Brief History

When the ECMA Technical Committee (TC33) for PCTE standardisation was formed, one of the decisions was to aim to create a Reference Model for frameworks of CASE environments to assist the standardisation process. A Task Group (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 frameworks of CASE environments. The Reference Model has gained increasing interest in the software environments community.

It should be understood that the Reference Model is totally independent of PCTE and is not intentionally biased towards PCTE. The ECMA group see it as an aid in identifying future standards that will be needed in addition to PCTE. It is also a valuable way of describing, comparing, and contrasting frameworks of CASE environments. The particular services of the Reference Model are described to a degree of detail that is complete enough for the Reference Model to be used to describe existing systems and proposals.

No existing framework covers all aspects of this Reference Model. It is hoped that the concepts described within the Reference Model can guide the evolution of frameworks of CASE environments in the right direction.

Part I of this Technical Report explains what a Reference Model is and outlines the aims of this Reference Model for frameworks of CASE environments. A technique for applying the Reference Model is described.

In Part II the Reference Model is presented. The Reference Model groups together sets of "services". The overall structure of the Reference Model is described together with the services contained therein. Service descriptions are made consistent and complete by the use of a set of "dimensions". Each of the services is described in more detail using the dimensions to structure these descriptions.

This Technical Report does not claim to describe a fully complete Reference Model. It contains the best results achievable in ECMA TC33/TGRM with the given resources. It does not rule out changes to, or further development (or extension of scope) of, the Reference Model.

Accepted as an ECMA Technical Report by the General Assembly of December 1990.

TABLE OF CONTENTS

PART I - INTRODUCTION TO THE REFERENCE MODEL FOR FRAMEWORKS OF CASE ENVIRONMENTS

1. SCOPE

1.1 Scope of CASE Environments and CASE Environment Frameworks

CASE stands for Computer-Assisted Software Engineering. "Software Engineering" means the planned process of producing well-structured, reliable, good-quality, maintainable software systems within reasonable time frames [1]. "CASE Environment" means the system [2] which exists with the direct responsibility for the engineering of software systems.

NOTE

Digits between square brackets refer to the bibliography (page 49).

Alternative equivalent names for a CASE Environment are IPSE (Integrated Project Support Environment), I-CASE (Integrated Computer-Aided Software Engineering), SDE (Software Development Environment), SEE (Software Engineering Environment), ISEE (Integrated Software Engineering Environment), and ISF (Information Systems Factory). No distinction is made between these terms since they are broadly used to describe the same thing.

A CASE environment deals with information about the software under development (e.g. design data, source code, test data, project plans). A CASE environment can deal with information about people working on the production of software and their assignments and management duties. It can deal with company policy and guidelines on the production of software. It can handle customer contacts and requests to tender for software contracts. A CASE environment may store information about target hardware for which software is being developed, but it will not store information about the design or development of the hardware. A CASE environment will not deal with company finances or marketing information because these do not relate directly to software engineering.

These examples make it clearer where the boundary of the systems under discussion lies. They do not imply that a system must provide all of these kinds of facilities to be of interest. The reference model is simply not trying to deal with those parts of systems which lie outside these boundaries.

In short, the CASE environment models the software engineering part of an enterprise. It is a model, automated in many respects, which provides substantial support to that part of the enterprise. It is important to remember that a CASE environment is essentially an information system.

Current thinking in the area of CASE environments is that an environment consists of a (relatively) fixed set of core facilities which form the "Environment Framework", and a set of facilities, called "Tools" (see clause 8), which are more specialised for particular environments and which cannot be presumed to be available in all environments. Tools use the services provided by the environment framework to a large extent. But tools may also use services provided by other tools. Services in one part of the framework may use services in other parts of the framework.

The work of producing a CASE environment is carried out by environment builders. They make use of a CASE environment framework, produced by framework builders Framework Builder, and of tools and utilities written by tool writers. The environment is customized according to an environment definition produced by environment definers to support a software engineering methodology. The customisation process continues throughout the life of the CASE environment. Software developers and other users (e.g. managers, environment administrators, configuration controllers, secretaries) use the customized environment to build target systems according to the customers' software requirements.

1.2 Scope of the Reference Model for CASE Environment Frameworks

A Reference Model is a conceptual (and functional) framework which helps experts to describe and compare systems. It allows experts to work productively and independently on the development of standards for each part of the reference model.

A reference model is thus not a standard itself; it should not be used as an implementation specification nor as the basis for appraising the conformance of actual implementations.

It should identify areas for developing or improving standards, and provide a common reference for maintaining consistency of all related standards.

This document provides a reference model for CASE environment frameworks . This reference model concentrates on characterising and relating the particular services which can be found in environment frameworks. So this reference model does not cover all aspects of CASE environments. It does have a place for tools and does make some statements about them. It does not, however, attempt to examine particular types of tools or to relate the use of tools to particular methods of software development. That is not to say that these issues are unimportant.

The reference model may evolve and expand to cover many more interesting aspects of tools and their use in some future version.

1.3 Coverage of User Interface, Security and Distribution

This reference model concentrates on services which are specific to the requirements of CASE environment frameworks. User interface, security and distribution are three more general areas where much relevant work is going on outside the CASE environments world. They are only briefly covered in this reference model.

This reference model refers to and summarises existing user-interface (clause 11) and security (clause 12) reference models. These are not included or imported as part of this CASE reference model, which recognises these areas as very important but does not address them itself since they are better dealt with elsewhere. It is simply recommended that the other reference models referred to are a good starting point for discussing those subject areas within the context of CASE environment frameworks. There may exist other reference models covering those areas which are just as appropriate.

The reference model is intended to be applicable to environment frameworks that are distributed and heterogeneous. It is thus assumed that a service may be available from any location in the system which forms the CASE environment. The reference model does not constrain any decisions about how this occurs, or where the service resides or is performed. A service itself may be distributed as may be the data and processes it involves. No services are provided to specifically address distribution, though some services may cover some distribution issues (e.g. location, repository, task management, and sub-environment services).

1.4 Aims of the Reference Model

The purpose of this section is to describe and provide rationale for the main aims of this reference model for CASE environment frameworks. An aim has been to keep the number of these requirements to a level such that they can be kept in mind while the reference model is developed and discussed. It does also mean, however, that in some places requirements are grouped together which the reader may believe should be treated more distinctly.

1.4.1 Description and Comparison

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

One justification for this requirement is that it enables validation that the reference model has the correct scope for addressing issues concerned with building environment frameworks.

The area of CASE environments is very young, even within the context of the youth of computer science. It is still developing some of its most basic premises and terminology. Much development effort can be 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 define problems nor is it to impose solutions. One of its major contributions is to provide the necessary framework for them to be meaningfully discussed; a vehicle to explore environments and to identify their strengths and weaknesses.

1.4.2 Evolution of Standards

The reference model should provide a framework for the smooth and coordinated evolution of future standards, in particular to ensure that the early standards are developed in such a way that further standards may easily achieve alignment in the sense of upward compatibility.

A smooth transition to the widespread use of CASE environments will rely on users being able to continue to use their existing techniques and organisation within an CASE environment which will provide more automation and tool support.

System developers should be able to see that the standards that they invest in today will not be obsolete tomorrow.

Inevitably, the reference model itself should also be capable of evolution.

1.4.3 Integration and Interoperability

The reference model should address interoperability and integration of tools.

The wide range of expertise required to provide all the elements of a complex support environment will not be found within a single organisation. Standards provide a way for environment users to buy facilities which meet their requirements from vendors who are not directly cooperating and may even be competitors.

Standards will be very useful in enabling tools to be ported to, and to interwork over, different types of hardware.

1.4.4 Degree of Generality

The reference model has to be able to be used to describe a wide range of CASE environment framework designs, but should balance this against a requirement to be able to define points at which useful standards can be defined.

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

At the other extreme, a reference model could be proposed which prescribes a single design for CASE environment frameworks. In this case, the places where standards could be defined would be obvious, but there would be a lack of any sense of generality and such a model would not allow discussion about CASE environment 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.

1.4.5 Education

The reference model should be capable of being used as the basis for educating systems engineers in the subject of CASE environment frameworks.

There is little doubt that, in appropriate places, the reference model will have to be terse and precise. For instance, all the technical terms used in its description will be defined and formalisms may be usefully employed to some degree.

But that is not to say that explanations of a tutorial nature should not be provided as well. Indeed they are essential to ensure that the reference model can be understood by those who were not involved in its creation.

CASE environments provide opportunities to radically change the way systems are developed. A reference model for CASE environment frameworks has the responsibility to provide a basic education for system engineers, especially while the area is so new.

1.4.6 Unifying Concepts

The number of unifying concepts required to describe the reference model should be small and the model should recognise the importance of the relationships which exist between its elements.

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

The elements of some types of CASE environment frameworks have been agreed to a reasonable degree for some time. Thus their functionality is quite well understood. It is the relationships between these elements which has proved most difficult to establish, and their identification is a major contribution of a CASE environment framework reference model.

1.4.7 Software Development Method Independence

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

The reference model should recognise that there are many approaches to software development that an environment may support.

The reference model should not be unduly influenced by existing CASE environment 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 very useful body of knowledge applicable to a reference model.

1.4.8 Related Models

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

There are other reference models which address standards and improved understanding of CASE environments. It is desirable that, where the CASE reference model addresses a subject area, then any reference model addressing that area may be used successfully in conjunction with the CASE reference model.

There are some reference models which cover scopes more general than CASE environment frameworks. A CASE environment framework reference model may be compatible with one or more such models. If this is the case then it should be explicitly stated in order for characteristics of the general model to be applied.

It is not necessary that one reference model for CASE environment frameworks be compatible with any other.

2. FIELD OF APPLICATION

2.1 Reasons for Application

There are several reasons for applying the CASE environment frameworks reference model to describe existing systems. Some of these can be seen as coming from the aims of the reference model. For example, having means of comparing and contrasting systems and identifying applicable standards.

Another reason is that the reference model itself needs to evolve as research into frameworks and their practical application continues. If it cannot cope with certain aspects, these should be identified and the reference model modified or extended. The same applies if the reference model cannot allow useful, more enlightening statements to be made than could be made without it.

One point to be aware of when applying the reference model is that all existing projects and proposed standards have been developed without any explicit reference model of any sort (let alone this one). Their developers' visions and descriptions may well not line up neatly with the dimensions and boundaries of this reference model. Before the reference model is applied to a system it is vital that those doing so have a good understanding of both the system in question and the reference model itself in order to avoid injustice to either. Such problems will hopefully lessen with more widespread application of the reference model.

2.2 An Application Technique for Evaluation

There is an application technique for using the reference model to analyse CASE environment frameworks. The first step is the acquisition of a set of documents describing all aspects of the system to be evaluated. These may include an interface specification, a user manual, and some internal documentation.

The second step is to identify the services provided by the system. These are then grouped under the relevant service headings of the reference model. This grouping may not correspond to existing documentation of the system and so it is useful to keep a record of which parts of the documentation the services are referenced in.

The next step is to take each reference model service in turn and describe what the system under review does and does not provide. This description is structured by placing points under the relevant dimension headings. The dimensions have been chosen to help clarify such descriptions, and do appear to help in structuring the descriptions, identifying ambiguities and inconsistencies, and providing a checklist for completeness.

A final step is to generate a summary of what has been discovered. It is useful to summarise the degree of coverage systems offer with respect to the overall reference model architecture. Another effective summary is a table showing coverage of the services against the dimensions. In this way for example, an interface specification should stand out as having no coverage of the internal dimension.

2.2.1 Measuring the External Dimension

There is a more precise range of criteria for summarising the external aspect of a service:

A: The operations covering the service are complete, consistent, and (relatively) elegant. There exists an abstract (language independent) binding for the service. There exists one or more bindings for the service.

B: The operations covering the service are complete, consistent, and (relatively) elegant. There exists one or more bindings for the service.

