-- --

IEEE P1003.0 Draft 13 – September 1991

This guide only cites government and de facto standards and specifications in discussion of gaps in available standards.

3.5 POSIX Open System Environment Profiles

The results of Open System specification projects are collected into an expanding set of ''Base Standards,'' addressing a growing subset of functional requirements.

Profile projects then select among these base standards to create a tailored, consistent set of standards addressing a more specific type (or instance) of system or set of application software. Profiles satisfy the requirements of application ''domains'' such as office or industrial automation, transaction processing, or real-time control systems.

This framework provides a way to characterize the functionality of profile activities. The current OSI profiles tend to focus strictly on the communications EEI. D
Other profiles might focus on a single component or span multiple interface types.

3.6 Application Platform Implementation Considerations

HLJ : All instances of ''System Integration Interface'' in this clause have been D
changed to ''System Internal Interface'' without further diff marks. D

Profile writers need to be aware that in an open system environment, the application platform can be decomposed into independently procurable components. While standards are interface specifications, and as such are independent of implementation, there are aspects of platform implementation or construction that may affect the specification of standards, and that profile writers may want to address.

For each case, the portion of the application platform that implements any particular independently procurable service is described as the service component. Figure 3-6 shows an application platform made up of several service components. If components interact, the specification of the interface between service components within the application platform may be standardized or nonstandard (including proprietary).

An intercomponent interface is labeled in Figure 3-6 as ''System Internal Interface'' because it may be used to assemble an application platform from multiple components. Figure 3-6 shows how a System Internal Interface is shown in the reference model.

A standards-based SII between the application platform service components addresses portability and interoperability of the application platform service components, not portability and interoperability of application software and systems.

Development of an SII would also require a consensus to emerge on the ''best'' design and implementation of system software/hardware. Very little consensus has developed on the partitioning of the platform into components and consequent

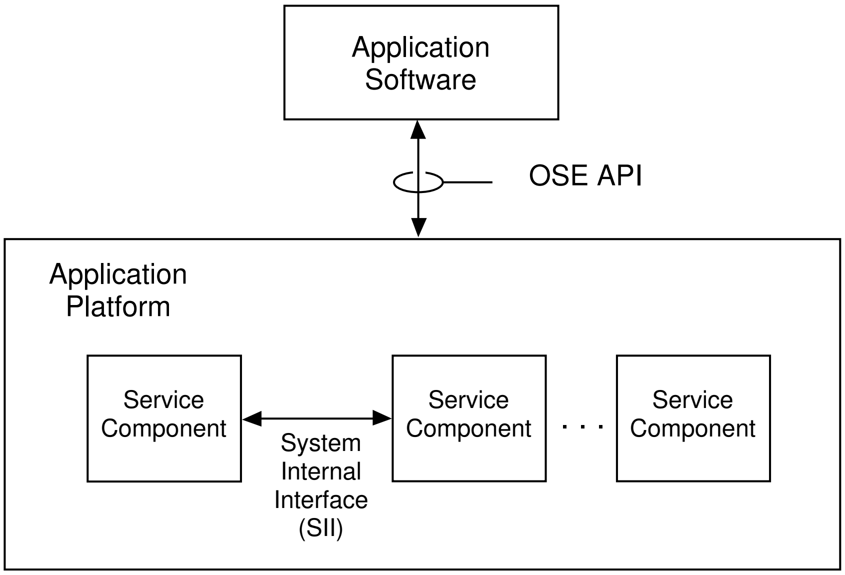

Figure 3-6 – Service Components and Interfaces

allocation of function to each. In fact, this aspect of system design has been in a constant and accelerating state of innovation for decades. One of the major objectives of the API is to provide a more stable interface that decouples application software from the constantly changing platform. This enables the migration of application software to platforms based on constantly upgraded technology. (See 3.1 ‘‘Accommodation of New Information System Technology’’.)

The relationship and services exchanged among the components may be quite complex and varied in different implementations. This complexity and variety would, of necessity, be reflected in an SII. It would not, however, be visible to the application software at the API, since one of the major objectives of the API is to hide this complexity. (See 3.1 ‘‘Implementation Transparency’’.)

Since SII specifications

— do not affect application portability and interoperability, and

— do not affect specification of the API and EEI, and

— are primarily driven by specific implementations of the application platform,

SII specification is beyond the scope of this guide. D

Specification of SII in this guide would represent an unnecessary constraint on the implementation of the application platform, and are unnecessary for the specification of the API and EEI.

There are a number of ways which the Application Platform can be divided into separate service components. The main decomposition methods are division, layering, and redirection. These methods are indistinguishable to the application software and external entities, in that they all interface to the application platform via the API and EEI, respectively. They assume a starting base application platform, which provides a subset of the required services.

3.6.1 Subdivision

In this commonly used method, the application platform is simply subdivided into a base and one or more service components. See Figure 3-7.

