**Pyramid OsX** The software should run on a Pyramid computer running OsX. The particular system used was a Pyramid 98xe running version 4.0 of OsX.

Since a Berkeley UNIX system is the primary development platform for ISODE, this documentation is somewhat slanted toward that environment.

## 1.4 Organisation of the Release

A strict layering approach has been taken in the organisation of the release. The documentation mimics this relationship approximately: the first two volumes describe, in top-down fashion, the services available at each layer along with the databases used by those services; the third volume describes some applications built using these facilities; the fourth volume describes a facility for building applications based on a programming language, rather that network-based, model; and, the fifth volume describes a complete implementation of the OSI directory.

In *Volume One*, the "raw" facilities available to applications are described, namely four libraries:

- The *libacsap*(3n) library, which implements the OSI association control service (ACS).
- The *librosap*(3n) library, which implements different styles of the OSI remote operations service (ROS).
- The *librtsap*(3n) library, which implements the OSI reliable transfer service (RTS). and,
- The *libpsap*(3) library, which implements the OSI abstract syntax and transfer mechanisms.

In *Volume Two*, the services upon which the application facilities are built are described, namely three libraries:

- The *libpsap2*(3n) library, which implements the OSI presentation service.
- The *libssap*(3n) library, which implements the OSI session service. and,

- The *libtsap*(3n) library, which implements an OSI transport service access point.

In addition, there is a replacement for the *libpsap2*(3n) library called the *libpsap2-lpp*(3n) library. This implements the lightweight presentation protocol for TCP/IP-based internets as specified in RFC-1085.

In addition, *Volume Two* contains information on how to configure the ISODE for your network.

In *Volume Three*, some application programs written using this release are described, including:

- An implementation of the ISO FTAM which runs on Berkeley or AT&T UNIX. FTAM (file transfer, access and management), is the OSI file service. The implementation provided is fairly complete in the context of the particular file services it offers. It is a minimal implementation inasmuch as it offers only four core services: transfer of text files, transfer of binary files, directory listings, and file management.

- An implementation of an FTAM/FTP gateway, which runs on Berkeley UNIX.

- An implementation of the ISO VT which runs on Berkeley UNIX. VT (Virtual Terminal) is the OSI terminal service. The implementation consists of a basic class, TELNET profile implementation.

- An implementation of the "little services" often used for debugging and amusement.

- An implementation of a simple image database service.

In *Volume Four*, a "cooked" interface for applications using remote operations is described, which consists of three programs and a library:

- The *rosy*(1) compiler, which is a stub-generator for specifications of remote operations.

- The *posy*(1) compiler, which is a structure-generator for ASN.1 specifications.

- The *pepy*(1) compiler, which reads a specification for an application and produces a program fragment that builds or recognises the data structures (APDUs in OSI argot) which are communicated by that application.

- The new *pepsy*(1) compiler which will replace *pepy* and *posy* programs in a future release of the ISODE.

- The *librosy*(3n) library, which is a library for applications using this distributed applications paradigm.

In *Volume Five*, the QUIPU directory is described, which currently consists of several programs and a library:

- The *quipu*(8c) program, which is a directory system agent (DSA);

- The *dish*(1c) family of programs, which are a set of directory shell commands; and,

- The *libdsap*(3n) library, which is a library for applications using the directory.

## 1.5 A Note on This Implementation

Although the implementation described herein might form the basis for a production environment in the near future, this release is not represented as "production software".

However, throughout the development of the software, every effort has been made to employ good software practices which result in efficient code. In particular, the current implementation avoids excessive copying of bytes as data moves between layers. Some rough initial timings of echo and sink entities at the transport and session layers indicate data transfer rates quite competitive with a raw TCP socket (most differences were lost in the noise). The work involved to achieve this efficiency was not demanding.

Additional work was required so that programs utilising the *libpsap*(3) library could enjoy this level of performance. Although data transfer rates at the reliable transfer and remote operations layers are not as good as raw TCP, they are still quite impressive (on the average, the use of a ROS interface

(over presentation, session, and ultimately the TCP) is only 20% slower than a raw TCP interface).