C: The operations covering the service are complete, consistent but not elegant. There exists one or more bindings for the service.

D: The operations covering the service are incomplete and not elegant.

O: The service is not covered.

The notion of "elegance" is obviously subject to interpretation. It may include the homogeneity of style, the uniform ordering and naming of parameters, the naming of operations, etc.

3. ACRONYMS

ATIS	A Tools Integration Standard
CASE	Computer-Assisted Software Engineering
DDL	Data Definition Language
ESF	EUREKA Software Factory
I-CASE	Integrated Computer-Aided Software Engineering
ICE	Internal Conceptual and External
IPSE	Integrated Project Support Environment
IRDS	Information Resource Dictionary System
ISEE	Integrated Software Engineering Environment
ISF	Information Systems Factory
LID	Logical Identifier
OID	Object Identifier
OTI	Open Tool Interface
PCTE	Portable Common Tool Environment
ROD	Rules Operations and Data
SDE	Software Development Environment
SDS	Schema Definition Set
SEE	Software Engineering Environment
TIM	Types Instances and Metadata
UI	User Interface
UIMS	User Interface Management System

PART II - THE REFERENCE MODEL FOR FRAMEWORKS OF CASE ENVIRONMENTS

4. REFERENCE MODEL STRUCTURE

The CASE environment frameworks reference model is based on grouping sets of services together. Figure 1 shows the overall structure of the reference model. The purpose of grouping the services as they are is that standards will cover at least a whole group. Thus the important interfaces between existing and forthcoming standards can be identified. Some of these groupings also enable various kinds of integration to be discussed: presentation integration (user interface services); control integration (task management services plus the message services); and data integration (data repository plus data integration services).

The diagram should not be interpreted as a set of layers (see 5.4.2 for a discussion of the relationships between the services). The message server services allow two-way communication: service-to-service; tool-to-tool; and tool-to-service. Remember this is not an implementation design.

The tool slots reserve a place for extending the facilities provided by the environment framework with integrated tool sets. The extra tools (or services) available will vary across environments just as database schemas and process programs will be different. The presentation aspects of a tool are deliberately separated from its semantic behaviour 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.

This "Toaster Diagram" (Figure 1 by George Tatge of Hewlett Packard) is useful in providing a top level summary of the areas covered by different CASE frameworks and standards.

Later sections of this document describe each of the parts of the model in some detail. There is now a summary description to give the reader a chance to view the overall model so that the details can be read in context. The order of description follows Figure 1 from top to bottom.

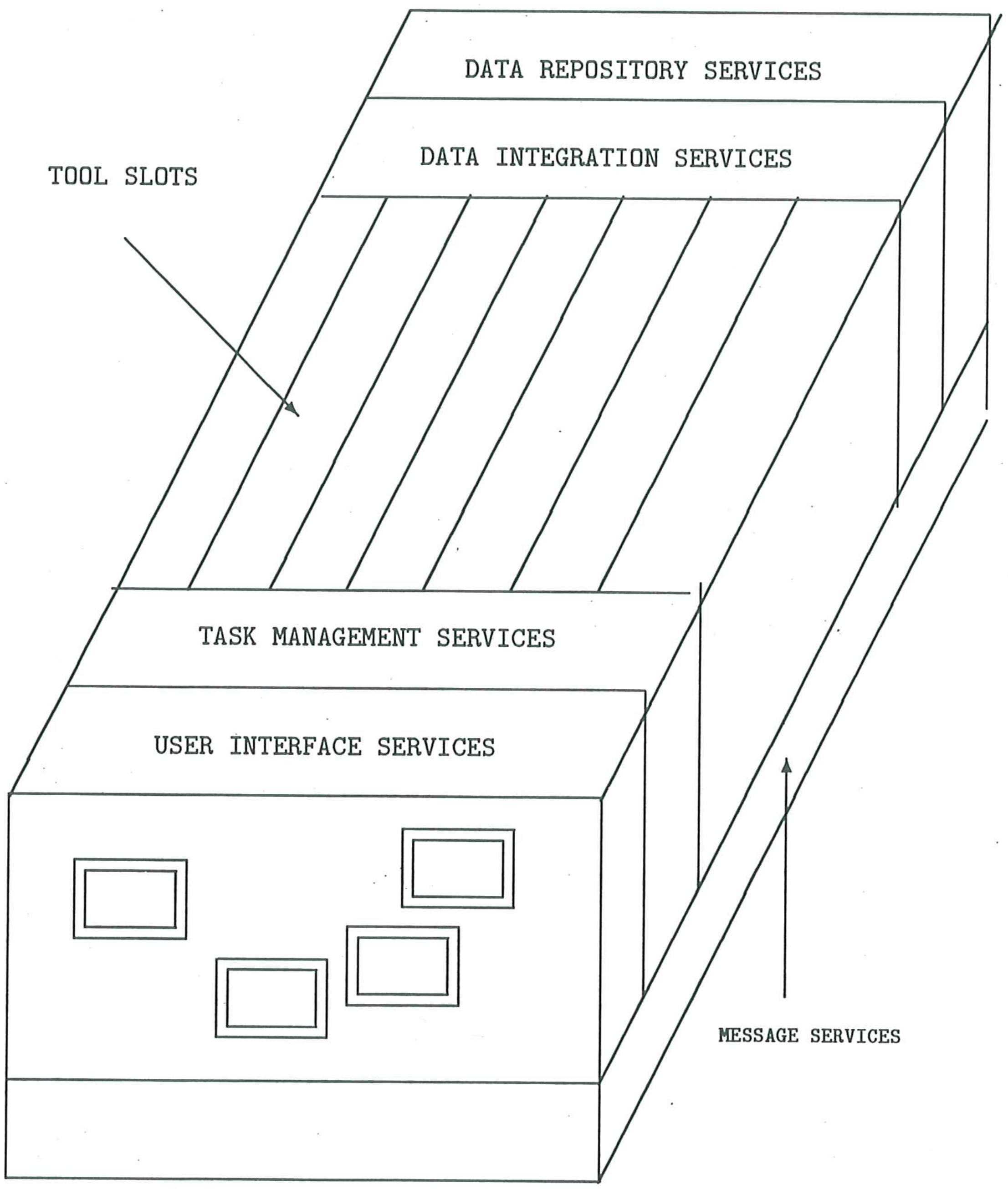

Figure 1 - Overall Reference Model Structure

4.1 Data Repository Services

The maintenance, management, and naming of data entities or objects and the relationships among them is the general purpose of the data repository. Basic support for process execution and control is also addressed here along with a location service to support physical distribution of data and processes.

The services here are:

Data Storage	The data storage service (in 6.1) provides a means and a place for keeping things or references to things.
Relationship	The relationship service (in 6.2) allows the definition and maintenance of relationships.
Name	The name service (in 6.3) maintains the relationships between surrogates and names.
Location	The essence of the problem to be solved (in 6.4) is to enable all the services in the CASE environment framework to be available over a distributed collection of processors and storage devices.
Data Transaction	A "transaction" (in 6.5) 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).
Concurrency	The Data Transaction Service is required to cope with the simple fact that things cannot be guaranteed to work all the time. This is independent of whether or not resources are being shared by more than one transaction and transactions are happening concurrently (see 6.6).
Process Support	The process support service (in 6.7) provides the basic support mechanisms for active entities.
Archive	The archive service (in 6.8) allows online information to be transferred to an offline form.
Backup	It is the backup service (in 6.9) which can be relied on to restore the development environment to a consistent state after media failure.

4.2 Data Integration Services

The data integration services enhance the data repository services by providing higher-level semantics and operations with which to handle the data stored in the repository.

The services here are:

Version	The service concerned with managing data from earlier states is the version service (in 7.1).
Configuration	The problem addressed by the configuration service (in 7.2) is that one wants to look at and operate on a "thing" as a single item sometimes but at other times consider it as a (structured) collection of things.
Query	The query service (in 7.3) is an extension to the data storage service's "read" operation. In the general case, it returns sets of values retrieved from sets of entities.
Metadata	Metadata is data about data. The metadata service (in 7.4) provides control and maintenance of metadata.
State Monitoring	The state monitoring service (in 7.5) enables the definition/specification of database states and state transformations, and actions to be taken should these states occur or persist.
Sub-Environment	The sub-environment service (in 7.6) enables the definition of a subset of the data and operations in the whole environment along with the definition of a two-way transformation between the data and operations in the

	underlying environment and the set of derived data and operations of which the sub-environment consists.
Data Interchange	The data interchange service (in 7.7) satisfies a need to be able to exchange data between environment frameworks.

4.3 Tools

A tool (see clause 8) is a piece of software that is not part of the CASE environment framework and which calls upon the services provided by the CASE environment framework. Tools enable the general services to support a particular application.

4.4 Task Management Services

The purpose of these services is to provide the ability to insulate the user from the details of fine grain tools in the framework. The task management (alternative terminology is software process management) services provide a layer of abstraction which allows the user to deal with tasks as opposed to accomplishing each job by a tedious series of invocations on individual tools. They also allow for the increase of "intelligence" within the framework so that increased automation of the overall environment may be realistic.

The services here are:

Task Definition	The task definition service (in 9.1) supports the definition of tasks.
Task Execution	The task execution service (in 9.2) provides what is necessary to control and support the execution of tasks.
Task Transaction	The task transaction service (in 9.3) supports the notion of transaction at the task level.
Task History	One of the responsibilities of the task history service (in 9.4) is to record information about the execution of tasks. Its other main responsibility is to make that information usefully available.
Event Monitoring	Just as the state monitoring service (see 7.5) allowed the definition of states to enable actions to be triggered, so the event monitoring service (in 9.5) supports the definition of events, and actions to be taken should an event happen.
Audit and Accounting	The audit and accounting service (in 9.6) exists to maintain and present a record of what has been done and been used within the development environment.
Role Management	The role management service (in 9.7) exists to handle information about people and roles, and the relationships between them.

4.5 Message Services

These services aim to provide a standard communication service which can be used for inter-tool and inter-service communication.

The services here are:

Message Delivery	The message delivery service (in 10.1) provides a communication service which can be used for two-way inter-tool, inter-service, and tool-to-service communication.
Tool Registration	The tool registration service (in 10.2) deals with making a tool known to the message services.

4.6 User Interface

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 11.

4.7 Security

The CASE reference model regards security as a service which crosses many of the boundaries of the reference model divisions. A very brief overview of a security reference model is given in clause 12.

4.8 Framework Administration and Configuration

A CASE environment will have to be carefully administered, not least because its precise configuration may be constantly changing to meet the changing needs of the software development enterprise. Clause 13 discusses this.

5. DIMENSIONS OF THE REFERENCE MODEL

The reference model for CASE environment frameworks divides an environment framework into functional elements which are called "services" . Services are grouped together where they are closely related from the point of view of their functionality. This grouping offers the very high level reference model diagram in Figure 1.

Despite the model breaking CASE environment frameworks down into about 30 services, it is useful to structure descriptions of those services to ensure 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 which 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.

The advantage of using dimensions to structure a service description is that it not only enables the important characteristics of the service to be identified and emphasized, but also offers consistency between descriptions of different services (enabling them to be related more easily in different ways) and between descriptions used for a service description for each system the reference model is applied to.

In a way, the concept of "dimension" in this reference model is somewhat similar to the idea of a "projection" (a perspective is a model that describes some subject from a particular viewpoint. A projection is a perspective that is a complete description of the whole subject considered) in other models (e.g. the ANSA model for distributed systems [4] and the CCITT reference model for systems support environments [5]). In the latter case, the whole system is described with respect to a point of view (e.g. the implementation view or the information handled). In the case of the CASE reference model, the services are first separated and then the emphasis moves to describing them from different points of view. Perhaps the two overall approaches could be considered equivalent since the reference model service descriptions could be re-divided according to dimensions.