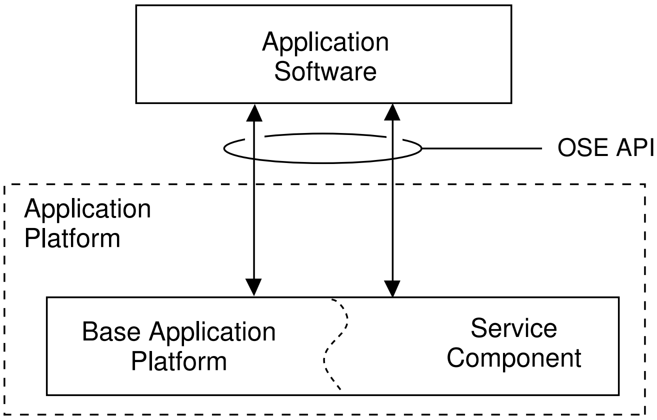

Figure 3-7 – Application Platform Implementation — Subdivision

One possible implementation of this is to link the appropriate service modules directly into the system kernel.

The internal interfaces used in this method are normally proprietary, and hence normally imply that both components will come from the same vendor.

In this case the Application Platform and the Application Platform Base are the same entity.

3.6.2 Layering

In layering, the service is interposed as a layer between the application software and the base application platform. See Figure 3-8.

This is the most common method of supplying a service component that is independent of the base. One possible implementation is to provide the service component as a set of library routines.

Whether the interface between the service layer and the base application platform conforms to any standards affects the portability of the service component. Note that specifying a standard API for this interface guarantees only that this

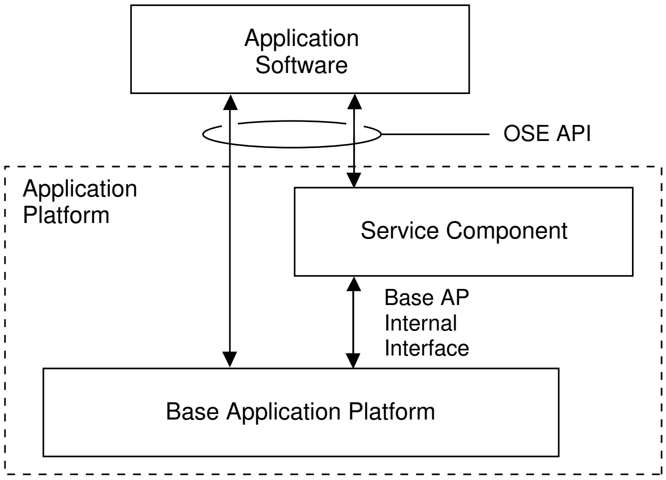

Figure 3-8 – Application Platform Decomposition II — Layering

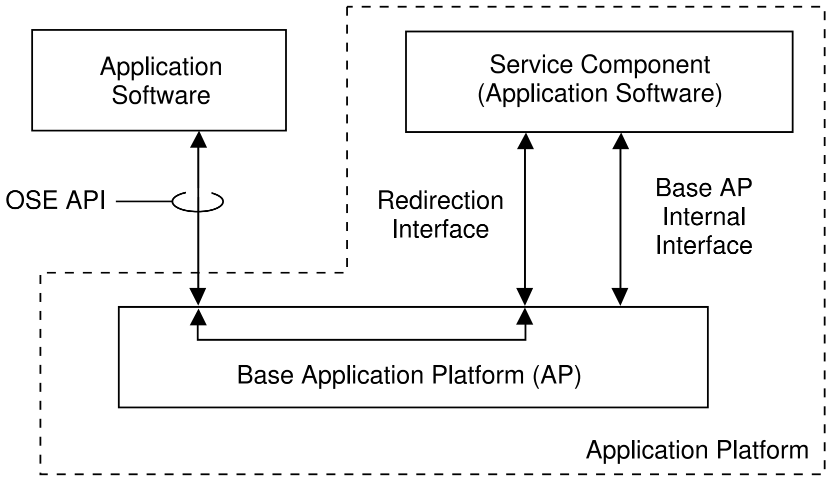

Figure 3-9 – Application Platform Decomposition III — Redirection

component will be portable at the source level.

3.6.3 Redirection

Redirection allows a service component to ask the base application platform to redirect all requests for that type of service to the service component. See Figure 3-9. Possible examples of such services are device drivers, network protocol handlers, and database engines.

In actual implementation, the service component may or may not be a separate process. Possible implementations are: dynamically loadable kernel modules, library routines layered over IPC, and lightweight kernel processes.

Note that there are three interfaces. The application software normally sees a complete, standard API to the base. The service component has two interfaces— one to effect the redirection, and one to provide base services to the service application software entity. Considerations for portability discussed under Layering also apply here.

Note also that no POSIX standardization activity currently exists for the redirection interface.

Section 4: POSIX Open System Environment Services

Responsibility: Fritz Schulz

This section describes the services required in support of the objectives identified in this guide. The services are grouped in major categories defined in Section 3, with more detailed breakdowns within each category as appropriate. These categories are:

Fundamental System Services

4.1 Language Services

4.2 System Services

Communications Services

4.3 Network Services

Information Services

4.4 Database Services

4.5 Data Interchange Services

Human-Computer Interaction Services

4.6 Windowing System Services D

4.7 Graphic Services

4.8 Character-Based User Interface Services

4.9 User Command Interface Services

Domain Services

4.10 Transaction Processing Services D

4.11 Software Development Environment Services

Criteria used to partition services are outlined in 3.2, and discussed at the beginning of each clause. The discussion for each of the service category subclauses follows the same outline, and is as follows:

4.*n*.1
4.*n*.2 Overview and Rationale, Scope

This text introduces the scope of this service category, and the criteria used to identify the services within it.