]. The Vbase object-oriented DBMS [3] provides triggers to be associated with changes of values of attributes or the invocation of methods.

4.20 Sub-Environment (Views) Service

The Sub-Environment 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 the full life-cycle process is not conveniently or usefully understood as a single complex system, the SEE framework supporting this process may provide support for breaking down the OM data or even the whole SEE into sub-environments for tool/process access in which parts of the overall project may be carried out. The basis of division may vary; for example, it may be according to ownership; to security clearance; or relevance to the current process. The SEE framework may provide all or few (or even none) of these.

4.20.1 Conceptual

A whole environment in the object management context may be considered as the full schema (defined in some data model), together with the set of objects and possibly a set of operations (if they are part of the data model). The Sub-Environment 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 sub-environment 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 sub-environment. Another way may be by means of derivations, i.e., defined in terms of the operations provided by the OM services, or user-written transformations in some general-purpose programming language.

Hierarchies of sub-environments may exist, in which case the underlying objects and operations are themselves derived. Some notion of self-consistency of a sub-environment (e.g., all the types referred to by objects and operations in the sub-environment have to be available in the sub-environment) may be imposed by the Sub-Environment Service.

A Sub-Environment Service may have a major impact on how integrity and consistency is dealt with by an environment. It may be that while objects are visible only within a sub-environment it is inconsistent with objects within and without that sub-environment. Overall consistency may be provided by control of the visibility of such "inconsistent" data.

Also, this type of service could be provided beyond object management, and applied to the full SEE as a way of subsetting tools and data.

4.20.2 Operations

The set of operations supported by the Sub-Environment Service could include the following: create, update, delete, query sub-environment definitions, instantiate a sub-environment (to cause the bindings to be made),

and possibly operations to access sub-environment elements (e.g., types, objects).

4.20.3 Rules

To be discussed in a future edition of this publication.

4.20.4 Types

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

4.20.5 External

The Sub-Environment 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 programatics interfaces. Significant reuse of sub-environments could be envisaged. Whatever external representations are chosen may take this into account.

4.20.6 Internal

Sub-environments 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 sub-environments 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 dealing to whether sub-environment 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 [16]). It is important that the conceptual and internal aspects of sub-environments are clearly separated in descriptions and discussions.

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 Sub-Environment Service may be related to the models of work supported by the process management services, in particular to the Process Visibility and Scoping Service (see 5.3). 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 sub-environment may be made.

The control of exchange between sub-environments may heavily involve the Version and Configuration Services. There are also clear relationships with any security services in the environment framework.

The significant amount of data storage and manipulation involved in the Sub-Environment Service means it is a major user of the Data Storage and other OM services.

A Sub-Environment 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.

An interesting issue is how the boundaries imposed by the Sub-Environment Service impact upon boundaries of other services.

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 [32]; the "role databases" of ISTAR [22]; the "workshop" and "studio" processes of the Workshop System [16]; Perspective's "domains" [20]; a "view" in CAIS-A [27]; and the role-based access to tools and subschemas of SLCSE [67].

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. 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

To be discussed in a future edition of this publication.

4.21.4 Types

To be discussed in a future edition of this publication.

4.21.5 External

To be discussed in a future edition of this publication.

4.21.6 Internal

To be discussed in a future edition of this publication.

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) [45] 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 provides common formats for transferring information among sites in the worldwide internet computer network.

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 lifecycles. In addition to technical development activities, these potentially include management, documentation, evaluation, assessment, policy-enforcement, business control, maintenance, and other activities.

5.1 Process Definition Service

It is expected that an organization may have a library (repository) of process assets, each of which may be a complete process, a process element (subprocess), or a process architecture. Also, an SEE may provide facilities to define 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 process system and assignment at 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
- allowable concurrency and synchronization (if any) with other processes
- process events potentially of interest for signalling 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 definition, during initiation of enactment, during enactment)
- process state information operated upon (in the case of processes that provide process control services, see 5.5)
- 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 process systems, 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 description.

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

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 definer (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 definer 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 at a complete process definition) 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 description 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 definer 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's sub-environment must be a subset of its parent's sub-environment or that a parent process cannot complete until all its child processes have.

5.1.2 Operations

Process definition operations include:

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

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 services may operate.

An SEE might also provide a process exchange 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 aspects of process definition information (e.g., process assets, process architectures, plans, models, policies), or it may provide type definition facilities for process definers to create types appropriate for their process definition approach.

5.1.5 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 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 (e.g., documents)

- 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)
- a set of declarative rules (e.g., a knowledge base)
- a forms-filling paradigm
- an interactive user dialog with an SEE service.

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 description (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 definition (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 a 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 processes 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 process definition services, if not actually provide some or all aspects of process definition services.

All other categories of process management services listed in this clause may have significant interaction with and dependence upon process definition services. Process enactment services require the existence of at least some of the services and capabilities described for process definition, and the other categories of process management services might be regarded as extensions of the process definition service.

5.1.8 Examples

IPSE2.5's PML [66], Arcadia [68], EAST [8], ESF [61], EFA [51], COHESION [12], and SLCSE [67] all provide some process definition 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

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

process agents). Note that enactment is only one possible usage of process descriptions; 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 defined 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 definition 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.

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.

5.2.2 Operations

Process enactment operations might include:

- instantiate (if not part of process definition service)
- bind
- enact
- suspend and restart
- abort
- commit
- checkpoint and rollback
- simulate
- analyze (various)
- change (dynamic).

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 enactable process representations (that may be equivalent to process definitions or translated from them).

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

The external dimension of the process enactment service provides 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; the process control services described in 5.5 are typical of service implementations that might need this interface into 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.)

Data for the simulation, analysis, or behavior modeling of a process may be extracted from past-project performance libraries that capture historical performance data from executing processes or provide simulated data scenarios to support "what if" planning; these might be provided by the object management facilities.

5.2.7 Related Services

Process definition services are closely related, as they provide 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 process definition services) may extend the semantics of specified process systems or utilize process state information maintained by the process enactment services.

Object management services are related as process enactment services interact 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 [66], EAST [8], Arcadia [68], EFA [51], COHESION [12], SLCSE [67], and EIS [48] provide some support for process enactment.

5.3 Process Visibility and Scoping Service

Several enacting (sub)processes may cooperate to achieve the goal of a higher level process or process system. Logically, the extent of such cooperation is part of the definition of processes and may be provided by integrated visibility and scoping features in a unified set of process definition services; however, independent services may be provided to deal with inter-process interactions such as visibility of common data, common events, and propagation of information.

5.3.1 Conceptual

The process visibility and scoping services define which portions of an enacting process's 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 communication (interchange of data) or coordination (interchange of control). Visibility and scoping services 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.

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, these are rules affecting process definitions.

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 subenvironment types (see 4.20).

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 specify these services 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 (e.g., documents)
- 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)
- a set of declarative rules (e.g., a knowledge base)
- a forms-filling paradigm
- an interactive user dialog with an SEE service.