The dimensions are now described. Experience indicates that these dimensions enable a service description which carries all the important information about that service. These dimensions also specifically address those aspects of services which are often overlapped leading to confusion. The aim of the chosen dimensions is to improve the clarity of understanding of descriptions.

5.1 The ICE Dimension

The first, and most important dimension is the "ICE" dimension. ICE stands for Internal, Conceptual, and External. The terminology and much of the semantics is borrowed from the ANSI/SPARC 3-schema architecture [6]. 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).

At the conceptual level the essence of a service should be described without reference to either how it is implemented or to the ways in which it may be made available to other services or to people. Neither a language nor a notation for making the conceptual description is offered by the CASE reference model, although this is an ideal area for the application of abstract formalisms because they satisfy the demands that implementation and presentation issues are not important at the conceptual level.

The internal level of the ICE dimension is the place in which to discuss implementation issues. In general, good design separates implementation issues from functionality provided. If it is difficult to separate the internal level from the conceptual level then this is some indication that a service is not well-designed. In fact, it should be possible to describe a mapping from the conceptual level to the internal level so that the way in which a service is provided can be understood. See figure 2.

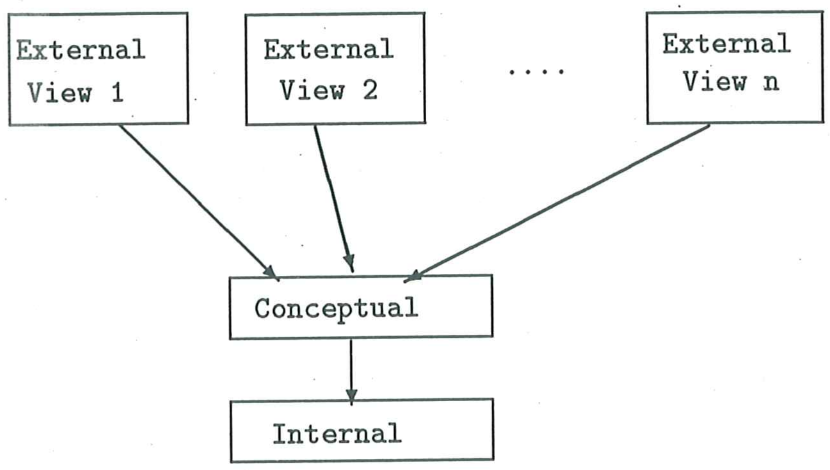

Figure 2 - ANSI-SPARC Schema Architecture

The external level of the ICE dimension offers a place to discuss how the service is made available to be used. A service may be used, for instance, by other services, by tools (or application programs) or quite directly by users. In each case it may make more sense to offer the same service concepts in a variety of ways; each suited to the mode of usage. An example of a service being provided in different external forms is a relational data repository service offering a calculus-based and algebraic interface. There may be some external views of a service which are made available only to a limited subset of the other services.

Just as there is a conceptual to internal mapping, there has to exist a set of external to conceptual mappings.

5.2 The ROD Dimension

The ROD dimension provides a way of obtaining a precise service description in terms of the Rules, Operations, and Data dealt with by a service. The source for these elements is the definition of a data model [7] although they undoubtedly have equivalents in other areas of computer science. Its aim is to identify the set of data manipulated by a service, the operations used to do that, the constraints placed upon the state of the data, and the applicability of operations.

A service provides a set of operations. Operations can be decomposed into just four basic categories: create; update; query; and delete. Create brings into existence a new entity. Update changes the values associated with an entity; Query returns the values associated with an entity; and delete results in an entity ceasing to exist.

The basic set of operations can be composed together to form more complex operations. The sequential methods of composition also fall into three categories of: choice, iteration, and sequence. Concurrency is not precluded. Operations may be parameterised. The ROD dimension is a place to at least identify the set of operations which form part of a service.

Associated with the data and operations of a service are a set of rules to constrain the states the data may reach and the changes to state which operations may make. For instance, a service may allow deletion of entities in general but may restrict the deletion of some entities in particular. So for all services there is a "built-in" set of rules. In addition, some services may provide the ability for additional rules to be defined and parameterised. An example of this may be a service which ensures a manager manages no more than five programmers.

5.3 The TIM Dimension

The TIM dimension separates the concerns to do with Types (generic descriptions), Instances (of types), and Metadata (information recorded about types).

The aim of the TIM dimension is to ensure a proper distinction is made between instances, types, and information about types (metadata). This distinction is as applicable in the task management and tool areas as in the data management area.

The terms are borrowed from the database world (the IRDS work [8] is a reasonable source) but they are used more generally here.

5.4 Other Factors in a Service Description

5.4.1 Degree of Understanding

CASE environment framework technology covers a broad spectrum of subjects in computer science and software engineering. It is inevitable that the degree of understanding of each of these areas will vary. Some ideas are still just theories waiting to be tested while other technology has been successfully used. When describing services it is important not to give the impression that each service is as well understood as another. It should be clear from the description to what degree the service (and its dimensions such as internal or external) is developed. It is also important to state how well the application of a service within a CASE environment is understood. It may be that a well-understood technology from another discipline is employed in a CASE environment framework without proof that it is at all applicable.

5.4.2 Relationships Between Services

The ways in which one service can and does interact with another service is one of the most important areas in CASE environment framework development. One reason for its importance is the desire for an open, extensible, heterogeneous environment, particularly in the sense that different vendors' hardware and software components can work together effectively. For this reason, the identification and possible standardisation of the relationships between services is important.

Another area of interest in service to service relationships is that the effective tight combination of services can result in powerful facilities (an example is where systems make effective use of the data repository and data integration services to handle data and metadata concerning tasks). Such close relationships between services is a conflicting aim to that of clearly defined, standard relationships.

This reference model identifies many of the accepted relationships between services. It does not dictate a set of relationships which must exist since current understanding of environment frameworks and the services of which they consist does not allow that. The relationships identified here should be regarded as sensible ones which can be found in existing systems. Some of these may be considered "layers" but the reference model does not impose layering in any sense. Identification of other relationships is encouraged.

It is also interesting to separate static and dynamic relationships between services. In this way it is not only possible to describe which services can use which others, but also look at the sequences of interactions which can take place. In the CASE reference model, such dynamic aspects are not emphasised because they appear to be areas requiring substantial research.

5.4.3 Justification for Including Services

This reference model does not directly address the requirements for a CASE environment framework. Such issues seem more than adequately addressed in the family of RAC documents (e.g. [9, 10]) as well as the literature concerning requirements for certain elements of CASE environment frameworks such as the database [11, 12]. However, The service descriptions here often include a short note to indicate the usefulness of the service. It is important to understand that the reference model is not saying that a system being described by the reference model has to have every service. The reference model should have a place in which to discuss important aspects about all the services which may sensibly contribute to a CASE environment framework.

5.5 Summary of the Dimensions

Figure 3 summarises the dimensions which should be considered for each and every service. Clearly, some are more important than others when dealing with particular services, and the descriptions within this reference model concentrate on some of the most interesting and important points. A full description of a system according to this reference model should check every aspect of each service in order to ensure completeness. Each point made in a service description rests at a point in the multi-dimensional space defined by the reference model 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. To some extent, this is what the overall reference model structure does.

In this presentation of the reference model, the dimensions are used as sub-headings for any points concerning the particular dimensions. A "Conceptual" heading is always used and the other dimensions (if included) are used in the following order: Justification, Conceptual, Degree of Understanding, Data, Types, Operations, Rules, Metadata, Instances, External, Internal, and Related Services.

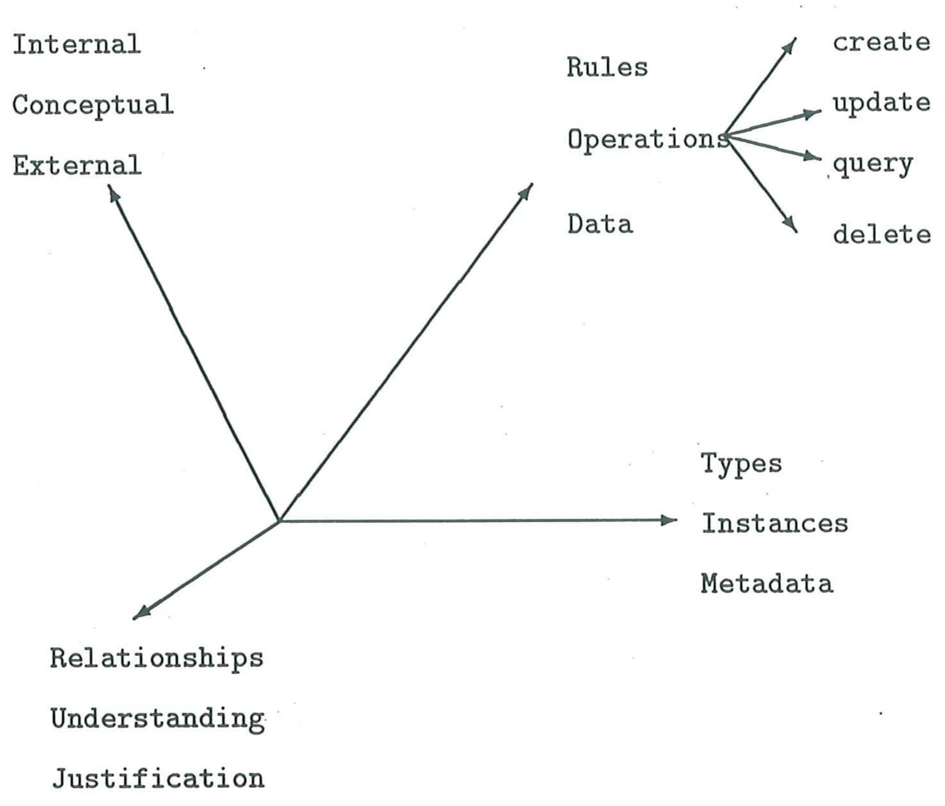

Figure 3 - Dimensions of the Reference Model

6. DATA REPOSITORY SERVICES

The maintenance, management, and naming of data entities or objects and the relationships among them is the general purpose of the data repository services. Basic support for process execution and control is also addressed here along with a location service to support physical distribution of data and processes.

The following subsections describe each of the services in more detail.

6.1 Data Storage Service

6.1.1 Conceptual

A software development effort deals with many kinds of things. The data storage service provides a means and a place for keeping those things or references to those things. The things are kept in the "data store" are called "objects" or "entities" (the term entity is used from now on). Features or facets about entities can be values associated with them and are called "attributes" or "properties".

Entities are created and stored. Creation brings into existence an entity with a distinct identity from all other entities. The storage aspect of the service means that the entity can be observed and examined after its creation (until it is deleted (if ever)).

The usefulness of this service is increased by the association of values with entities. The set of attributes associated with an entity may be fixed or updatable (as may be their values).

6.1.2 Degree of Understanding

The structure, operations, and constraints defined for a data storage service constitute a "data model". The study of data modelling 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 [13] [14] [15] [16] [17]. Brown [18] specifically addresses database support for software engineering. This service can thus be described in some detail.

6.1.3 Types

A data model may or may not have the notion of "type". There may be "entity types", "attribute types", and "value types". The existence of a type does not necessarily depend on instances of that type existing. Entities of the same type have the same set of attributes. Similarly, values of the same type belong to the same set of values associated with a value type. Examples of types are plans or programs.

Entity types may be related to one another (see 6.2). One way this is often done is a "subtype hierarchy" in which entity types are related by a subtyping relationship. The exact form of this relationship can be defined in a variety of ways but all are related to the basic form which says that all entities of a given type, T1, which is a subtype of type, T2, have all the attributes which entities of type T2 have. A term used in this context is "attribute inheritance".

Typing is not the only useful grouping of entities. Another way of grouping entities is to put them into a set perhaps according to some criteria which involves their type. These sets may be called "cover aggregations" and may themselves be entities. Examples of these may be a photography club or a convoy of ships, submarines, aircraft etc.

