Tool Integration
and
Environment Architectures

Kurt C. Wallnau

Peter H. Feiler

May 1991

Tool Integration
and
Environment Architectures
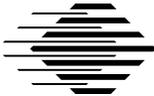
Kurt C. Wallnau
Peter H. Feiler

Software Environments Project

This report was prepared for the

SEI Joint Program Office
HQ ESC/AXS
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

(signature on file)

Thomas R. Miller, Lt Col, USAF
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense.

This document is available through Research Access, Inc., 800 Vinial Street, Pittsburgh, PA 15212.
Phone: 1-800-685-6510. FAX: (412) 321-2994. RAI also maintains a World Wide Web home page. The URL is http://www.ra-i.com

Copies of this document are available through the National Technical Information Service (NTIS). For information on ordering, please contact NTIS directly: National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161. Phone: (703) 487-4600.

This document is also available through the Defense Technical Information Center (DTIC). DTIC provides access to and transfer of scientific and technical information for DoD personnel, DoD contractors and potential contractors, and other U.S. Government agency personnel and their contractors. To obtain a copy, please contact DTIC directly: Defense Technical Information Center / 8725 John J. Kingman Road / Suite 0944 / Ft. Belvoir, VA 22060-6218. Phone: (703) 767-8222 or 1-800 225-3842.

Use of any trademarks in this report is not intended in any way to infringe on the rights of the trademark holder.

Table of Contents

List of Figures

List of Tables

Tool Integration and Environment Architectures

Abstract: The expanding CASE market is having substantial impact on software development environment technology in the area of environment support for tool integration. Sharpened awareness of CASE integration requirements, particularly in the context of the large number of fully developed CASE tools, has resulted in a technology shift away from monolithic integrated project support environments (IPSE) derived from the Stoneman model in favor of highly distributed environments based upon a federation of environment services. Federated environments promise environment framework support for the reuse of a large number of existing CASE tools and the development of highly interactive, tightly-integrated CASE environments. The evolution of environment framework technology to support CASE federation is predicated on an improved understanding of the techniques and issues of tool integration. One reflection of this improved understanding is recognition of the need to address integration mechanisms, tool semantic integration, and tool process integration as separate but related issues.

This paper describes the evolution of environment architectures to support federated CASE integration and outlines the implications of this evolution on the technical issues of CASE tool integration.

1. Introduction

The burgeoning computer aided software engineering (CASE) tool market has had substantial commercial impact on the development of software development environment (SDE) technology in the 1980's. To some extent, this development was envisioned as early as the late 1970's, and the need for software environment framework technology to support tool integration was discussed in the seminal Stoneman [1] report, which described the requirements and architecture of an Ada Program Support Environment (APSE). Since Stoneman, several SDE frameworks following the Stoneman model have been developed, most notably the Portable Common Tools Environment (PCTE) [8] and the Common Ada APSE Interface Set (CAIS) [2]. To date, however, no Stoneman-style framework has found substantial commercial success.

In the meantime, the CASE tool explosion has coincided with the rapid proliferation of bit-mapped workstation technology. This proliferation has resulted in a CASE tool platform market consisting of multiple hardware systems, operating systems, window systems, etc., with consequent impact on the way CASE vendors design their tools. Most substantially, vendors attempt to integrate with systems that provide the greatest degree of platform independence. In practice, this means adopting a "least common denominator" approach to platform service integration. As a result, vendors tend to shy away from complex frameworks such as PCTE in favor of more primitive "standards" such as UNIX and the X Window System.

While vendors have been arguably successful in achieving tool portability to a wide variety of platforms, the emphasis on portability and low-level integration has had an adverse impact on

the degree to which tools cooperate with each other. Rather than behaving like an integrated whole, CASE tools today more commonly behave like isolated "islands of automation" [9]. Worse than islands, the insularity of vendors brought on by market competition as well as the natural tendency to broaden CASE tool support to encompass a wider variety of development functions has resulted in tools which offer considerable overlapping and competing services. Further, each CASE tool may support or enforce different software development process models. As a result, existing CASE environments are difficult to manage, and they present conceptually muddy tangles of tool services.

As the CASE market stabilizes and fewer new CASE entries establish themselves in vertical or horizontal tool niches, CASE customers are increasingly demanding a resolution to the integration dilemma [37], and are searching for ways to achieve the ideals of tool integration envisioned by Stoneman in this era of widespread availability of sophisticated tools that have been developed on minimal platform services. This demand has led to increased exploration of strategies and techniques for CASE integration. Zarrella [32] discusses the results of these explorations from the perspective of the CASE tool and CASE vendors.