An effective user interface is a crucial factor in the effective incorporation of specification of these services into an SEE.

5.3.6 Internal

Process visibility and scoping services could be supported by the same mechanisms that support process enactment (basic SEE framework services, tools, and humans) and that support process definition (especially open or closed object stores for specifications).

5.3.7 Related Services

Process visibility may use the operations within the subenvironment service (see 4.20) to provide scoping properties.

Process definition services are closely related, as they provide 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 process definition services) may extend the semantics of specified process systems or utilize process state information maintained by the process enactment services.

Object management services are related as process enactment services interact with them, 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 [51], SLCSE [67], EAST [8], ESF [61], Arcadia [68], and ISPE 2.5 [66] all provide some process visibility and scoping services.

5.4 Process State Service

During enactment, a process has an "enactment state" that changes. Certain changes in the enactment state of a process may be defined as "events" and may act as conditions or constraints affecting other processes; again, logically this may be an integrated part of the process definition services or independent services in the framework.

5.4.1 Conceptual

The process state services provide much useful information regarding the status of the project (an enacting complete process) for process engineers and other enacting processes. This information may be used for current or future decision making (i.e., planning) and to coordinate the activities being conducted within the project.

Process state services include:

- process state definition services
- process monitoring services
- event management services
- process transaction definition and management services.

The process state encompasses a wide variety of information. Some of it is process specific. This portion includes both the process's 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).

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

Event management is another process state service that detects the occurrence of specific events and invokes processes to respond to these events. 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. Just as the state monitoring service (see 4.19) allowed the definition of states to enable actions to be triggered, so the event management 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. One would however expect the granularity of events that may be defined to, and detected by, the event monitoring service to be at least at the level of initiating, interrupting, or completing processes. 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 definition service (see 5.1).

Transaction definition and management is another process state service. 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., [46], [6], [38], [28]). 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 information base 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 within a process should not necessarily be discarded if a failure occurs
- It is unreasonable to lock the information used by a 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 process the database may have to go through what would be considered externally as an inconsistent state in order to satisfy a post-condition.

5.4.2 Operations

Event monitoring operations might 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)

Analogous operations may be provided by a framework to deal with specification of process enactment state information to be maintained by enactment services, specification of object state information to be regarded as part of process state, process monitoring, and process transaction definition and management.

5.4.3 Rules

All process state services are subject to constraints in their effect as defined by visibility and scoping operations. Possible rules might be prohibitions or restrictions on how process transactions may be nested or redefined dynamically, or validity for partial completion.

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 specify these services 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 (e.g., documents)
- 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)
- a set of declarative rules (e.g., a knowledge base)
- a forms-filling paradigm
- an interactive user dialog with an SEE service.

An effective user interface is a crucial factor in the effective incorporation of specification of these services into an SEE.

5.4.6 Internal

Process state services could 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 the event-monitoring service and the process execution service.

5.4.7 Related Services

Process definition services are closely related, as they provide 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 process definition services) may extend the semantics of specified process systems or utilize process state information maintained by the process enactment services.

Object management services are related as process enactment services interact with them, at least for operation upon project objects and sometimes for storage of process definition representations and process enactment state information.

5.4.8 Examples

IPSE2.5 [66], Arcadia [68], EAST [8], ESF [61], EFA [51], COHESION [12], EIS [48], and SLCSE [67] all provide some process state services.

5.5 Process Control Service

A process being enacted by an SEE may be recorded, measured, controlled, managed, or constrained; this may be done by other processes with either human or machine enactment agents.

5.5.1 Conceptual

Process control services provide for the enactment of processes that operate on other processes, generally at the level of a life-cycle process. Process control services are services provided in support of the enactment of software processes. Process control services also provide for the modification of software engineering processes.

Process control services also provide control for a portion of the SEE available to users, based on the user's role and the resource management service's knowledge of process enactment agents and process enactment support mechanisms. Project management capabilities could be developed using various process control services.

Process control services include:

- *metrics collection service:* This service provides the user the ability to specify and record both product- and process-related metrics of interest to the project, the organization, the customer, or process improvement objectives.
- *auditing and accounting service:* The audit and accounting service exists to maintain and present a record of what has been done and been used within the development environment. There are two main purposes for this recording: "auditing" and "accounting." The first concentrates on checking that what happened was done correctly (from some point of view) and the second concentrates on the cost of services used and linking these costs with those to which they should be debited.
- *scheduling (including enactment planning and tracking):* This service is used to specify when processes are to be enacted and to resolve conflicts when processes compete for resources. This is a part of project management information of interest in and tracked against project plans.
- *selecting history elements to record:* This service provides identification of process state data of interest for recording; during process enactment, that data is maintained by the process state services and captured by the history collection service. The state data for selection might include the following categories:
  - data required for performance analysis, such as process enactment time, execution duration, or error history
  - data to be recorded prior to the suspension of process enactment per request from the scheduler
  - acknowledgement/handshake data, interrupts, and events occurring during process enactment
  - data required by other processes.
- *history collection service:* Knowledge of historical project events or historical knowledge of other projects may influence and guide the conduct of the software development project. In addition, most projects have organizational or contractual requirements to capture a project history. For these reasons, one of the responsibilities of the history service is to capture and record aspects of the process state and to make that process history usefully available.
- *configuration management (CM):* This service consists of configuration identification, configuration audit, configuration status, and configuration control. The service provides a means for identifying and controlling baselines or configurations in addition to individual items. Through automation of the CM process, a means for capturing deliverable software and system documentation is achieved. CM services include library services, change control services, and problem reporting services. They are closely linked with project management services and project policies. For example, delivery of products could be

defined in terms of their submission to a CM organization. Thus, the status of particular tasks may be directly related to configuration status reports.

- *policy enforcement:* This service "enforces" any of a variety of policies that a project or organization may apply to a project development process. For example, the automatic evaluation of code to determine violations of coding standards may be provided by this service. Access rights to objects and processes, applied to user role types or instances of users, is another example. Customer- or organization-imposed security systems and reuse are other possible policies imposed on projects that an SEE may support.
- *quality assurance (QA):* This service consists of services that control and measure the quality of products. Often, CM is viewed as part of QA. As in the case of the relationship between metrics and auditing and accounting, CM is separated from QA in the reference model because either may be important to a project independently of the other. Services included as part of QA are: tracking the number of errors in specific modules, reliability predictions, complexity measures, testing, and development of QA capabilities.
- *process queries:* Project users (especially managers) typically define tailored reports of various aspects of process state, metrics, and history that they need periodically. These may be provided via interactive queries or formatted, printed reports.
- *analyses:* Additionally, project users may be provided with analyses of collected process-related objects, perhaps for the purpose of process evolution and improvement.

5.5.2 Operations

Process control operations (for each process control service) might include:

- define
- bind (to another process(es))
- select ("turn on")
- deselect
- record
- analyze recorded information
- change
- delete.

5.5.3 Rules

All process state services are subject to constraints in their effect as defined by visibility and scoping operations.