6.1.4 Operations

The basic operations on entities and entity types are create, read, update, and delete. An important point about update is that it cannot be simulated by a sequence of create and delete operations. This is because of the unique existence of entities. The update operation changes value of attributes. Update and delete may be replaced by a "new-version" operation (see 7.1). In other words update and delete may not be done "in place".

All operations are subject to the basic constraints of the data model definition.

There are data models which associate operations with entities and these can be related to the typing rules. The term used for models with these (and other) characteristics is "object oriented" [19]. In this case the subtyping or inheritance rules can be substantially more complicated and involve the implementation of operations (or "methods").

6.1.5 External

Because of the many relationships with other services in the reference model (see 6.1.7), the presentation of the data storage services is difficult to make general statements about. It may be that the primitive operations on types and entities will be visible to the applications. On the other hand, they may be well-hidden behind other related services which make extensive use of them. The basic operations may be built into compound services.

One main purpose of providing this service is that it enables many of the concerns to do with the detail of data storage and manipulation to be hidden from an application requiring such facilities.

6.1.6 Internal

Implementation strategies of the data storage service can 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 disc storage.

There may be other implementation strategies adopted which are more suitable for object oriented database management systems, and how those services are used in a CASE environment framework. It may be that the service interface will allow 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".

Note that there are alternative strategies to an all-pervasive data model such as: a fixed data representation which all parts of the environment framework use; or n-squared translators between all data representations; or n translators for a "common" data (or data type) representation. These are considered as alternative ways of implementing a data storage service.

6.1.7 Related Services

In some cases other services (e.g. configuration and version services) can relate to the data repository service in a layered fashion. Another kind of relationship is where the data storage service is used (often in conjunction with other services such as the relationship service) to store data and/or metadata for other services.

The data storage facility is directly related to many of the other services described in the reference model. In fact all the other services under the headings "data repository" and "data integration" are affected and affect directly the data storage services.

The term "data modelling" is more generally applicable to many of the services related to the data storage service.

6.1.8 Examples

Examples from real systems include the objects, attributes, and object types of PCTE+ [20, 21] and CAIS-A [22]; the entities, entity types, and attributes of Aspect; [23, 24] the objects and object types of ATIS; [25] and elements of the following models and systems: Damokles [26]; the relational model [27]; Taxis [28]; SDM(+) [29, 30]; and Iris [31].

6.2 Relationship Service

6.2.1 Justification

The entities in a CASE environment do not exist in isolation from other entities. The following are good reasons why not:

- There is a particular dependency between entities, e.g. object code is compiled from source code.
- A recorded relationship exists between two or more entities, e.g. members of a project or author of a document.
- A constructional relationship is useful to record, e.g. parts of a design or versions of a test.

6.2.2 Conceptual

The relationship service allows the definition and maintenance of "relationships". A simple form of a relationship can be a single (directional) "link" between two entities.

A relationship may be an entity in its own right [30]. The benefit is that one set of operations, terminology, and rules can be defined instead of maintaining a dual set for both entities and relationships. For example, the entity relationship model [32] treats entities and relationships separately while RM/T [33] treats relationships as entities.

6.2.3 Types

As implied by 6.2.2 relationships generally share the same features as entities in terms of typing. Thus relationships can be typed and have attributes. But the relationship concept can add further constraints such as "cardinality" (i.e. the number of entities allowed to participate in a relationship). And different kinds of relationships can be defined which offer differing semantics to the delete operation (and add constraints to the create and update operations). E.g. a relationship cannot exist unless the involved entities exist. This offers alternatives of rejecting a delete request, or "cascading" the delete to delete entities in a relationship if one entity is deleted.

6.2.4 Operations

The basic operations apply to relationships: create, delete, update, and read. A further useful operation is "transitive closure" which can be defined informally as the set of entities which can be reached from a given entity by following relationships of a given type.

6.2.5 External and Internal

The same comments concerning the internal and external dimensions of the data storage service apply to the relationship service.

6.2.6 Related Services

The relationship service is generally considered to be an intrinsic part of a data model. It can be considered as being well understood and is treated separately here since it can be provided without a complex data storage service (e.g. ACTIS [34]). Separating the topics also makes them both of a reasonable size to discuss.

Relationships offer a way to find entities by navigation without a query language (see 7.3). Foreign keys play the role of links in the relational model and links are not explicit in some object oriented systems. The transitive closure operation supports the formation of composite objects through relationships (see 7.2).

6.2.7 Examples

Example systems are the same as for the data storage service (see 6.1) plus ACTIS.

6.3 Name Service

6.3.1 Justification

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

The term "surrogate", introduced by Hall, Owlett, and Todd [35] captures the idea that every entity of the outside world is associated with a surrogate which stands for that entity 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, by allocating surrogates when the entity is introduced to the system. It is therefore always true that two entities are the same if, and only if, their surrogate values are identical. Surrogates are never re-used to ensure that events such as restoring entities from backup storage (see 6.9) or the archive service (see 6.8) cause no conflicts.

6.3.2 Conceptual

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

The name service may also support the additional concept of a "namespace". Any surrogate can be associated with a name which is unique within the context of a namespace.

6.3.3 Types

The type hierarchy may be used to enforce a naming policy. For example, nameable entities may be distinguished from those which cannot be named, or those which have to be named.

6.3.4 Metadata

Names and namespaces can be entities themselves.

6.3.5 External

The surrogate concept does not necessarily imply that surrogates should be either totally visible nor totally invisible to the external user. There can usefully be degrees of surrogate visibility:

- User-keys are still used extensively, and surrogates, hidden completely, simply help maintain the integrity of the system;
- Users are aware of an attribute recording surrogate values which can be taken advantage of in queries, but actual values are always hidden;
- All entities are named, and whenever a user would see a surrogate value, a name is presented instead;
- Surrogates are completely visible to the user, but they cannot, by definition, update, or in any way control, the values presented.

Within a CASE environment framework context, the third option is generally the most useful. However the fourth can be useful to the CASE environment framework developers or toolwriters 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 entity 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 [36].

6.3.6 Internal

The implementation of a name service has to take care that it does not become a system bottleneck.

6.3.7 Related Services

The name service does not provide access to entities (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 7.3). Where the name of an entity is clearly distinguished from the means of accessing it, data independence is not compromised and it is clear whether or not two different names refer to the same entity.

But the two services can be very closely related so that a name service reflects the access path to entities (i.e. a navigational approach). The PCTE approach is an example of this.

In general, the name service will be used by all the other services which present an external interface to the user. Otherwise, entities will be referenced by the services as surrogates.

6.4 Location Service

6.4.1 Justification

The requirement for distributed software development support is firmly established as a requirement for CASE environment frameworks. Distributed CASE environment frameworks have been constructed.

The essence of the problem to be solved is to enable all the services in the CASE environment framework to be available over a distributed collection of (potentially heterogeneous) processors and storage devices.

There are many aspects of distribution which CASE environment 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.

6.4.2 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 can maintain a logical model which abstracts some of the physical details so that e.g. collections of devices can 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 can be invoked. The unit of distribution may be a whole, a set of, or part of, an entity 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 7.6), nor what protocols exist for communciation between services and tools (see clause 10).

6.4.3 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 the one machine e.g. 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 e.g. because of network failure may leave a wide variety of choice for actions to be taken.

6.4.4 Metadata

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

6.4.5 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 can be made available for tools and users to make greater use of the networked resources according to particular applications.

6.4.6 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 those related concepts.

The name service (see 6.3) may well be closely related to the location service as the naming of distributed entities contains many problems. The query service, data storage service, and relationship service are also particularly closely related to the location service.

Clearly, the process support service (6.7) is related to the location service when it comes to the distribution of processes over a network. The message services (see clause 10) may use some of the location services extensively.

6.5 Data Transaction Service

6.5.1 Conceptual and Operations

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 semantics is that the transaction either succeeds in carrying out all the specified operations or restores the data upon which it was acting to a state as though the transaction never happened. Note that some operations may be irreversible. In particular, once output messages are sent they cannot be "unsent". Particular examples may be a "print" message, or a message to fire a rocket. In general, any external communication breaks atomicity.

Example transactions in a CASE environment 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 will have to succeed to achieve the final aim. If any fail, the CASE environment should not be left in a state with a half-processed manual, or a half-employed programmer.

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 6.9).

6.5.2 Operations

Planned successful termination of a transaction is achieved by the "commit" operation, while unsuccessful termination can 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.

6.5.3 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 CASE environment frameworks. It should be noted that a distributed system can have significant effect on what can 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 CASE environment frameworks.

6.5.4 External

It is considered beneficial that users and applications of this service are unaware of the complexity of recovery. The service provided in a particular systems has to be aware of the following points:

- The responsibility division between the service and what is using it has to be made very clear;
- It should be provable that the service really does offer the safeguards it claims.

6.5.5 Related Services

The services particularly related to the data transaction service are the concurrency service (6.6), the sub-environment service (7.6), and the task transaction service (9.3).

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

6.6 Concurrency Service

6.6.1 Justification

The Data Transaction Service is required to cope with the simple fact that things cannot be guaranteed to work all the time. This is independent of whether or not resources are being shared by more than one transaction and transactions are happening concurrently. The latter case brings more problems with it which are addressed here. The description of the data transaction service is independent of whether the system is being used by a single user or more than one. The concurrency service (note that this service would often be treated under the heading of "Transactions") addresses the extra considerations which have to be managed when more than one transaction can happen at one time.

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 7.6) and single-role transactions take place only within the context of a sub-environment. But even in this case, system metadata (see metadata service 7.4) has to be protected during concurrent access and update.

6.6.2 Conceptual

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 will read the same original design and then commit their respective changes. Unfortunately, the designer who commits some changes first will have those 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 "serialises" 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. task management or a tool).

6.6.3 Degree of Understanding

Concurrency control for traditional applications is well-understood, but it is not clear that the assumptions underlying its study in that context apply so well within a CASE environment framework.

6.6.4 Rules, Operations, and Data

The data associated with this service are transactions and the locks on data items. 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. And the detailed constraints on the acquire and release lock operations depend to a great extent on the type of locks defined. It may be that the following hold for certain types of locks:

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

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

6.6.5 External

The degree of visibility of the concurrency service will vary 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. more than one transaction waiting for each other to release locks which the other(s) hold(s)).

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.

6.6.6 Internal

A traditional approach to achieving serialised transactions is to "lock" entities or their attributes. A lock puts constraints on operations which can 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 data usage, granularity, and what is locked. Applying good or bad methods can have serious impacts on the degree of concurrency of transactions and the overall performance of the system.

6.6.7 Related Services

There are services related to the concurrency service. Clearly, support for concurrent processes is essential. This is provided by the process support (see 6.7) service. The data transaction service (see 6.5) is obviously closely related and the task transaction service (see 10.3) may use or simplify this service extensively.

6.7 Process Support Service

6.7.1 Conceptual

Just as the data storage service deals with the basic support mechanisms for managing data entities, so the process support service provides the basic support mechanisms for active entities. It provides mechanisms to support them while they are and are not executing and mechanisms to monitor them while they are.

A term used for a process which is not executing is "static context". Useful information which may be stored about a static context includes the type of locations it can run at, and whether or not it is interpreted.

There is useful information which can be maintained about a process which is executing. This includes: relationships to other processes (e.g. parent processes, or the process's interpreter); relationships to concepts in other framework services such as associated schemas, tasks, or execution site; and priority and status of the process (e.g. ready, waiting, running, terminated etc.).

A further concept incorporated in the process support service is that of a "foreign process". All the services of the development environment are able to execute, to a greater or lesser extent, on the hardware and software which is considered to be the environment framework platform. But the software under construction within the environment may not run on such a platform. Examples could be microprocessor control systems (which are not powerful enough to support all of the functionality of the environment) or operating system development (which is too risky to test on the same platform as the environment) or software for a different hardware architecture from that on which the environment runs.