This report discusses the results of CASE integration explorations from the perspective of software development environment technology. The most notable results concern the evolution of software environment architectures away from the centralized IPSE characterized by Stoneman in favor of a decentralized services-based environment, described in this report as a federated CASE environment. This evolution is supported by a growing awareness of the complexities of tool integration issues. In particular, models of integration based upon the control, data and presentation integration classification schemes have been found to be wanting, especially where tool integration interacts with software development process models.

Structure of this Report

Section 2 discusses the trend in software environment technology away from the monolithic repository-based IPSE framework towards a more federated services-based framework. Section 3 describes an evolving understanding of different classes of integration issues that need to be addressed. This discussion can be seen as a starting point for identifying the kinds of services that tools and tool frameworks need to provide, and as a way for tool integrators to focus their efforts on important aspects of tool integration. Finally, Section 4 concludes with predictions for future CASE integration environment trends.

2. Evolution of Integration Framework Architectures

The fast developing CASE market has forced a re-evaluation of long-standing assumptions concerning architectures for integrated software development environments. The most significant development has been a trend away from monolithic, centralized environments in favor of more loosely coupled, tool-centered environments. Figure 2-1 depicts this trend in three distinct phases.
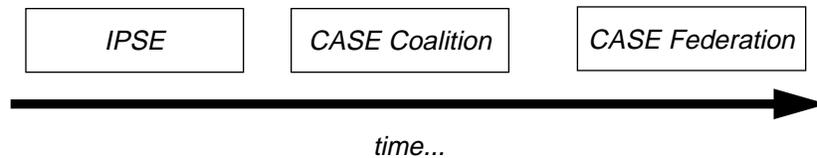
Figure 2-1 Framework Evolution to Federation

The first phase, highlighted in Figure 2-1a, is characterized by the centralized integrated project support environment (IPSE) and reflects the vision of integrated environment that formed in the early 1980's. The IPSE is depicted as evolving upward through object manage-
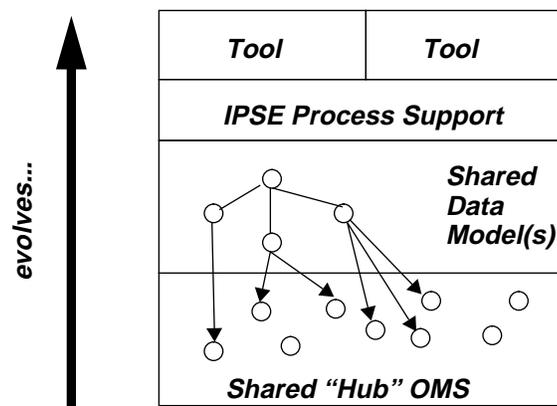
IPSEs are characterized by centralized ("hub") object management services and centralized support for software process management. IPSEs evolve vertically through OMS development, data model specification and, lastly, integration of tools.Software process support can be in the form of dynamic process control, such as found in ISTAR contracts [15], or static process control, such as found in the SLSCE data model [34].

Figure 2-1a Integrated Project Support Environments (IPSE)

ment and into tools. This reflects the concept that the OMS provides central services with which tools are later integrated. Although the term "IPSE" is often used synonymously with

terms such as project support environment, software development environment, software engineering environment, etc., for the purposes of this report IPSE will denote environment architectures characterized by integral, centralized software process support, and a centralized data repository that serves as the principle tool integration medium.