5.5.4 Types

Representations of process state (5.4) known by a framework's process enactment services and process state services are used by the framework to provide process control services. It is possible that a framework has fine-grained types for representing the differing linguistic aspects of process services.

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 specify these services 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 (e.g., documents)
- 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)
- a set of declarative rules (e.g., a knowledge base)
- a forms-filling paradigm
- an interactive user dialog with an SEE service.

An effective user interface is a crucial factor in the effective incorporation of specification of these services into an SEE.

5.5.6 Internal

One way of implementing most process control services is by using the object management services to model information about process execution. This enables a general tools to be easily provided (or even be available by default) to provide the process control services.

Furthermore, process control services could be supported by the same mechanisms that support process enactment – basic SEE framework services, tools, and humans.

5.5.7 Related Services

The process enactment service might provide the functionality of the process control services if process control services are actually processes.

Process definition services are closely related, as they provide 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 process definition services) may extend the semantics of specified process systems or utilize process state information maintained by the process enactment services.

Object management services are related as process enactment services interact with them, at least for operation upon project objects and sometimes for storage of process definition representations and process enactment state information.

5.5.8 Examples

IPSE2.5 [66], Arcadia [68], EAST [8], ESF [61], EFA [51], COHESION [12], and EIS [48] all provide some process control services.

5.6 Process Resource Management 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) by project management and captured in a process plan. These allocations are often provided by services independent of those for process definition.

5.6.1 Conceptual

Resource management services include:

1. the definition of resource management types,
2. the registration of resources through the instantiation of resource management types, and
3. the interrogation of resource management 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, 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 users are 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 user role management service exists to handle information about people and roles and the relationships between 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 of the roles and people such as: experience; age; or ability. Relationships may exist between roles such as: "all people carrying out quality assurance may carry out the programming role."

Definitions for resource management types could include attributes, operations resource types perform, and objects required to support their operations.

Resource registration (mapping) permits the instantiation of resources onto defined resource management types, e.g., enactment agent types and enactment support mechanisms.

Resource interrogation permits users, processes, and processors to interrogate resource management types and their instantiations to satisfy the needs of processes being enacted or processors enacting processes.

5.6.2 Operations

Process resource management operations might include:

- define (e.g., create, modify)

- register
- map
- remap
- interrogate.

5.6.3 Rules

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

Indirectly, these services define constraints on process enactment services.

5.6.4 Types

The models managed by the process resource management services 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.

There are some roles of which there may be 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

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 specify these services 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 (e.g., documents)
- 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)
- a set of declarative rules (e.g., a knowledge base)
- a forms-filling paradigm
- an interactive user dialog with an SEE service.

An effective user interface is a crucial factor in the effective incorporation of specification of these services into an SEE.

5.6.6 Internal

One way of implementing most process resource services 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 control services. However, some frameworks implement resource management independently of object management services.

Furthermore, process resource management services could be supported by the same mechanisms that support process enactment – basic SEE framework services, tools, and humans..

5.6.7 Related Services

Roles relate very closely to the security and the sub-environment services. Process Resource Management Service is related to Framework Administration Services (see 10).

Process definition services are closely related, as they provide 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 process definition services) may extend the semantics of specified process systems or utilize process state information maintained by the process enactment services.

Object management services are related as process enactment services interact with them, at least for operation upon project objects and sometimes for storage of process definition representations and process enactment state information.

5.6.8 Examples

IPSE2.5 [66], ISTAR [22], EAST [8], the Atherton Backplane [57], EFA [51], COHESION [12], EIS [48], and SLCSE [67] have a role database or otherwise provide process resource control services.

6 Communication Service

There is a need for explicit communication between the components of an SEE. Tools have to communicate and exchange information with the set of framework services and frameworks may require services from other frameworks. There is a requirement for a service which supports managed communication among a large number of elements of a populated environment framework.

6.1 Communication Service

Communication Service needs to provide two-way communication between tool to tool, service to service, tool to service, and framework to framework. Several mechanisms have been proposed to enable this transmission; they are defined more fully in 6.1.2.

6.1.1 Conceptual

The Communication Service provide communication over a distributed (in both the logical and physical senses) collection of services and tools. They do not provide a distribution service per se. These services may be used for two-way inter-tool, inter-service, tool-to-service, and framework-to-framework communication.

Various mechanisms have evolved for providing these basic communication services. These include messages, process invocation, remote procedure calls, and data sharing as means for interprocess communication.

6.1.2 Operations

Operations vary markedly, depending on the nature of the communication that is supported. Many of the operations depend on various forms of name or location services, for connecting with an OS process or an object location or both.

Messages. The exact set of operations depends on the message type chosen. Apart from create, delete, and update, there may be send, receive, acknowledge, reply, and ignore. There may also be registration and de-registration operations (see clause 10.1) if a multicast delivery service is being used. It is with this operation that a tool or service indicates which kinds of messages it is or is not interested in. 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.

OS Process Invocation and Remote Procedure Calls. 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.

Data Sharing. The operations for this are largely normal data manipulation services; the sharing (communication) occurs often because of common agreement on the location of the data, either in memory or in various forms of secondary storage. Included here are also the operations for other forms of I/O.

6.1.3 Rules

To be discussed in a future edition of this publication.

6.1.4 Types

The object to be exchanged or messages may be categorized in a number of ways. For example, for messages there may be request messages, notify messages, and failure messages. Each type of message may contain fields for different types of information. For shared data, there may be shared understandings of the formats involved.

6.1.5 External

The Communication Service is most likely made available through procedure call interfaces.

6.1.6 Internal

Within a framework, 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,
3. tool to service, and
4. framework to framework.

All are implemented in very different ways for the various means of communication.

Messages. There are a very large number of conceivable "message protocols" and implementations which could be employed.

There are three fundamental methods for exchanging messages:

"point to point"	The sender identifies the recipient(s);
"broadcast"	The message goes to all tools and services; and
"multicast"	Some selection mechanism identifies the tools and services that receive the message.

OS Process Invocation and Remote Procedure Call. These capabilities are much more likely to be implemented as intimate parts of the underlying operating system.

Data Sharing. These services may involve various details in their implementation, including shared memory and various I/O protocols. The form of data sharing represented by file sharing is typically implemented as an intimate part of the underlying operating system. Frameworks promote data sharing through the OMS.

6.1.7 Related Services

All services communicate via the Communication 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.

6.1.8 Examples

Particular examples of the types of communication services that may be provided are 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 have message services are: ESF [61], ACTIS [50], HP SoftBench, and the OMG Object Request Broker.

7 User Interface Services

User Interface Services involve all aspects of the framework, including components of the environment such as process management and object management, the integration of the user interfaces of the tools, and the tools themselves in order to provide presentation attributes between the tools and the user of the environment. Because of the desirability of inter-tool user interface (UI) consistency, a coherent set of UI services must be offered as part of the environment.