The process support service provides a means of communication and control between the development environment framework and the target environment. This is a means of transferring software, execution and debugging results, and of controlling execution. It may be a service of considerable complexity as the target environment may be a distributed heterogeneous communicating network.

One phase of the service is the definition of what should be monitored in the target environment by the development environment framework. Another phase is the transfer of the static context (which may include compilation and loading). And the final phase is the control and monitoring of the execution or interpretation of the process.

A further use of foreign processes can be made to provide an Open Tool Interface (OTI). In this case, the emphasis is on enabling the services of any underlying operating system to be made available to framework users. The trade off is typically reduced use of the framework services in exchange for easier migration of existing tools.

6.7.2 Operations

Typical operations of the process support service include: start (asynchronously or synchronously); terminate; suspend; and resume. And there are a set of operations to transfer, control, and monitor foreign processes.

6.7.3 Types

An environment framework may offer further control of processes by making processes be of a process type.

6.7.4 Metadata

It has been shown (at least in theory) that both static and dynamic information about processes can 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.

6.7.5 Related Services

In an object oriented model, the process services would be addressed in conjunction with the objects in the data storage service. In non-object oriented systems the services may be less closely related but there can still be "binding" issues.

The process support service can be used by most of the other environment framework services to enable their execution, but the task management services (see clause 9) may take effective control of much of the process service.

6.8 Archive Service

6.8.1 Justification

The amount of data which will be associated with a project in a CASE environment may be more than can 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. There can be a service which allows online information to be transferred to an offline form.

6.8.2 Conceptual

The archive service carries out a mapping between the online storage and offline storage of entities. A placeholder may represent the entity in online storage, while the entity is archived offline.

6.8.3 Operations

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

6.8.4 Rules

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

6.8.5 Metadata

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

6.8.6 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.

6.8.7 Related Services

Services related to the archive service are the data transaction service, the backup service, and to some extent all data repository services and some data integration services. In particular, the version service could offer a pointer to the most convenient units to archive. Also, the query service (see 7.3) will have an interest in how archived entities are marked and can be retrieved. The data interchange service (see 7.7) may impact the formats used by the archive service.

6.9 Backup Service

6.9.1 Conceptual

It is the backup service which can be relied on to restore the development environment to a consistent state after media failure. It 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 can be logs in the dump itself.

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

6.9.2 Metadata

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

6.9.3 External

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

6.9.4 Related Services

Services related to the backup service are the data transaction service, the archive service, and to some extent all data repository services.

The state monitoring and event monitoring services (see 7.5 and 9.5) can 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 6.4).

7. DATA INTEGRATION SERVICES

The data integration services enhance the data repository services by providing higher-level semantics and operations with which to handle the data stored in the repository.

7.1 Version Service

7.1.3 Justification

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.

7.1.2 Conceptual

Versioning is based on a simple concept: interest in aspects of an item's former state (including relationships with other, perhaps versioned, entities). The variations on this simple theme can generate many complex scenarios. Change throughout development has to be managed in CASE and the inclusion of versioning as one of the means of achieving this is thus justified in a reference model.

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 generalised version models [37, 38]. Just some of the choices available are described here.

There has to be the concept of a logically single entity which has a set of attribute values. Now creating a new version of the entity identifies something new. The version may or may not have the status of an entity, but it is something distinguishable with an associated set of values. The reason for which another state of the same logical entity 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 bugfix of the first. The distinction between variant and version is subject to the possibly differing perception of people.

To add to this complexity, the management of relationships between entities 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 their own version model or different models for different types of work. For example, one hundred people on an operating system development project require significantly different support from six people writing an accounting package. CASE environment frameworks may have version facilities which are parameterised or flexible enough to allow this (e.g. environments which come as a kit of parts).

7.1.3 Operations

Just as there is typically a data model associated with a data storage service, the version service can 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 can support more than one source for the creation of a merged version.

7.1.4 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 can be utilised to provide sophisticated transaction mechanisms.

7.1.5 Internal

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

7.1.6 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 configuration service which can then allow versions of composite entities and configurations of differing versions. Other closely related services are the name and query services.

7.1.7 Examples

The set of systems which address versioning includes: PCTE; DSEE [40]; ATIS [25]; NSE [39]; Aspect; and Damokles [26].

7.2 Configuration Service

7.2.1 Conceptual

The problem is that one wants to look at and operate on a "thing" as a single item sometimes but at other times consider it as a (structured) collection of things. The configuration service offers the facilities to enable this to be done.

Terms used for the collective whole are "configurations", "compound entities", or "composite entities".

One particular example of this kind of service is often found in software development environments. This is where a software system has to be transformed from a set of source code modules into executable code. The rules or procedures which have to be followed to achieve the transformation are considered as part of the configuration service. The general case of the above is a composite entity which is not only defined by its constituents and their relationships, but also in terms of transformations of other (possibly composite) entities.

Composite entities are not only found in CASE environments in the form of software configurations. Documents or tests (e.g.) have components too. Similarly, they may have to be transformed to generate a whole document or test.

The description of how composite entities can be defined and operated upon is called the "configuration model". General models for configuration management are described in Heimbigner [41] and Dillistone [37]. Many of the statements applicable to the version service (see 7.1) are applicable here and the reader is referred to that section.

7.2.2 Types, Instances, and Metadata

The configuration service may enable definitions of configuration types which can later be instantiated. There may be several instantiations of the same configuration type. The configuration service may keep track of configurations and their instantiations and configuration metadata.

7.2.3 Internal

One popular way of representing configurations is (directed) composition relationships between parts. The whole is then the transitive closure of the part over the composition links.

7.2.4 External

In dealing with composite entities, one has to be very clear at what level the discussion is being held. This is even more important if the configuration model is built "on top of" the data storage model. There will be different kinds of configuration models and a given support environment framework may (have to) support more than one of these.

7.2.5 Related Services

The configuration service can be closely related to the version control service. In particular, composite entities may be versioned.

7.3 Query Service

7.3.1 Conceptual

The query service is an extension to the data storage service's "read" operation. In the general case, it returns sets of values retrieved from sets of entities.

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

Once the entities are chosen, some manipulation of the chosen values may also be carried out. For example, to find the average age of programmers.

7.3.2 Degree of Understanding

The study of query services, query languages, and query optimisation is an ongoing research topic. Some environment frameworks will use well-understood techniques while others may use the latest research results.

7.3.3 Metadata

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

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

7.3.5 Internal

The implementation of the query service and query languages is closely related to the implementation of the data repository services. In many cases simple implementation of query resolution will result in a very inefficient service (particularly in the distributed case) and optimising parts of the process can be vital. The more information available to the query service concerning the expected and actual use and location of data, the more sophisticated optimisation techniques can be used.

7.3.6 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 will depend on the degree to which the basic data of other services is stored by the data storage service.

There will be 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 7.4) makes this a very feasible proposition.

The query service is intimately connected with nearly all of the data repository services (apart from data transaction and process support). Apart from its general usefulness for enabling querying for other services, it is of importance to the sub-environment service.

7.3.7 Examples

Examples from real systems include: Aspect's relational algebra and the PACT query service (DQMCS) built on top of PCTE.

7.4 Metadata Service

7.4.1 Conceptual

"Metadata" is data about data. Examples would be data about the types, attributes, or relationships held in a database. Names for such data (or where it is held) include "data dictionary", "catalog", or "schema definition". The metadata service provides control and maintenance of metadata.

7.4.2 Degree of Understanding

The study of metadata and its maintenance is ongoing [42]. Updating types and other forms of schema evolution are a difficult problem. For example, what happens to existing instances of an altered type? As new data models incorporating complex entities and versions are developed, so metadata research develops.

It is relatively straightforward to offer metadata query operations, but updating metadata has the effect of changing the structure of the database and ensuring consistency in this case can be much more difficult. It is for this reason that while existing environment frameworks often offer a metadata query service, metadata updating is done using different functions to those provided by the data storage service. For example, instead of creating an entity of type "attribute", there may be a create-attribute function. Object oriented database management systems (e.g. IRIS [31]) tend to be more uniform.

7.4.3 Operations

The set of operations the metadata service supplies is query, update, (perhaps delete), and identification of the building blocks of the data model in use. Operations which may be available deal with changing a type by adding or removing attributes or changing the type of an entity.

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

7.4.4 Internal

The drawback of a self-referential system is that it may constrain the methods of implementation (on the other hand it may allow the implementation of the metadata service to use the data storage service extensively), and it can lead to some rather tricky problems (e.g. what are the implications of creating a new version of an entity type?).

7.4.5 External

The specific data model (the term specific is used here to distinguish data models of particular environment frameworks from the more general style of data model (such as relational or networked)) can be presented in a different way from the data storage data model or it can be presented in the same way. If the latter is the case, then the data storage service will at least appear self-referential. The benefits of this are that the user/application has fewer concepts to deal with, a consistent interface (e.g. the query service can now return information about data and metadata), and a place for general unification of ideas.

One particularly interesting form in which the metadata service is presented is that of a "Data-Definition Language" (DDL). This enables the structure and instances of a database to be defined in a human and machine-readable form. Various services can be offered in association with this (e.g. validity checking). The language may also be useful elsewhere. For example, in the exchange of data and metadata between environment frameworks.

7.4.6 Related Services

The services related to the metadata service are all the data repository services as well as the sub-environment service (7.6). In particular, since the structure of the data in an environment will evolve, the metadata may be versioned.

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

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

7.5 State Monitoring Service

7.5.1 Conceptual

The state monitoring service enables the definition/specification of database states and state transformations, and actions to be taken should these states occur or persist. This is sometimes called "triggering". Its usefulness lies in the ways it can help automate the modelling of real-world situations.

The definition part of the service can 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, can take several forms. It may be a notification to the person/role or process which defined the state, or to some other process or person described in the definition. It may, alternatively, prevent that 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.

7.5.2 Degree of Understanding

The efficient implementation of detecting occurrences of the states which are defined is the crucial problem, and still very much a topic for research.

7.5.3 Operations

The kind of operations at the state monitoring service's interface would be: create; update; delete; and query a state/transformational definition. Set and unset operations control whether the checks should be applied or not. There are also separate operations to manipulate the actions associated with a state/transformation. Attach and detach operations (dis-)associate a state with a set of actions.

7.5.4 Internal

In general, the definition of static states can be done with the facilities provided by the query service (although other ways are possible), but the definition of dynamic transformations is a much harder problem in general.

To be totally consistent and general, all types and instances of data (and transformations) should be supported. Finding efficient implementation techniques may initially mean restrictions on this general service.

7.5.5 Related Services

The state monitoring service is related to many other services, both to help it support the definition of states and their transformations, and to ensure that appropriate actions are carried out.

7.5.6 Examples

Examples of the state monitoring service can be found in ACTIS and PCTE.

7.6 Sub-Environment Service

7.6.1 Justification

Since the real software enterprise is not conveniently or usefully understood as a single complex system, the CASE environment framework modelling it can provide support for breaking down the whole environment into sub-environments in which parts of the overall project can be carried out. The basis of division may vary across the CASE environment. For example it may be according to ownership; to security clearance; or relevance to the current task. The CASE environment framework may provide all or few (or even none) of these.

Sub-environments can also be used as the basis of support for the migration of pieces of work through (pre-defined or process-defined) states.

7.6.2 Conceptual

A whole environment can be considered as being made up of a set of data items and a set of operations on those data items. The sub-environment service consists of several concepts.

The first is the definition of a subset of the data and operations in the whole environment. The most general form of definition is an intensional one which defines some of the properties of the data and operations to be included in a particular sub-environment. Thus there are also binding issues involved. Whenever a sub-environment is accessed, it has to be determined which data and operations are actually in that sub-environment. The binding of data and operations to a sub-environment may take place once, or the binding may be re-evaluated dynamically. Data or operations may be allowed to be in more than one sub-environment at the same time.