The second phase is highlighted in Figure 2-1b and is denoted as Coalition CASE environments. CASE Coalition environments reflect the reality of existing tool and integration technology and are characterized by tool-based environments created by CASE vendor alliances featuring ad hoc integration of CASE tools using whatever means are available. The coalition en-
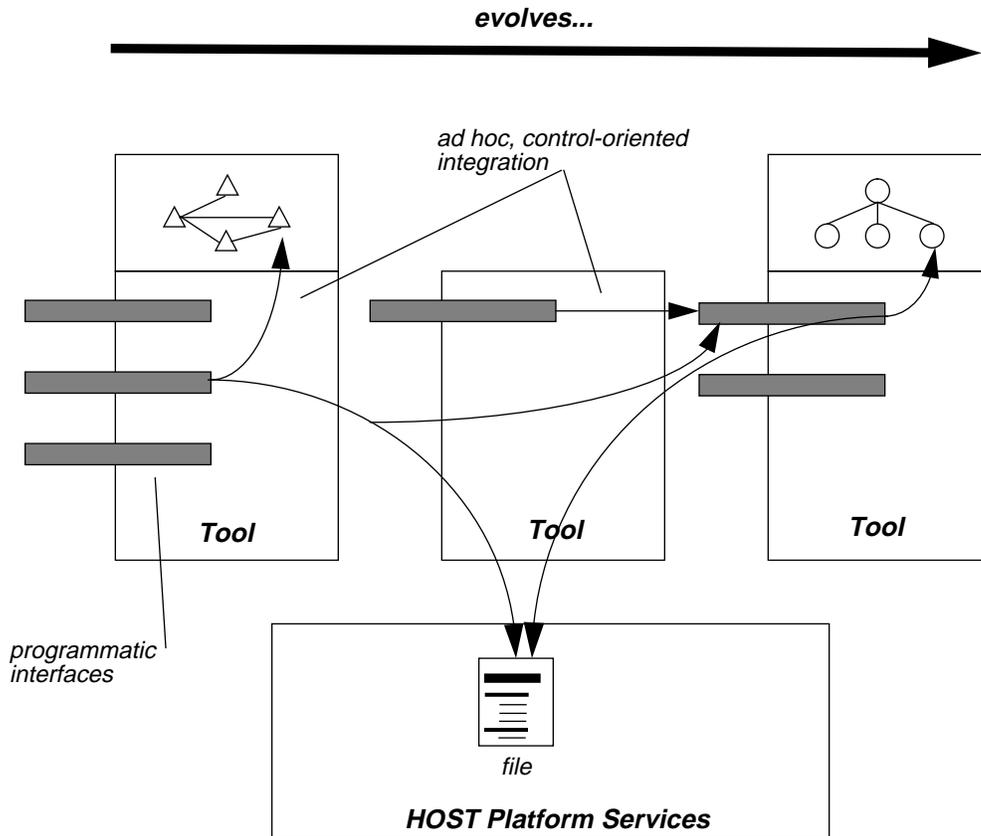
Coalition environments are characterized by specialized "point-to-point" integration of tool services offered by coalition participants. Coalition environments evolve laterally through point integration of new tools. Difficulties in achieving data interoperability among multiple private OMSs has resulted in a proliferation of integration techniques, with emphasis on control-oriented programmatic tool interfaces. Process models supported by CASE coalitions are implicit and embedded in the hard-wired integration of tool services.

Figure 2-1b Coalition CASE Environments

vironments are depicted in Figure 2-1b as evolving laterally. This reflects the point-to-point integration of coalition tool services with each other, rather than with underlying framework services.

The third phase is highlighted in Figure 2-1c and is denoted as Federated CASE environments. Federated CASE Environments are characterized by decentralized, tool and framework services-based environments. CASE federation environments represent a forward-look-

Vertical (Engineering) Service Domains

flexible, services-oriented integration

design services

testing services

Design Tool

Testing Tool

evolves...