Seeheim Reference Model

Since 1982, the most influential user interface reference model has been the Seeheim model (3), defined by a group of designers at a workshop sponsored by Eurographics and IFIP in Seeheim, Germany [39].

This model defines the run-time user interface as consisting of three components, the application interface, the dialogue 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

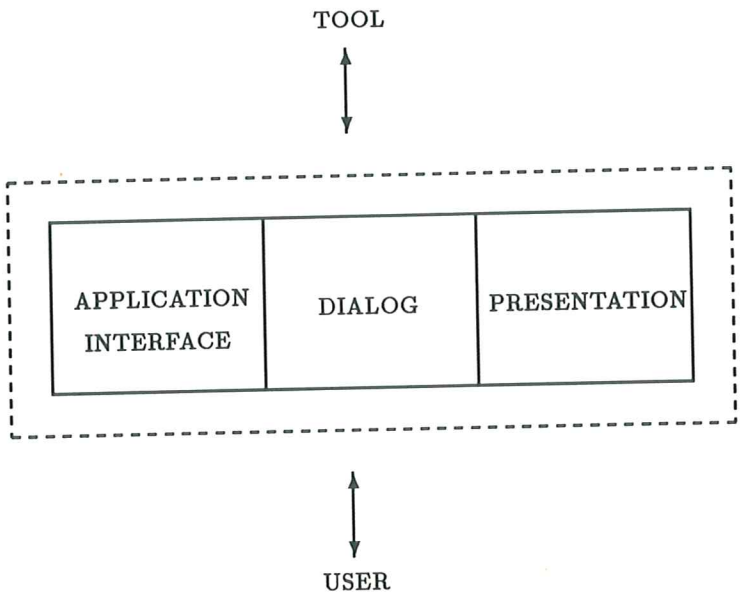

Figure 3: User interface model.

the physical appearance of the user interface including all device interactions. The dialogue control component manages the dialogue between the user via the presentation component and the application.

At that time, a lexical representation of the information flow back and forth between user and application seemed most appropriate, so these three components were thought of as exchanging tokens (like tokens identified during the lexical phase of a compilation).

Popular window system implementations have tended to honor the Seeheim model more in the breach than the observance. The X Window System from MIT [62], 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 dialogue control components, but have neglected the interfaces to the application and presentation components.

New techniques and ideas in user interfaces have led to a reappraisal of the Seeheim model. Rapid prototyping of applications and their user interfaces have made isolating change to the relevant component more important than it was. User interface toolkits which are functionally separate from the higher-level presentation component require an elaboration of the Seeheim presentation model.

The set of services needed to support various user interface services are given in the following clauses.

7.1 User Interface Metadata Service

7.1.1 Conceptual

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

7.1.2 Operations

Some of the necessary operations are: create, delete, query, and modify.

7.1.3 Rules

To be discussed in a future edition of this publication.

7.1.4 Types

To be discussed in a future edition of this publication.

7.1.5 External

To be discussed in a future edition of this publication.

7.1.6 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, might be stored in more volatile and efficiently accessed storage.

7.1.7 Related Services

To be discussed in a future edition of this publication.

7.1.8 Examples

To be discussed in a future edition of this publication.

7.2 Session Service

7.2.1 Conceptual

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

7.2.2 Operations

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

7.2.3 Rules

To be discussed in a future edition of this publication.

7.2.4 Types

To be discussed in a future edition of this publication.

7.2.5 External

Internationalization (e.g., multiple-language character sets, formats for date and time) are determined by the external view of the Session Service.

7.2.6 Internal

To be discussed in a future edition of this publication.

7.2.7 Related Services

The Session Service depends on the Security Service for authentication, trusted paths, encryption, etc.

7.2.8 Examples

A particular tool may require a 24-bit true color display device, while the user has started the session on a monochrome display. A tool may run most efficiently on a display with built-in 3D PHIGS support, but also run on a plain 2D display.

7.3 Security Service

7.3.1 Conceptual

The Security Service must mediate 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.

7.3.2 Operations

The authentication operation permits registration of authentication requirements, requesting of an authentication token, exchanging of an authentication token, and querying of an authentication token.

Managing trusted paths includes opening a trusted path, re-authenticating a trusted path, and closing a trusted path.

Representation includes querying security labels, displaying security labels, and warning of security label violations.

7.3.3 Rules

To be discussed in a future edition of this publication.

7.3.4 Types

To be discussed in a future edition of this publication.

7.3.5 External

To be discussed in a future edition of this publication.

7.3.6 Internal

There must be an efficient and well-defined interface between the UI and the Security Service.

7.3.7 Related Services

The security and integrity of the UI depend on the underlying security model and services provided by the environment.

7.3.8 Examples

To be discussed in a future edition of this publication.

7.4 Profile Service

7.4.1 Conceptual

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 environment manager may set up particular collections of tools or repository views for various user roles.

7.4.2 Operations

Some of the necessary operations are create a user or role profile, delete a user or role profile, copy an existing user or role profile, query a user or role profile, modify a user or role profile, and register a prototype user or role profile.

7.4.3 Rules

To be discussed in a future edition of this publication.

7.4.4 Types

To be discussed in a future edition of this publication.

7.4.5 External

To be discussed in a future edition of this publication.

7.4.6 Internal

To be discussed in a future edition of this publication.

7.4.7 Related Services

Managing the profiles for an environment is part of the responsibility of the framework administration and configuration. The Profile Service interacts with the UI Metadata Service and Session Service in determining permitted characteristics for the desired profile.

Internationalization of both tools and applications must be supported by the environment. The environment may have to support changes between sessions, within sessions, multiple language support within a session, and development of multiple language interfaces for applications.

7.4.8 Examples

The X Window System allows users to start a set of applications by putting them in the *.X11Startup* file [62]. The *.Xdefaults* file is used to set resources for various applications. *Xmodmap* redefines key mappings for various national character sets.

7.5 User Interface Name and Location Service

7.5.1 Conceptual

In a distributed or multi-user environment, there must be a facility for specifying what, whom, and where user interface services are being used.

7.5.2 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, and register an existing name or location reference.

7.5.3 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).

7.5.4 Types

To be discussed in a future edition of this publication.

7.5.5 External

To be discussed in a future edition of this publication.

7.5.6 Internal

To be discussed in a future edition of this publication.

7.5.7 Related Services

UI Name and Location Services depends on the underlying OMS Name and Location Services (see 4.4).

7.5.8 Examples

The user needs location-independent names to access services from a networked workstation in distributed environments.

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.

7.6 Application Interface Service

7.6.1 Conceptual

The Application Interface Service is the main data interface between the executing application and the user interface (and ultimately to the user). Although specific implementations of UI's use different methods to exchange data with the user and the environment, some general operations are common to all UI's.

7.6.2 Operations

Some of the necessary operations are get and put data from and to the UI, cut-and-paste data (displayed) from one tool to another, and capture data from a single tool or the whole UI.

7.6.3 Rules

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

7.6.4 Types

To be discussed in a future edition of this publication.