The second concept in the sub-environment service is the definition of a two-way transformation between the data and operations in the underlying environment and a new set of derived data and operations of which the sub-environments actually consist. The transformation is only applied to the underlying data and operations considered to be in the sub-environment (according to the definition described in the first concept above). The transformation has to be a two-way mapping so that changes to the data caused by operations at the sub-environment level can be reflected in the underlying data. The transformation may be defined exclusively in terms of the operations in the data repository services, or user-written transformations in some general-purpose programming language may be allowed. The binding issues described above are also a concern here.

Thus a sub-environment is a set of data and operations derived from a defined subset of underlying environmental data and operations by the application of a two-way transformation. Hierarchies of sub-environments can exist, in which case the underlying data and operations are themselves derived. Some notion of self-consistency of a sub-environment (e.g. all the types referred to by data and operations in the sub-environment have to be available in the sub-environment) may be imposed by the sub-environment service.

The third major concept of the sub-environment service concerns the exchange of derived data and operations between sub-environments. In addition to maintaining the self-consistency of sub-environments, the control of transfer between sub-environments offers environment builders the opportunity to control software development processes in a natural way. With this third concept, the contents of a sub-environment are extended by the set of (derived) data and operations which it imports. The data and operations which are exchanged between sub-environments are controlled by a protocol enforced by the sub-environment service. A particular example of this may be the evolution of a document from a "draft status" sub-environment to a "reviewed status" sub-environment.

7.6.3 Degree of Understanding

The sub-environment service is a complex one both in itself and in the relationships it has with other services. Most environment frameworks provide a subset of the concepts described here (see 7.6.11 for some examples). Different frameworks provide different subsets in different ways. Thus experience is being gained in using elements of the sub-environment service, but there is still a lot to learn about how the elements fit together and can be related to other framework services.

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

7.6.4 Types

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

7.6.5 Operations

The set of operations supported by the sub-environment service could include the following: create, update, delete, and query sub-environment definitions (including the transformations); instantiate a sub-environment (to cause the bindings to be made); and a set of operations defined by the exchange control protocol such as publish, acquire, and release.

7.6.6 Instances

Instances of sub-environments are the bound sub-environments.

7.6.7 Metadata

Since control over the exchange of the contents of sub-environments is so important, there can be information held by the sub-environment service regarding the definitions of sub-environments and the relationships which exist between them.

7.6.8 Internal

One simple rationale for environment subdivision is the performance gains which can be achieved through simplifications of the distribution strategy (in fact making it mirror more closely the way people typically organise 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 [43]). It is important that the conceptual and internal aspects of sub-environments are clearly separated in descriptions and discussions.

7.6.9 External

The sub-environment services are ones which may well have to be visible to CASE environment framework users. The obvious ways this could happen are sub-environment definition languages, and elements of the protocols which will exist to control the exchange of information between sub-environments. Significant reuse of sub-environments could be envisaged. Whatever external representations are chosen may take this into account.

7.6.10 Related Services

The query service can be used in the intensional definition of a sub-environment.

The decisions about when to do the various bindings associated with the sub-environment service are closely related to the models of work supported by the task management services. For example, a particular task (or task type) definition may be bound to a particular sub-environment definition, and when the task starts the particular bindings in the sub-environment may be made.

The control of exchange between sub-environments can 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 will mean it is a major user of the data repository services.

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

7.6.11 Examples

Examples from real systems include: the Schema Definition Sets (SDSs) and working schemas of PCTE; the "role databases" of ISTAR [44]; the "workshop" and "studio" processes of the Workshop System [43]; and Perspective's "domains" [45].

7.7 Data Interchange Service

7.7.1 Justification

The data repository and data integration services combine to handle data within a CASE environment framework. But there is a need to be able to exchange data between 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 the earlier versions of 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 data has to be represented in a form suitable for transfer to and from portable storage media.

7.7.2 Degree of Understanding

This service is not a difficult one to implement or define. The problem lies in reaching agreement on standard formats.

7.7.3 Conceptual

The data interchange service offers two-way translation between data in the data repository and some format for that data which can be stored on portable storage media.

7.7.4 Operations

By definition, the basic operations provided by the data interchange service are read and write.

7.7.5 Related Services

The data interchange service has to translate the data of all the other data repository and data integration services. It is not closely related to any other services.

7.7.6 Example

CASE/EDIF [46] is an example standard in this area.

8. TOOLS

8.1 Conceptual

The set of environment framework services exist partly to support one another, but mainly to provide a useful interface which can be used in building facilities which 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. This is because the term "tool" is currently in general use for a whole packaged software product which often consists of a set of co-operating functional elements. A functional element is a piece of software which calls upon the services provided by the CASE environment framework and/or other functional elements. Functional elements "plug in" to the CASE framework and communicate with the services provided by the framework via the message services (see clause 10).

A functional element does not necessarily exist in one place. It may be distributed.

A "composite functional element" ("composite tool" or "toolset") can 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 CASE framework.

In addition to using the services from all the other parts of the framework, the tool part of the framework provides 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 which acts as a layer between the tool and the framework.

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.

In some cases a tool may already have facilties 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 an environment 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) those services become technically indistinguishable from services provided by the framework, since all are accessible in the same way (through the message delievery 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 specialise 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 specialised 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 8.6 for a discussion of integration).

8.2 Types and Instances

Any description of an environment's tools according to this part of the reference model should at least identify which tools exist. Of course, a more detailed review would describe the purpose and facilities of each tool.

This reference model chooses not to provide a framework in which to classify (or type) tools (e.g. management tools, or requirements analysis tools etc.) as that goes beyond the scope of the current charter and intentions of the reference model.

Tools could be usefully categorised as "vertical" (i.e. applicable at just one stage of the development of a software system) or "horizontal" (i.e. applicable throughout the software lifecycle). An example of the former is a compiler, while an example of the latter is a project management tool.

8.3 Internal

One can envisage two extremes of tool implementation. The first being a piece of software which, to some degree, automates the invocation of services which exist within the environment framework. In this case, the tool stores (almost) no data within its own code, and provides (almost) no processing except that which can be called upon the environment framework services to provide. Only tools written for the particular environment framework will lie at this extreme.

The second extreme is where a tool calls upon very few (or even none) of the framework's services. In this case the tool keeps "private" data and "private" methods (or process). Encapsulated tools will lie at this extreme.

The implementation of many tools will lie between these two extremes.

8.4 Related Services

Tools use all the the services a framework provides for them. Tool encapsulations may make particular use of the message delivery services and the user interface services.

8.5 External

A consequence of tools "plugging-in" is that it cannot be assumed by a service or another tool that a particular tool is present in an environment. Of course, this is a rather strict statement which is really saying something about the difference between a tool and a service. In a software system such as a CASE environment, adding a piece of software which is invokable, such as a tool, tends to make the tool indistinguishable from the environment framework itself. The identification of a piece of functionality as a service is somewhat like saying that it is a tool which will be available in all environments (of this kind).

8.6 Tool Integration

This section discusses the question of integration; what it means, what kinds of integration exist, and how different levels (or degrees) of integration can be defined.

Three kinds of integration are distinguished. They are data integration (covered by the data repository and data integration services in the reference model), control integration (covered by the task management and message services), and user interface integration (covered by the user interface services). It will be seen from their descriptions that some relationships do exist between the different kinds of integration, but it is useful to be able to discuss them orthogonally. Integration of one kind does not necessarily depend on the existence of integration of another kind.

Within each kind of integration, three levels are identified: the interoperability level; the partial-integration level; and the full-integration level. In this case, full-integration depends on partial-integration, and partial-integration depends on interoperability. Degrees of interoperability and integration are allowed. That is, two or more functional elements (tools or services) may lie somewhere between two of the defined layers for a particular kind of integration.

Thus, to describe to what extent two or more functional elements are integrated, one has to describe their degree of integration of each kind. Functional elements are only fully integrated when they reach that level for each kind of integration.

Integration of tools (and services) is not simply achieved by providing a good set of services. There are many factors, and one of these is a guide for the toolwriter explaining what the services are intended to achieve and suggesting the "common" ways to use them. An example is the PACT Tool Writer's Guide [47].

8.6.1 Data Integration

Two or more functional elements are "data-interoperable" when they are connected via a data transfer mechanism or share data in a data repository.

Two or more functional elements are "partially-data-integrated" when they are data-interoperable and have a mutual understanding of the data formats.

Two or more functional elements are "fully-data-integrated" when they are partially-data-integrated and have a shared understanding of the meaning of their data and metadata.

8.6.2 Control Integration

Two or more functional elements are "control-interoperable" when they can communicate events to each other.

Two or more functional elements are "partially-control-integrated" when they are control-interoperable and can understand the information connected with events. This relates to their level of data integration.

Two or more functional elements are "fully-control-integrated" when they are partially-control-integrated and cooperate to behave in a rational manner in following a plan or reaching an objective.

8.6.3 User Interface Integration

Two or more functional elements are "UI-interoperable" when they can be used on the same display devices.

Two or more functional elements are "partially-UI-integrated" when they are UI-interoperable and use similar visual and behavioural primitives.

Two or more functional elements are "fully-UI-integrated" when they are partially-UI-integrated and are understood by a user to offer similar look and behaviour. This relates to their level of control integration.

9. TASK MANAGEMENT SERVICES

The focus of task (or activity) management services is the provision of services which allow the CASE environment to be task-oriented; to support process modelling of the software enterprise in which the environment is being used. The terminology, "process programming" is also used [48].

The concepts in these services are refined in some detail to cover aspects of procedural definition; tools and their execution; and the pre- and post-condition method of activity definition. The reference model does not describe particular activities which often occur in a CASE environment. Other aspects which are discussed are activity hierarchies, both in terms of what is planned to happen within the CASE environment, what is executing, and a history of what happened.

There is no doubt that this is a complex set of services (at least at the same level of complexity as high-level programming languages). A complete internal view of the system providing the concepts described above may have to descend into details of services provided in other parts of the reference model. As the subject becomes better understood (it is currently very new) support will be provided in a well-structured way among the services.

The required external views of what is going on within the CASE environment (and hence the projects it is supporting) could be many and varied, and depend to a great extent upon the concepts supported by the CASE environment framework and the mechanisms it employs to provide them.

Task management and its support is a subject still in its infancy. The degree of understanding of all these services should be considered very low in comparision to the data repository services.

9.1 Task Definition Service

9.1.1 Conceptual

It must first be stated what a "task" consists of before the possible elements of a "task definition" can be listed. A task instance is first considered, rather than a generic task description (i.e. a task type).

A task has start and end points. The task description may therefore specify under what conditions the task is to start and/or complete. There are two kinds of ways of specifying start and end points (which may coexist). The ways are prescriptive and descriptive. The first case is rather like in a procedural programming language where one statement follows from the preceding statement (perhaps according to some conditions). The order of statements is written down beforehand (as a program). In this case a task plan (or project plan) exists as a task "program".

In the second case, the descriptive one, a set of preconditions (or rules) act as a guard to starting a task. By definition, once these preconditions are met, the task commences. An equivalent set of post-conditions can exist for a task's completion.

A task has to be carried out by a role or roles (see 9.7). Therefore, one element of all task descriptions is the set of roles assigned to carry it out. This set may remain empty until the task is assigned to a role. In general, there may be a many-to-many relationship between tasks and roles.

A task may execute in some "context", using some resources. Elements of a task description may therefore be a statement of what resources it can use. Of particular interest as resources at this level are: tools; sub-environments; and other tasks. Let us describe their importance.

A task exists to change the overall state of information in the CASE environment. The person effecting those changes (with the support of the CASE environment) is helped in coping with complexity by working with a tailored (and perhaps restricted) set of data, tools, and services. The sub-environment service offers this extra support and context in which to carry out a task. The person or role defining the task may not be in the best position to define the supportive context in which to carry out the task. Some tools may be identified as part of this context.