other services

user interface services

data integration services

message server services

Horizontal (Framework) Service Domains
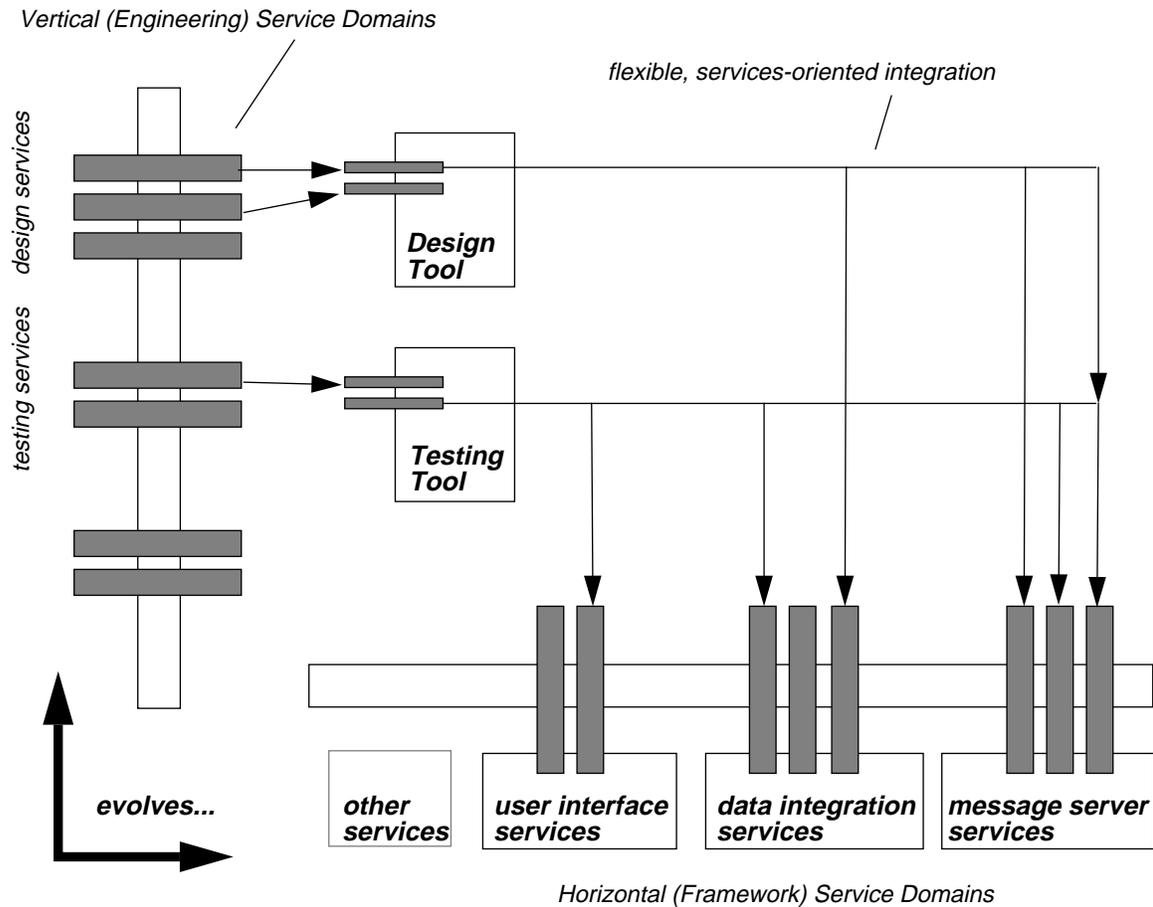
Federation environments are characterized by flexible, services-oriented integration. Federation environments evolve both horizontally through the introduction of new common services and vertically through the addition of new, specialized engineering services.Abstract service domains make the distinction between tool and intrinsic environment service moot. Service domains encapsulate semantically related sets of services and form a virtual machine basis for describing executable software processes.

Figure 2-1c Federated CASE Environment

ing vision of environment architectures, a vision which supports a merging and generalization of the best characteristics of IPSE and coalition environments. The federation environment is depicted as evolving both vertically and horizontally. This reflects environment evolution through the addition of new engineering services and new common framework services.

A more detailed view of the nature of the evolution of IPSE to Federated CASE Environments is provided in table 2-1.

	IPSE	CASE Coalition	CASE Federation
Environment Architecture	central OMS large-grained tools	multiple private OMS tool-function access	cooperating service domains
Integration Model	data-oriented integration	control-oriented integration	flexible control and data integration
Process Support	centralized, explicit large-grained process support	localized, implicit fine-grained process support	service domains as process enaction virtual machines

Table 2-1 Three Aspects of Environment Evolution

The following sections elaborate on each of the aspects shown in Table 2-1, describing where possible tie-ins to existing research and commercial technology and implications on tool integration.

2.1. First Generation IPSE

The environment taxonomy provided in Dart, et. al., [36] provides some insight into the historical and conceptual basis of the IPSE concept. The IPSE can be seen as an attempt to synthesize key aspects of language-centered, methods-based and toolkit environments into an organic whole. The key aspect of language-centered environments [13][14] incorporated by IPSE was the central repository, as this was seen as a means to construct a tightly integrated, semantically rich developer's toolkit. This, in concert with a desire for language and project genericity, i.e., a desire to "scale-up" the environments to include lifecycle and project-specific semantics, led to the hypothesis that a central, generalized environment database would be a key IPSE feature. Support for methods became associated with process and project management support, which also became an integral part of IPSE's. This architectural vision, in combination with the desire to "plug-in" a variety of tools led to IPSE framework efforts such as PCTE.

The IPSE concept is best characterized by environment architectures such as the Stoneman [1] "wheel" (a depiction of environment services as concentric rings). Although Figure 2-1 shows only one layer between the host and tool (the Object Manager), other layers of services can, and usually are, provided by the environment framework; object management is highlighted because it is the most important framework service provided in most IPSE architectures.

The generalized IPSE has three essential characteristics:

- Environment functions are viewed as a collection of large-grained tools and common underlying "hub" object management.

- Tool integration is data-oriented, and achieved primarily through shared object management.

- Process support is explicit, centralized, and part of the IPSE architecture.

One example of a fully-realized generalized IPSE is ISTAR [15]. More recent examples of IPSE frameworks are PACT [3] and its conceptual successor, EAST [16]. While each of these systems has a different emphasis, all share the above characteristics. The following sections elaborate these characteristics with emphasis on implications to CASE integration.