7.6.5 External

To be discussed in a future edition of this publication.

7.6.6 Internal

To be discussed in a future edition of this publication.

7.6.7 Related Services

To be discussed in a future edition of this publication.

7.6.8 Examples

Some tools allow data to be transfer from other tools via cut-and-paste. Some environments allow the appearance of the screen to be captured as part of UI prototyping, to be stored for later use, transfered to video tape, or printed on a laser printer or color film recorder.

7.7 Dialog Service

7.7.1 Conceptual

In the process of exchanging data between the user and the process manager, the UI must observe various integrity constraints.

7.7.2 Operations

The Dialog Service may be grouped into the following set of operations:

Transaction. The UI must support and enforce the Dialog requirements of the framework.

Concurrency. In multi-tasking, multi-user environments, the UI must support concurrency of UI services and operations.

Function Attachment. A means for attaching various environment functions to user actions is necessary.

State Monitoring. The ability to monitor the state of the environment so as to update the UI is necessary.

Constraints. The ability to constrain the behavior of the UI so as to represent the state of the environment is necessary.

7.7.3 Rules

To be discussed in a future edition of this publication.

7.7.4 Types

To be discussed in a future edition of this publication.

7.7.5 External

To be discussed in a future edition of this publication.

7.7.6 Internal

To be discussed in a future edition of this publication.

7.7.7 Related Services

To be discussed in a future edition of this publication.

7.7.8 Examples

To be discussed in a future edition of this publication.

7.8 Presentation Service

There is a need to create and manage the display area provided to users interacting with the SEE.

7.8.1 Conceptual

System design has moved from simple character-based terminals to graphical devices managing multiple windows on a display. The mechanisms for building display objects is the role for the Presentation Service.

7.8.2 Operations

The operations for this service build display objects. This includes operations to create windows, scrollbars, menus, adding text to windows, etc.

7.8.3 Rules

To be discussed in a future edition of this publication.

7.8.4 Types

To be discussed in a future edition of this publication.

7.8.5 External

The objects created by this service appear as screen display items, e.g., icons, windows, menus, scrollbars, and buttons.

7.8.6 Internal

To be discussed in a future edition of this publication.

7.8.7 Related Services

This 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.

7.8.8 Examples

The Xlib and Xtoolkit features of the X Window System is an important example of this service [62].

7.9 Internationalization Service

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

7.9.1 Conceptual

Often local conventions affect the way individuals use computer resources. Such issues like character codes, formats for date and time, and other local standards affect input and output data. This service describes those capabilities that are affected by national conventions and rules.

Some of the relevant issues are:

- Collating Sequences
  - sorting
  - indexes
- Country-specific data formats
- Icons, symbols, and pointer shapes
- Scanning direction.

7.9.2 Operations

Internationalization operations include the following:

date. There is a need to translate between time zones, 12-hour and 24-hour clocks, and other local conventions into and out of framework standard time.

Code-translate. 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.

Data-translate. 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, etc.

7.9.3 Rules

An environment may be localized during installation and configuration. Some environments and tools may allow the administrator to localize them on a per-user or per-session basis. Some may allow the user to choose.

7.9.4 Types

External representation of such objects are often outside of the control of the framework designer. Date formats, monetary units, language syntax, etc., are often dictated by national laws or conventions.

7.9.5 External

To be discussed in a future edition of this publication.

7.9.6 Internal

To be discussed in a future edition of this publication.

7.9.7 Related Services

To be discussed in a future edition of this publication.

7.9.8 Examples

To be discussed in a future edition of this publication.

7.10 User Assistance Service

User Assistance (UA) (or Help) is an environment-wide, cross-tool concern.

7.10.1 Conceptual

To offer context sensitive help, the tool (or UA Service) must 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.

7.10.2 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.

7.10.3 Rules

To be discussed in a future edition of this publication.

7.10.4 Types

To be discussed in a future edition of this publication.

7.10.5 External

To be discussed in a future edition of this publication.

7.10.6 Internal

To be discussed in a future edition of this publication.

7.10.7 Related Services

Installing UA facilities for the various tools is part of the responsibility of the framework administration and configuration.

7.10.8 Examples

To be discussed in a future edition of this publication.

8 Tools

A framework exists to support domain-specific applications. These applications are created by adding tools to the framework.

8.1 Tool Service

The primary function of the SEE framework is to provide support for the various applications that will create a full SEE out of this framework. These applications are commonly called "tools."

8.1.1 Conceptual

The environment framework services exist partly to support one another, but mainly to provide a useful interface which may be used in building applications (or, more generally, facilities) that support particular forms of software development. The name most commonly used for these facilities is "tools."

Here, the term "functional element" is sometimes used and defined in preference to tool. The term "tool" is currently in general use for a whole packaged software product often consisting of a set of cooperating functional elements. A functional element is a piece of software that calls upon the services provided by the SEE framework or other functional elements. Functional elements "plug in" to the SEE framework and communicate with the services provided by the framework via the Communication Service (see clause 6) or via data using the object manager (see 4).

A "composite functional element" ("composite tool" or "toolset") may be constructed as a system of functional elements which present a single interface. Such composition of functional elements is supported by the services provided by the framework.

In some cases a tool may already have facilities built into it which are logically equivalent to some of the framework services (e.g., a diagram support tool may have its own versioned database and user interface). When bringing such a tool into a framework some decisions have to be made. Services within a tool may be accessible as separate functional elements. In this case, the options include allowing the tool to use the equivalent framework services for the same purpose or letting the functional elements within the tool provide an additional service for other tools. It may also be possible to let the framework service access the tool's functional elements.

Once there are tools in the environment which provide services (either through their use of framework services or the encapsulation mechanism (see clause 3.1)), those services become technically indistinguishable from services provided by the framework, since all are accessible in the same way (through the Communication Service). The three main distinguishing properties are: firstly, that all framework services will be available in all environments of the same kind; secondly, that the people supplying framework services (who specialize in designing and building framework technology) will be different from those who supply tools (who know how to use environment framework services to build tools for specialized application areas); and thirdly, all

framework services for environments of the same kind will be designed to work together in harmony, while many tools will be written without knowledge of other tools they will later need to cooperate with (see clause 3.1 for a discussion of integration).

8.1.2 Operations

The operations provided by a tool perform those tasks that implement the application domain of the SEE. For example, a software engineering development environment implementing a standard "waterfall" life cycle would include tools (and therefore operations) that provide many of the following functions:

- reusability
- requirement specification, traceability and analysis
- design specification, traceability and analysis
- prototyping and modeling
- user engineering and human factors
- program generation, compilation, and unit testing
- system integration
- project planning
- project control
- quality assurance
- verification and validation
- configuration management
- maintenance support
- project communications
- document generation and control
- office automation
- staffing allocation and control
- knowledge engineering
- training.

A more complete list of such functions can be found in the E&V Reference Manual [2]. Most tools provide some subset of the following classes of operations: create, define type, define attribute(s), delete, modify, translate, process, compare, review, select, convert, apply, store, retrieve, and invoke.