In general, the granularity of tasks can range from tasks "to do a whole project", to those at the level of "fix this bug". For this reason, a task may be manageable as a whole unit and the task definer may wish to "refine" the task definition to describe "subtasks". Just as subroutines in programming languages share almost all the properties of programs, so subtasks can share all the properties of tasks. The exact definition of the relationships between parent and child tasks is very much at the discretion of a CASE environment framework designer, although rules may be expected such as a child task's sub-environment must be a subset of its parent's sub-environment, or, a parent task cannot complete until all its child tasks have. From one point of view, a tool may be considered as a subtask.

9.1.2 Operations

The task definition service has to offer the ability to create, update, and delete task descriptions. Also, as many tasks will have similarities between one another, there is scope for task-types, and a set of task instantiation operations in a CASE environment framework.

In all CASE environments there has to be an initial task, and hence an initial task definition (or else no task could ever occur). The basis of the initial task will be a context in which further tasks can be described in order to develop a project plan (there is perhaps scope for manipulation of high-level plans and plan-types in a process management service. This is not covered by this reference model).

9.1.3 Internal

The internal dimension of the task definition service is one of the several task management services which can be greatly supported by the data repository and data integration services.

9.1.4 External

One external dimension of this service may be a (perhaps very complex) process programming language. An effective user interface to this appears a crucial factor in its development.

9.2 Task Execution Service

9.2.1 Conceptual

The task execution service provides what is necessary to control and support the execution of tasks. The sophistication of task control may range from low level (e.g. a simple ``shell'' invocation) to a much higher level (e.g. based on a comprehensive process programming language). The definition of tasks is going on while other tasks are executing. There even has to be scope for a task's definition to change while the task itself is executing. Change is a fact of life in software development and the support environment framework has to be able to manage it. This puts considerable pressure on the internal dimensions of the task execution service.

9.2.2 Internal

During the execution of tasks, tools are run (or may be treated as subtasks). It can be seen how effective it can be in existing environments when tools can communicate, be composed, and synchronise. It has yet to be established if these concepts are as effective at the task level, but the task execution service could provide them for tools at least.

9.2.3 External

The external dimension of the task execution service has to provide up to date information on the state of tasks. Such information revealing the progress of a project can be important to all members of the development team, but will benefit managers in particular. The use of this information by tools dealing with for example, estimates, or critical path analysis, can also be facilitated.

9.2.4 Related Services

A very important use of the task execution service may be made by tools invoking the services of other tools. Tools may directly control the execution of other tools or there may be a more sophisticated control mechanism in place.

9.2.5 Examples

Task definition is an important part of IPSE2.5 [49].

9.3 Task Transaction Service

9.3.1 Conceptual

An investigation of process control within an CASE environment framework [50] emphasised considerable differences with traditional concepts of transactions (see 6.5). The problems are specifically addressed by a number of researchers (e.g. [51, 52, 53, 54]).

A conventional transaction is defined as a sequence of primitive operations which 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 transaction started. The concept can be extended to a nested set of transactions.

The transaction idea is useful within a CASE environment framework. For instance, a tool can be thought of as a sequence of either primitive operations or nested tools which must all successfully complete or leave the information base unchanged. But there is a more important situation to model. That is, there is a task which, when appropriate, should be carried out to achieve some development within the CASE environment. It will probably not be possible beforehand to prescribe exactly how that development should be carried out in terms of a sequence of operations, but there will be a way to describe when the aim has been achieved.

Such tasks 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 tasks has four implications for the control mechanism within a CASE environment framework:

- Work performed within a task should not necessarily be discarded if a failure occurs;
- It is unreasonable to lock the information used by a task 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 task the database may have to go through what would be considered externally as an inconsistent state in order to satisfy a post-condition.

9.3.2 Operations

So much research is yet to be done in this area that no particular set of operations can yet be included in this reference model.

9.4 Task History Service

9.4.1 Conceptual

During a software engineering project, much useful information can be obtained by knowing what has happened earlier in this project or even what happened in previous projects. For this reason, one of the responsibilities of the task history service is to record information about the execution of tasks. Its other main responsibility is to make that information usefully available.

The task history service can consist of three phases: definition of the types of information recorded about tasks (which may only be available in sophisticated environment frameworks); the actual recording of information about task instances and their execution; and the answering of queries about tasks executed.

9.4.2 Internal

One way of implementing the task history service is where the data repository and data integration services are used to model information about task execution. This enables a query service to be easily provided (or even be available by default).

9.4.3 Related Services

It is the task execution service which uses the create and update facilties of the task history service. This is because the task history service aims to provide an accurate record of what has happened. The query facilities can be made more widely available.

The task history service has many similarities to the audit and accounting service (see 9.6).

9.5 Event Monitoring Service

9.5.1 Conceptual

Just as the state monitoring service (see 7.5) allowed the definition of states to enable actions to be triggered, so the event monitoring 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 to be taken by the particular CASE environment framework designer. One would however expect the granularity of events which can be defined to, and detected by, the event monitoring service to be at least at the level of initiating, interrupting, or completing tasks.

Event definitions may be be specific such as: when designer C finishes her next specification; or more generic (specifying a type of event): each time a document is approved by the quality assurance department. The ways an event definition can be expressed are dependent on the external presentation of tasks made available by the task definition service (see 9.1).

9.5.2 Operations

The kind of operations at the event monitoring service's interface could be: create; update; delete; and query an event definition. Set and unset operations control if the checks are to be applied or not. There may also be separate operations to manipulate the action definitions associated with an event.

Attach and detach operations (dis-)associate sets of actions with events.

9.5.3 Internal

The efficient detection of events which match any of those defined is an interesting part of the internal aspects of the event-monitoring service and the task execution service.

9.6 Audit and Accounting Service

9.6.1 Conceptual

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 those costs with those they should be debited to.

There are three relatively distinct phases provided by this service. In phase one, it is stated what should be recorded and what resources are costable (who costs should be assigned to, and the exact cost can also be defined here, or left to the third phase). Phase two is the recording process. And the third phase makes the results available.

These phases can be explored to greater detail. A number of points associated with implementation methods and related services are made here.

9.6.2 Internal

In phase one, it may be that the audit and accounting service "knows" enough about the system to allow an application to specify explicitly only high level requests while the service deduces some implicit things which will have to be recorded.

Phase one may not exist at all. The system may have everything recorded by this service and the third phase is used to specify by queries what information is required.

9.6.3 Related Services

Even with the ordinary three phases, the third phase may still make extensive use of the query service.

The other closely related service (note that all services have some relationship, that of potentially being recorded) would be the state monitoring service. This could, for example, be used to notify an authority when the accounting figures show an application has exceeded its use of a service. This would be particularly applicable if an environment was communicating with an "outside" service such as a library database.

There may be a relationship with "task histories" (see task history service, 9.4).

9.7 Role Management Service

9.7.1 Justification

In a software development project, the people involved typically have job titles or "roles" which they are employed to play throughout the project (e.g. as "system analyst", "technical architect", or "programmer"). In general, the mapping between individuals and roles can be many to many and can (and typically will) change throughout the project's lifespan.

The possibility of a definition of a standard set of role types is one which this reference model notes but does not address.

9.7.2 Conceptual

The 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 which 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 can carry out the programming role".

9.7.3 Types and Instances

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

9.7.4 Operations

The operations the role management service could provide include: create; update; and delete: roles; people; and the relationships between them. It may also be able to support the definitions and applications of roles such as "Person E should never be a programmer".

9.7.5 Internal

The internal dimension of this service is one which appears could be a fairly simple application of the data repository and data integration services.

9.7.6 Related Services

Roles relate very closely to security and the sub-environment service.

9.7.7 Examples

ISTAR [44] has a role database.

10. MESSAGE SERVICES

In many of the previous service descriptions it can be seen that many services have a set of related services which are critical to their proper functioning. Tools have to carry out extensive communication with the set of framework services. There is therefore a requirement for a service which supports managed communication between large number of elements of a populated environment framework.

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

Particular examples of the types of services which can 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. Another kind of possible service is a tool registration service which brings a new tool (instance or class) into the environment.

Some systems which have message services are: ESF, ACTIS, and SoftBench.

10.1 Message Delivery Service

10.1.1 Conceptual

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

There are three fundamental classes of message servers generally used to accomplish these:

1. "point to point" (in which the sender identifies the recipient(s));
2. "broadcast" (in which the message goes to all tools and services); and
3. "multicast" (in which some selection mechanism identifies the tools and services which will receive the message).

If the message service employed does include multicast message delivery, 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 services of the event monitor); or the interest may be in the data contained in the message (e.g. all messages referring to my objects); or various combinations of these.

10.1.2 Types

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

10.1.3 Operations

The exact set of operations on messages depends on the protocol chosen. Apart from create, delete, and update, there may be send, receive, acknowledge, reply, and ignore.

There can also be registration and de-registration operations 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.

10.1.4 Metadata

The message delivery service handles messages for the message services themselves. These may be handled differently from messages between tools and services.

10.1.5 Related Services

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

10.2 Tool Registration Service

10.2.1 Conceptual

There are two aspects to tool registration: making the tool known to the message services; and having the tool register interest in receiving certain types of messages. The tool registration service deals only with the former, since the latter is covered by the message delivery service.

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, data exchange formats, and identification mechanisms etc.

The tool registration service is also responsible for handling the process of tools leaving the environment framework. This may involve notifying (or even requesting permission from) the services and tools making use of the tool which is leaving.

10.2.2 Operations

Example operations for the tool registration service include register and unregister.

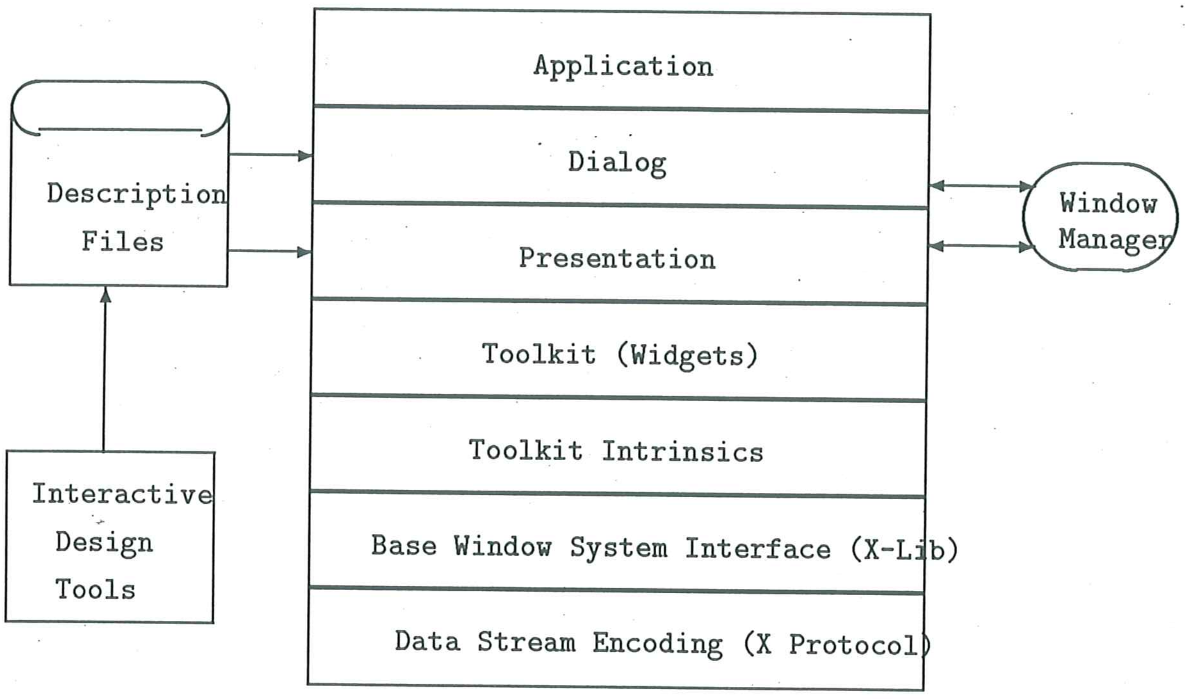

Figure 4 - Layers of the User Interface Reference Model

11. USER INTERFACE SERVICES

The CASE reference model offers a summary description of an existing layered reference model for user interfaces. The model summarised here, and shown in figure 4, is taken from OSF's Rationale Document [55] which is adapted from a draft User Interface Reference Model developed by NIST (National Institute of Standards and Technology), which in turn was based upon a framework developed by X/Open.