## 1.6 Changes Since the Last Release

A brief summary of the major changes between v. 6.0 and v. 7.0 are now presented. These are the user-visible changes only; changes of a strictly internal nature are not discussed.

- A new program, *pepsy*, has been developed to replace both *pepy* and *posy*. It is described in *Volume Four*.
- The *dsabuild* program has been removed, in favor of some shell scripts.
- The "higher performance nameservice" has been discontinued in favor of a "user-friendly nameservice". As such, the syntax of the `str2aei` routine has changed. This routine will soon be deprecated, so get in the habit of using the new `str2aeinfo` routine discussed in the ISODE Manual.
- The `na_type` and `na_subnet` fields of the network address structure described in the ISODE Manual have been renamed. For compatibility, macros are provided. These macros will be removed after this release.
- The stub directory facility is now deprecated in favor of an OSI directory based approach. As a result, the *aetbuild* program has been removed.
- An interworking problem in the presentation layer has been fixed in a backwards compatible manner. This should not affect interoperability in this release. The backwards compatibility will be removed after this release.

As a rule, the upgrade procedure is a two-step process: first, attempt to compile your code, keeping in mind the changes summary relevant to the code; and, second, once the code successfully compiles, run the code through *lint*(1) with the supplied lint libraries.

Although every attempt has been made to avoid making changes which would affect previously coded applications, in some cases incompatible changes were required in order to achieve a better overall structure.

# Part II

# Overview

# Chapter 2

# Motivation and Concepts

Remote operations are a well understood technique for building distributed applications. Currently, OSI technology provides a powerful notation for specifying the external interactions of these systems. Remote operations in OSI are particularly attractive as they specify the homogeneous externally visible and verifiable characteristics needed for interconnection compatibility, while avoiding unnecessary constraints upon and changes to the heterogeneous internal design and the implementation of the information processing systems to be interconnected. Many feel that this capability is likely to be a key factor in the overall success of OSI standardization.

In [ECM85], a methodology for remote operations is presented which aims to package this technology concisely. Although the latter half of this document has been rendered obsolete by more recent work in the standards bodies (e.g., [ISO88a, ISO88b]) The aim of this volume, *The Applications Cookbook*, is to provide programming support for this methodology.

## 2.1 A Model for Distributed Applications

Remote operations are viewed as part of a methodology for building distributed applications.[1] In order to understand the role remote operations play in this programming model, one other notion must be explained first.

[1]This section presents a (hopefully) simplified version of the information presented in the first half of [ECM85]. Interested readers should consult this original work for further details.

### 2.1.1 Abstract Data Types

An *abstract data type* is a concept for describing a data structure which is accessed in a well-defined manner. This has several implications:

- Although the data structure may have a *concrete representation* on a given local system (e.g., a *C* `struct`), its corresponding abstract data type is defined in an implementation-independent fashion, termed its *abstract syntax*.

- A well-defined set of rules, defined by an *abstract transfer notation*, is used to serialize the abstract syntax so that a data structure corresponding to the abstract data type may be unambiguously transmitted on the network. There are actually two mappings here:

  - First, the data structure is mapped to the abstract syntax for the abstract data type. This is an implementation-dependent mapping.
  - Second, the abstract syntax is mapped to the concrete syntax. This is a serializing activity resulting in a stream of octets (although some concrete syntaxes produce bit-aligned, rather than octet-aligned, streams). Do not confuse the concrete representation described above (which deals with programming languages), with the concrete syntax (which deals with transmission properties).

- Access to an abstract data type is defined by a set of unitary actions termed *operations*, which define the complete behavior of an instance of the abstract data type.

- *Strong typing* results when operations are the only permitted means of accessing an abstract data type.

- An *object model* for programming follows from the use of abstract data types rather than concrete data structures. Since operations are used to ultimately manipulate the data structures, an important level of indirection has been introduced: data structures may be accessed without regard to their actual implementation.