8.1.3 Types

Each tool operates on objects of specific types. The objects may vary in complexity from simple scalar data like integer parameters to an OS process up to complex data objects defining file systems or large source programs including complex syntactic and semantic information about these objects. Passing common objects among diverse tools is a major goal of SEE integration activities (clause 3.1).

8.1.4 Rules

To be discussed in a future edition of this publication.

8.1.5 External

To be discussed in a future edition of this publication.

8.1.6 Internal

To be discussed in a future edition of this publication.

8.1.7 Related Services

Tools need to communicate with each other, with users and with framework services. Therefore, tools utilize most of the other services of the framework. It can generally be said that most framework services exist primarily to support and provide facilities for tools.

8.1.8 Examples

Almost any program used by a programmer is a tool. Tools may be single processes enacted by a user or a program (e.g., an editor such as VI or EMACS) or may be collections of cooperating processes (e.g., most C compilers consist of separate programs that preprocess macros, convert source to another form and convert this internal form to executable machine language). Framework services may also be enhanced by tools, such as the ModifyER tool as part of SLCSE [67].

9 Policy Enforcement Services

The reference model uses the term "policy enforcement" to cover the similar functionality of security enforcement, integrity monitoring, and various object management functions such as configuration management. It is not specific about what the policies are or how they are expressed, except for the case of DoD security policy.

Policy enforcement requirements are described in terms of *confidentiality*, the prevention of disclosure of information to unauthorized users; *integrity*, the prevention of unauthorized or uncontrolled modification of resources; and *conformity*, adherence to a plan or (process) model of activity. A framework may be capable of enforcing three types of policies: confidentiality policies, integrity policies, and conformance policies.

Such policies divide naturally into two classifications, *mandatory* and *discretionary*. Mandatory rules are those created or imposed by an administrator, a person possessing authority in the area the rules cover. Discretionary rules are those chosen by a user to govern manipulation of those things over which he is said to have ownership. There are therefore six kinds of policy enforcement:

- mandatory and discretionary confidentiality
- mandatory and discretionary integrity
- mandatory and discretionary conformity.

Policies are stated in terms of the *objects* of the framework, generally stored in the object manager, and 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* because they have the authority to manipulate particular policies. The things policies apply to are called "objects"; other entities referred to in policy statements are called "subjects" and include users, devices, programs, and other computer resources.

The following text draws heavily on the TCSEC (Trusted Computer Security Evaluation Criteria) or "Orange Book" [26] for examples. The nascent ITSEC (International Trusted Security Evaluation Criteria) document being produced by the international information security community should be used for that same purpose when it becomes available.

9.1 Mandatory Confidentiality Service

Mandatory confidentiality policies are those established by an administrator 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") and "need-to-know" markings ("NOFORN," "EYES-ONLY"). These are described in full in the TCSEC.

9.1.1 Conceptual

The TCSEC deals mostly with mandatory confidentiality. It describes in full detail a particular policy and is very specific about how that policy is to be enforced. A framework intended to be used for classified government work may be required to support TCSEC security to one of the several levels (C2, B3, etc.) it describes. A framework that is to be used for commercial work, unclassified government work, or even classified work where all objects are at a single level of classification and the entire host system is physically secure ("system-high") are generally not so required.

9.1.2 Operations

To be discussed in a future edition of this publication.

9.1.3 Rules

To be discussed in a future edition of this publication.

9.1.4 Types

TCSEC mandatory confidentiality rules are based on labels (attributes) attached to every subject and object in the system. These particular labels are normally implemented as strings containing the name of a hierarchical level and the names of zero or more non-hierarchical tags; the Reference Monitor in the trusted computer base compares all components of the subject and object labels to determine if the requested kind of access is permissible. The labels must be changeable only by the administrator.

9.1.5 External

To be discussed in a future edition of this publication.

9.1.6 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 may 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.

9.1.7 Related Services

Confidentiality pervades the entire design of the framework set of services, and security "holes" may appear almost anywhere in the framework set of services. See discussion under the Internal dimension (9.1.6).

9.1.8 Examples

The TCSEC ("Orange Book") [26] provides an emerging international security standard. There exists a reference model for security produced by ECMA TC32/TG9, described in [30] and [31] and summarized by Cole [19].

9.2 Discretionary Confidentiality Service

The TCSEC also defines and discusses discretionary confidentiality, but with much less rigor than mandatory confidentiality. With mandatory and discretionary conformity separated out as separate categories, discretionary confidentiality becomes largely a matter of personal privacy.

Note that access modes allowing modification of the object should actually be considered with integrity controls rather than with confidentiality because modification is an integrity issue (see 9.4).

9.2.1 Conceptual

It should be possible for users to control (permit and deny) individual modes of access to objects that they own by individual users and all members of sets of users. There should be fairly powerful ways to group users into sets using various user characteristics. In the extreme, the rules could take the form of predicate expressions involving the subject's attributes, evaluating to a boolean value.

9.2.2 Operations

To be discussed in a future edition of this publication.

9.2.3 Rules

To be discussed in a future edition of this publication.

9.2.4 Types

Discretionary access rules may be associated with the definition of an object type, thus applying to all instances of that type, or to individual objects.

9.2.5 External

To be discussed in a future edition of this publication.

9.2.6 Internal

As with mandatory confidentiality, 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.

9.2.7 Related Services

To be discussed in a future edition of this publication.

9.2.8 Examples

To be discussed in a future edition of this publication.

9.3 Mandatory Integrity Service

For the purposes of this model, integrity is defined as the protection of objects from unauthorized or unconstrained modification. Integrity, in this sense, provides assurance that a system object maintains (or at least tracks) the "purity" or "goodness" of an object by recording exactly what has been done to the object and how it was done.

9.3.1 Conceptual

Mandatory Integrity rules are those established by an administrator to maintain or track modifications of an object.

9.3.2 Operations

To be discussed in a future edition of this publication.

9.3.3 Rules

In the simplest form of mandatory integrity, the framework provides 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 first alternative allows loss of integrity, but keeps track of the fact that it has happened. The second prevents loss of integrity; it is the equivalent of mandatory confidentiality with the integrity scale "turned upside-down."

There are not very many obvious alternatives to the above rules. Slight elaborations are possible, such as recording the identifier of a subject that lowers an object's integrity level. It has been suggested that multiple different integrity scales could be defined and maintained for every object, but there is no obvious use for a second scale.

When integrity level is being recorded, notification of the user or administrator when an object's level is decreased may be desirable.

9.3.4 Types

The integrity level value of each subject and object should appear as read-only attributes, with only the administrator and OS able to change them.

9.3.5 External

To be discussed in a future edition of this publication.

9.3.6 Internal

The most accurate possible implementation of mandatory 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.

9.3.7 Related Services

To be discussed in a future edition of this publication.

9.3.8 Examples

To be discussed in a future edition of this publication.