11.1 Conceptual

The Data Stream Encoding (e.g. X Protocol) layer is the "physical layer". It includes the implementation of the screen and input managers.

The Base Window System Interface layer gives a coherent set of primitives to manipulate "windows" and graphics. The X library is at this level.

The Toolkit Intrinsics layer concerns the general framework for defining user interface entity types. It provides a general model for user interface manipulation. The Xt Intrinsics is at this level.

The Toolkit layer provides a set of User Interface Entity types (widgets and gadgets), which have a defined and useful behaviour. The Motif toolkit and the OPEN LOOK X Interface are at this level.

The Presentation layer provides the ability to organise in a flexible and meaningful way the instances of the toolkit entity types needed by applications. Form management systems (e.g. the FORM widget of OPEN LOOK) are at this level as is the UIL (User Interface Language) of Motif. Interactive tools which support the design and capture of the general layout of the contents of a window are at this level.

The Dialogue layer handles the synchronisation of the different possible operations available to the user, which should be done in an application independent way. User Interface Management Systems (UIMSs) UIMS are at this level. Dialogue languages can be defined to handle this task.

The Application layer includes the application code and the mechanisms by which the application communicates with its User Interface.

It is clear that this layered reference model addresses only the issues of providing generic UI support for applications or tools. It does not have a place or dimension in which to discuss how the various services provided by an environment framework should be made available through a user's screen or keyboard. Perhaps a link needs to be made between the presentation and dialog layers with the semantic data model or sub-environment management services of the CASE environment framework reference model.

The user-interface reference model does not indicate which services of the framework it requires to provide user interface services.

The tool-centered approach of the user-interface reference model may not be completely satisfactory to discuss all the user-interface concerns within a CASE environment framework in connection with the services of the whole environment. However the CASE reference model currently has no proposal in this area.

Most CASE environment frameworks which have addressed user interface integration have adopted the general standard solutions.

11.2 Operations

Example operations include create-window or click mouse. There are many more which are not discussed here.

12. SECURITY

The CASE reference model regards security as a service which crosses many of the boundaries of the reference model divisions. There exists a reference model for security produced by ECMA TC32/TG9, described in [56, 57] and summarised by Cole [58]. A very brief overview of that model is given here.

The security model is based on the concept of "security information" and a set of security services for providing, mapping, and utilizing the security information. It is implementation independent, and independent of "security policy", and allows different "security domains" and administration in an open distributed system. Both "mandatory" and "discretionary" security are covered by these services.

Security information is generated by specific security services in response to requests which have been authenticated. The security information is then used by other security services to authorise actions or provide decisions to allow or disallow specific activities.

There are three classes of security services which are now described.

12.1 Security Information Class

This class is the source of security information for use within the distributed system.

Authentication Service

The primary operation supported by an authentication service is that of receiving some credentials, checking them, and issuing a certified identity in return.

Attribute Service

The purpose of an attribute service is to take in a certified identity or a set of certified privileges and some instructions, and return a more specific set of certified privileges and possibly some additional protection against misuse.

Interdomain Service

The purpose of an interdomain service is to map existing Privilege Attribute Certificates (PACs) held in one domain into one appropriate for use in another domain; including sealing the new PAC in a way that will be accepted in the other domain.

12.2 Security Control Services

This class consumes verified security information to control the activities of entities within the distributed system.

Secure Association Service

This service is designed to provide a secured association, including all the necessary communications security, between any two objects in the system, including associations over domain boundaries.

Authorisation Service

The purpose of an authorisation service is to deliver decisions on access control. It takes in the privilege attributes of the initiator, the control attributes of the target, a description of operation, and any appropriate context information. It returns: yes, no; or error.

12.3 Security Monitor Services

This class of services provides internal monitoring and control of security activities over the whole distributed system.

Security Audit Information Collection Service

The scope of security audit covers not only the collection of audit information, but also its processing and deciding which information to collect.

12.4 Related Services

All services are involved in aspects of security, but the data storage and sub-environment services are closely related.

13. FRAMEWORK ADMINISTRATION AND CONFIGURATION

13.1 Conceptual

There is no doubt that a CASE environment will have to be carefully administered, not least because its precise configuration may be constantly changing to meet the changing needs of the software development enterprise.

13.2 Operations

It is not believed that configuring and administering an environment should require any additional services than those already described. There should be a sufficient set of operations within each of the services to allow all appropriate manipulations to be made.

Clearly, some of those operations may only be available to those with system administration privileges. The point made is that those operations are provided as part of each of the services.

13.3 Examples

There may be a particular set of tools, or sub-environments defined for system administration or configuration. Again, this is just a customised use of the framework services.

Example use of framework services are the management of users, tools, services and hardware introduction and maintenance, controlling accounting, backups, and use of sub-environments and security policies.

BIBLIOGRAPHY

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

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

[3] ECMA. Support Environment for Open Distributed Processing (SE-ODP). Technical Report ECMA/TC32-TG2/89/26, ECMA TR/49, December 1989.

[4] Architecture Projects Management Limited. The ANSA Reference Manual, March 1989.

[5] H. Johansen(ed.). ANNEX 1 to Proposed New Question 6/X for the 1989 - 1992 Study Period. March 1988.

[6] ANSI. Interim Report of the ANSI/X3/SPARC Study Group on Data Base Management Systems. ACM SIGFIDET, 7(2):3--139, 1975.

[7] E.F. Codd. Data Models in Database Management. ACM SIGMOD, 11(2):112--114, February 1981.

[8] ISO OSI WG3. Information Resource Dictionary System (IRDS) Framework. Technical Report N776, ISO, Canada, 4 December 1988.

[9] DOD Ada Joint˜Program Office. Requirements and Design Criteria for the Common APSE Interface Set (CAIS). Technical report, DOD, 4 October 1986.

[10] G.I.E. Emeraude, Selenia, and Software Science Limited. Requirements and Design Criteria for Tool Support Interface (EURAC). Technical report, G.I.E. Emeraude, 17 July 1987.

[11] P. Hitchcock. A Database View of the PCTE and Aspect. In Software Engineering Environments, pages 37--49. Ellis Horwood, 1988.

[12] Alan Brown. Database Support for Software Engineering. Kogan Page, University of York, October 1989.

[13] M.L. Brodie. On the Development of Data Models, pages 19--47. Springer-Verlag, 1984.

[14] Data Semantics (DS-1), 7-11 January 1985.

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

[16] Joan Peckham and Fred Maryanski. Semantic Data Models. ACM Computing Surveys, 20(2):153--189, September 1988.

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

[18] Alan Brown. Database Support for Software Engineering. Number 1 85091 948 8. Kogan Page (Wiley in U.S.), October 1989.

[19] Scott E. Hudson and Roger King. Object-Oriented Database Support for Software Environments. ACM SIGMOD Record, 16(3):491--503, December 1987.

[20] CTE+ Project Team. PCTE+ C Functional Specifications, Issue 3, 28 October 1988.

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

[22] R. Munck, P.Oberndorf, E.Ploedereder, and R.Thall. An overview of DOD-STD-1838A (proposed), the Common APSE Interface Set, Revision A. ACM SIGPLAN Notices, 24(2), February 1989.

[23] J.A. Hall, P. Hitchcock, and R. Took. An overview of the ASPECT Architecture, pages 86--99. Peter Peregrinus Ltd., 1985.

[24] A.N. Earl and R.P. Whittington. Capturing the Semantics of an IPSE Database - Problems, Solutions and an Example. Data Processing, 27(9):33--43, November 1985.

[25] Digital Equipment Corporation. ANSI X3H4 Working Draft Information Resource Dictionary System ATIS. Technical Report ANSI X3H4/90-187, ANSI, 14 February 1990.

[26] Klaus R. Dittrich. The DAMOKLES Database System for Design Applications: Its Past, its Present, and its Future, pages 151--171. Ellis Horwood, 1989.

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

[28] J.Mylopoulos and H. Wong. Some Features of the Taxis Data Model. In Proceedings of the Sixth International Conference on Very Large Data Bases, pages 399--410. ACM, 1980.

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

[30] Michael L. Brodie and Dzenan Ridjanovic. Fundamental Concepts for Semantic Modelling of Objects, Oct 1984.

[31] Dan Fishman et al. Iris: An Object-Oriented Database Management System. ACM Transactions on Office Information Systems, pages 48--69, January 1987.

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

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

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

[35] P. Hall, J. Owlett, and S. Todd. Relations and Entities, pages 1--20. North Holland, 1976.

[36] A. Meier and R.A. Lorie. A Surrogate Concept for Engineering Databases. In VLDB 1983, 1983.

[37] B.R. Dillistone. VCMF - A Version and Configuration Modelling Formalism, pages 145--163. Peter Peregrinus, 1986.

[38] Peter Klahold, Gunter Schlageter, and Wolfgang Wilkes. A General Model for Version Management in Databases. In Proceedings of the Twelfth International Conference on Very Large Data Bases, pages 319--327, Kyoto, Japan, August 1986.

[39] E. Adams et al. Object Management in a CASE Environment. In Proceedings of International Conference on Software Engineering at Pittsburgh, May 1989.

[40] D.B. Leblang and R.P. Chase. Computer-Aided Software Engineering in a Distributed Workstation Environment. Sigplan Notices, pages 104--112, 19th May 1984.

[41] Dennis Heimbigner and Steven Krane. A Graph Transform Model for Configuration Management Environments. Sigsoft, 13(5):217--225, 1988.

[42] G.T. NGuyen and D. Rieu. Schema Evolution in Object-oriented Database Systems. volume 4, pages 43--67. North Holland, 1989.

[43] Geoffrey M. Clemm. The Workshop System - A Practical Knowledge-Based Software Environment. ACM Sigsoft, 13(5):55--64, 1988.

[44] P.W. Dell. Early Experience with an IPSE. Software Engineering Journal, pages 259--264, November 1986.

[45] P.Cronshaw. The Experimental Aircraft Programme Software Toolset. Software Engineering Journal, pages 236--247, November 1986.

[46] D. Ornstein. Developing a CASE Interchange Standard - EDIF/CASE. In Proceedings of CASE 88, 1988.

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

[48] Leon Osterweil. Software Processes are Software Too. In 9th International Conference on Software Engineering, pages 1--13, Monterey CA, March 1987.

[49] B. Warboys. The IPSE 2.5 Project: Process Modelling as the basis for a Support Environment. In Proceedings of Software Development Environments and Factories, Berlin, May 89.

[50] R. Weedon A.W. Brown and D.S. Robinson. Managing Software Development. In Proceedings of Software Engineering '86, pages 197--235. Peter Peregrinus, September 1986.

[51] Gail E. Kaiser. A Flexible Transaction Model for Software Engineering. 14 June 1989.

[52] F. Banuthon, W. Kim, and H. Korth. A Model of CAD Transactions. In 11th International VLDB, 1985.

[53] H. Garcia-Molina and K. Salem. Sagas. In Proceedings of ACM SIGMOD. ACM, 1987.

[54] Mark Dowson and Brian Nejmeh. Nested Transactions and Visibility Domains. In ACM Workshop on Software CAD Databases. ACM, 1989.

[55] OSF. OSF User Environment Component: Decision Rationale Document. 11 January 1989.

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

[57] ECMA. Security in Open Systems - Data Elements and Service Definitions. Technical Report Standard ECMA-138, ECMA, December 1989.

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