9.4 Discretionary Integrity Service

Discretionary integrity controls are implemented by all write, modify, and append permission functions defined for discretionary access controls (See 9.2).

9.5 Mandatory Conformity Service

Conformity policies are the results of automation of operational models. They are expected to be rules such as:

- Modifying a module in "Unit-tested" status sets it to "Untested" status.
- A source module cannot be submitted for integration until it has been approved by two members of the Quality Assurance group.

It is likely that the majority of conformity policies will be mandatory, i.e., set by an administrator such as the project manager.

9.5.1 Conceptual

A Configuration Management tool running on a framework is essentially establishing mandatory conformity policies and using framework facilities to enforce them. A framework should have facilities sufficient for the enforcement needs of any reasonable CM system.

9.5.2 Operations

To be discussed in a future edition of this publication.

9.5.3 Rules

Mandatory conformity rules are often partial expressions of the process being followed by the users of the framework. For example, a rule such as:

> When an Ada-Source file with Approval-Level UNIT-TESTED or higher is edited, Approval-Level is reset to UNTESTED.

represents a small fragment of the description of the development process being followed by the users of the system. There is a current movement toward doing complete statements of the process, called "Process Models," and automating them (clause 5). Such automation would be by means of mandatory conformity mechanisms. It is desirable to be able to run multiple, different processes on the same framework at the same time.

The most general form of conformity rules would be programs that execute in response to events and operate on the set of persistent objects stored in the OMS, where "event" has a wide definition. The terms "triggers," "user exits," and "methods" are often used for this kind of arrangement, the latter especially in an object-oriented OMS. (See 4.19, 5.4, and 7.7).

An obvious target for mandatory conformity policies are the actual expression and implementation of the confidentiality and integrity policies. That is, there could be conformity rules such as:

- The classification of an object can be changed from SECRET to CONFIDENTIAL only by J-Smith or R-Jones and results in the recording of (some set of information) in the security audit file. (This identifies J-Smith and R-Jones as *administrators* in the TCSEC sense.)
- An object classified TOP SECRET, EYES ONLY can be copied only with simultaneous approval of J-Smith and R-Jones.

9.5.4 Types

A desirable trait of the implementation of mandatory conformity rule enforcement is that the system may not achieve through normal operations a rule-violating state. That is, the rule is implemented as code that is invoked when something is done that produces an invalid state to change it to a correct state. The definition of "state" here is very general, involving all of the data and metadata of the system.

9.5.5 External

To be discussed in a future edition of this publication.

9.5.6 Internal

The mechanism that enforces conformity rules about TCSEC confidentiality policies must be inside the trusted computer base.

Mandatory conformity rules may be enforced by mechanisms built into the framework or by tools running on it. It could be argued that the schema of a strongly-typed object manager is in fact a set of mandatory conformity rules that are enforced by the OMS. Process model automation, as discussed above, may be implemented in the type mechanism, especially when it supports the association of user (administrator) code with changes in object state. "Object-oriented" OMSs do this with "methods" and other systems may achieve similar results with "triggers."

One elegant implementation is sometimes called "seamlessness," in which one or more system-supported programming languages support access to persistent objects in the OMS as program variables. Rules may then be written as programs or even single statements in the seamless language. The code may be compiled or interpreted, or some combination of the two; it is similar in some ways to the more powerful "shell" or "script" command interpreter languages.

9.5.7 Related Services

To be discussed in a future edition of this publication.

9.5.8 Examples

To be discussed in a future edition of this publication.

9.6 Discretionary Conformity Service

Individual users would use conformity enforcement to structure their own work environment. If conformity rules are easy to express, powerful, and include active functionality such as the running of tools under the right conditions, it could turn out to be the equivalent of "canned procedures" or "command scripts." Less powerful mechanisms could be used to prevent mistakes.

Discretionary Conformity is essentially identical to mandatory conformity (9.5). Users are effectively performing as administrators of their own data when setting them up.

10 Framework Administration and Configuration 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 sub-environments, 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 and Configuration services. These include Tool Registration, Resource Registration and Mapping, Metrication, and User Administration.

10.1 Tool Registration Service

Tool Registration provides the means to make a tool known to other services in the SEE.

10.1.1 Conceptual

The Tool 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. The complexity of a Tool 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 Tool 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.

There are two aspects to tool 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 Tool 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 Tool 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

To be discussed in a future edition of this publication.

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 Tool 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 Tool 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

To be discussed in a future edition of this publication.

10.2 Resource Registration and Mapping 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 Registration and Mapping 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.

10.2.3 Rules

To be discussed in a future edition of this publication.

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

To be discussed in a future edition of this publication.

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 [27].

10.3 Metrication Service

The purpose of this service is to provide the means to determine the productivity, reliability, and effectiveness of a framework and of the environment built on it.

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 management Process Control Service (see 5.5). 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 Auditing and Accounting Service (see 5.5) whose primary goal is to insure security and usage statistics within the environment.

10.3.2 Operations

This service needs an operation of collect which invokes the data collection service whenever certain events (user command as well as a specific trigger or time period) occur. In addition, there needs to be a report operation for reporting the results of the collected data and a clear operation to initialize the metrication data.

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 User Administration 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

Tool 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 [7] provides a mechanism for determining the appropriate metric to collect based upon the needs of the organization.

10.4 User Administration Service

10.4.1 Conceptual

This service provides the ability to add users to an environment, to characterize their modes of operation and roles (including security privileges), and to make available to them the resources they require.

10.4.2 Operations

An install operation is required. This involves the setting up of preferred modes of each user's operation which governs all their interactions with the environment, the initialization of their ability to act in certain roles, and their relationship to established Process Management controls. Another necessary operation is to be able to cleanly remove a user from the system.

10.4.3 Rules

To be discussed in a future edition of this publication.

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

This service covers only that aspect of User Administration that deals with the ability to specify the desired settings of parameters for a given user. Other framework services (e.g., Policy Enforcement, Project Management) provide 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

To be discussed in a future edition of this publication.

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

To be discussed in a future edition of this publication.

10.5.4 Types

To be discussed in a future edition of this publication.

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 [8].

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] Banuthon F., W. Kim, and H. Korth. A Model of CAD Transactions. *Proc. 11th International Conference on Very Large Data Bases*, 1985.

[7] 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.

[8] Bourguignon J-P. Structuring for Managing Complexity. *Managing Complexity in Software Engineering*, R J Mitchell (Ed.), Peter Peregrinus Ltd., 1990.

[9] Brodie M. L. *On the Development of Data Models*, Springer-Verlag, pp 19-47, 1984.

[10] Brown A.. Database Support for Software Engineering. Chapman and Hall, 1990.

[11] Brown A. W. (Ed.). *Integrated Project Support Environments: The Aspect Project*, Academic Press, 1991.

[12] Bucken M. Digital's Cohesion Taking on AD/Cycle. *Software Magazine* 11(4):71-72, March 1991.

[13] Checkland P. B. *Systems Thinking, Systems Practice.* Wiley and Sons, 1981.

[14] Chen P. P.. The Entity-Relationship Model - Toward a Unified View of Data. *ACM Trans. on Database Systems*, 1(1):9-36, 1976.

[15] 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.

[16] 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.

[17] Codd E. F. A Relational Model of Data for Large Shared Data Banks. *Comm. of ACM*, 13(6):377-387, 1970.

[18] Codd E. F. Extending the Database Relational Model to Capture More Meaning. *ACM Trans. on Database Systems*, 4(4):397-434, 1979.

[19] Cole R. A Model for Security Standards in Distributed Systems. *Computers and Security* 9(4), June 1990.

[20] Cronshaw P. The Experimental Aircraft Programme Software Toolset. *Software Engineering Journal*, pp 236-247, November 1986.

[21] *Data Semantics* (DS-1), pp 7-11, January 1985.

[22] Dell P. W. Early Experience with an IPSE. *Software Engineering Journal*, 1(6):259-264, 1986.

[23] Digital Equipment Corporation. ANSI X3H4 Working Draft Information Resource Dictionary System ATIS. Technical Report ANSI X3H4/90-187, ANSI, February 1990.

[24] 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.

[25] Dittrich K. R. The DAMOKLES Database System for Design Applications: Its Past, Its Present, and Its Future, pp 151-171. Ellis Horwood, 1989.

[26] DoD. Security Requirements for Automatic Data Processing (ADP) Systems ("Orange Book"), Department of Defense Standard 5200.28-STD, 1985.

[27] DoD. Common Ada Programming Support Environment (APSE) Interface Set (CAIS), MIL-STD-1838A, United States Department of Defense, April 1989.

[28] Dowson M. and B. Nejmeh. Nested Transactions and Visibility Domains. *ACM Workshop on Software CAD Databases*, 1989.

[29] 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.

[30] ECMA. Security in Open Systems: A Security Framework. ECMA TR/46, July 1988.

[31] ECMA. Security in Open Systems – Data Elements and Service Definitions. ECMA-138, December 1989.

[32] ECMA. Portable Common Tool Environment (PCTE) Abstract Specification, ECMA-149, December 1990.

[33] ECMA. A reference model for frameworks of computer-assisted software engineering environments. ECMA TR/55, December 1990.

[34] EIA. CDIF Organization and procedure manual. EIA/PN-2329, January 1990.

[35] 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. Review draft dated October 11, 1991.

[36] Feldman S. I., Make – a program for maintaining computer programs, *Software Practice and Experience* (9):255-265, 1979.

[37] Fishman D. et al. Iris: An Object-Oriented Database Management System. *ACM Trans. on Office Information Systems*, pp 48-69, January 1987.

[38] Garcia-Molina H. and K. Salem. Sagas. *Proc. ACM SIGMOD*, 1987.

[39] Green M. The University of Alberta user interface management system. *Computer Graphics* 19(3), pp 205-213.

[40] 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.

[41] Hall P., J. Owlett, and S. Todd. *Relations and Entities*, pp 1-20, North Holland, 1976.

[42] Heimbigner D. and S. Krane. A Graph Transform Model for Configuration Management Environments. *ACM Software Engineering Notes* 13(5):217-225, 1988.

[43] Hull R. and R. King. Semantic Database Modeling: Survey, Applications, and Research Issues. *ACM Computing Surveys*, 19(3):201-260, September 1987.

[44] IBM Corporation. *IBM Systems Journal* 29(2), 1990 (Entire issue devoted to AD/cycle).

[45] IEEE. A reference model for computing system tool interconnections (draft), P1175/D11, *IEEE Computer Society's Task Force on Professional Computing Tools*, May 1991.

[46] Kaiser G. E. A Flexible Transaction Model for Software Engineering. Technical Report, Computer Science, Columbia University, June 1989.

[47] 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.

[48] J. Krueger, 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.

[49] Leblang D. B. and R. P. Chase. Computer-Aided Software Engineering in a Distributed Workstation Environment. *ACM SIGPLAN Notices*, pp 104-112, May 1984.

[50] Leblang D. and D. Hare. CIS 89-008 Draft Proposal: Tool Integration Environment Program Interface. 6 June 1989.

[51] Legait A. Entreprise II: An Integrated and Open Software Engineering Environment, *Proc. of Milcomp 90.*

[52] Meier A. and R. A. Lorie. A Surrogate Concept for Engineering Databases. *Proc. International Conference on Very Large Data Bases*, 1983.

[53] 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.

[54] NIST. Information Resource Dictionary System (IRDS). Federal Information Processing Standard 156, April 1989. (Also ANSI X3.138-1988).

[55] Nguyen G. T. and D. Rieu. *Schema Evolution in Object-Oriented Database Systems.* vol. 4, pp 43-67, North Holland, 1989.

[56] Osterweil L. Software Processes Are Software Too. *Proc. 9th ACM/IEEE International Conference on Software Engineering*, pp 1-13, Monterey, CA, March 1987.

[57] 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.

[58] Peckham J. and F. Maryanski. Semantic Data Models. *ACM Computing Surveys*, 20(2):153-189, September 1988.

[59] 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.

[60] Purtilo J. The Polylith Software Bus. *ACM Trans. on Prog. Lang. and Systems*, 1991.

[61] Shafer W. and H. Weber. The ESF-profile, *Handbook of Computer Aided Software Engineering*, Van Nostrand, September 1988.

[62] Scheifler R. W. and J. Gettys. The X Window System. *ACM Trans. on Graphics* 5(2):79-109, 1986.

[63] Simmonds I. C development on PCTE, *International Workshop on Unix-Based Software Development Environments*, Dallas, January 1991.

[64] Simmons G. L. What is Software Engineering? Technical Report ISBN 0-85012-612-6, NCC Publications, 1987.

[65] Smith J. M. and D. C. P. Smith. Database Abstractions: Aggregation and Generalization. *ACM Trans. on Database Systems*, 2(2), June 1977.

[66] Snowdon R. A., A brief overview of the IPSE2.5 Project, *Ada User* 9(4):156-161, 1988.

[67] 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.

[68] 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.

[69] Syntagma. PACT Tool Writer's Guide. Technical report, Syntagma, April 1988.

[70] 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.

[71] 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.

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

[73] Tsichritzis D. C. and F. H. Lochovsky. *Data Models.* Prentice-Hall, 1982.

[74] 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.

[75] 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.

E C M A

EUROPEAN COMPUTER MANUFACTURERS ASSOCIATION

REFERENCE MODEL FOR FRAMEWORKS OF SOFTWARE ENGINEERING ENVIRONMENTS

ECMA TR/55

NIST Special Publication 500-201

This publication is a revision of Technical Report ECMA TR/55, published in December 1990, and has been prepared jointly by the ECMA/TC33 Task Group on the Reference Model and the ISEE Working Group of the National Institute of Standards and Technology (NIST) of the United States Department of Commerce. This report is also published as NIST Special Publication 500-201.

2nd Edition December 1991