2.1.1. Hub Object Management and Large-Grained Tools

The functionality of an IPSE is largely defined in terms of the collections of tools present in the environment. For example, Stoneman described the minimal APSE (MAPSE) in terms of a "minimal set of tools": compiler, editor, linker/loader, debugger, command line interpreter and configuration manager. It was expected that projects would add to the minimal tool set to extend environment capabilities.

In practice, the most important class of common services offered by generalized IPSE includes those related to object management. The assumption is that tool interoperability can be best achieved through use of common data models and data management services [30]. Examples of common object management specifications include the object management services of CAIS and PCTE. Such object management systems (OMSs) are considered by many to be a necessary technology for data management in software environment applications, since neither relational database technology [4] nor raw file systems are well suited to the data management requirements of software engineering.

However, OMS technology is still in the process of maturing, and more widespread experimentation will be necessary to resolve important open issues [26]. One large-scale issue that typifies environment OMS technology immaturity is contention between entity-relationship-attribute (ERA) and object-oriented data models. At present there is little consensus on whether object-oriented databases (OODB) represent the next evolutionary progression of current-generation ERA-based OMS, whether OODB can be constructed as a layer on top of ERA OMS, or whether ERA OMS can be constructed on top of OODB. OODB technology is further complicated by a lack of consensus regarding the formal semantics of OODB data models, as well as differences in the perspectives among object-oriented programming language designers who are adding persistence to their languages versus database designers who are developing next-generation DBMS technology.

Besides this uncertainty over future-generation OMS technology, current-generation ERA OMS technology immaturity is found in two key areas: OMS performance and OMS object-granularity.

OMS Performance Issues

In practice, the performance of OMS implementations is disappointing both in terms of system throughput and in consumption of system resources. Stringent data security requirements and the complex semantics introduced by extending object management services into the realm of virtual operating systems services has produced OMS specifications that are difficult to implement and difficult to understand. OMS performance is a crucial concern for CASE vendors, since the interactive nature of most such tools is a vital part of tool functionality [40][41].

Recognition of the fact that for performance reasons CASE tools often require highly specialized data models and data management services has led some to the conclusion that a single OMS for all tools may not be practical. Recommendations such as the use of nested object management systems [26] reflect this recognition and attempt to model both the need for heterogeneous OMSs and the desire for a common data model at some level of granularity. Such proposals offer a partial solution to the OMS performance problem since OMS overhead can be isolated to tool start-up and shutdown times. However, from the vendor's perspective, nested OMSs offer little value-added to their tools at a potentially significant cost of re-targeting the tool to a different set of platform/framework services.

OMS Granularity Issues

While OMS performance may be an inhibitor to widespread integration of commercial CASE tools to IPSE frameworks, there are more fundamental issues concerning the suitability of a hub OMS as a primary means of achieving tool integration. One well known issue concerns object granularity. For performance reasons, most OMSs manage course-grained objects, usually corresponding in size to files managed by the file system. This may be a by-product of the historical view of the IPSE OMS as a replacement for conventional file systems. However, data management requirements for software engineering include management of objects spanning at least seven orders of magnitude in size; no existing OMS satisfies this requirement. As a result, OMSs tend to focus on lifecycle-process size artifacts, while leaving tools the responsibility for fine-grained object management, e.g., for nodes in a parse tree. Thus, even in a hub OMS-based IPSE, there is strong pressure for tools to provide private, tool-specific object management services. This contradicts the IPSE assumptions concerning tool interoperability and generates disincentives to the use of OMS services that introduce performance and system resource constraints.

Large-Grained Tools

The primary consequence of OMS granularity and performance issues is that IPSE tools tend to present monolithic architectures, i.e., offer a large-grained package of tool services that are only accessible from within the context of a running tool. For example, editors modify text files, and compilers translate text to binary, etc., but by and large, there are few opportunities for finer-grained interaction among such tools using first-generation IPSE OMS. There may well have been additional technological constraints on the development of less monolithic tools, e.g., the immaturity of low-level remote procedure execution standards. Regardless of the un-

derlying reasons, IPSE environment services tend to be viewed by environment users as being clumped into large, monolithic tools.

2.1.2. Data-Oriented Tool Integration Medium

The prominence of the IPSE OMS, when combined with the technological immaturity of remote execution concepts, resulted in a natural tendency to stress the data interoperability aspects of tool integration. However, the underlying IPSE assumption that a common underlying data model will enable tool data integration is not quite sufficient. Rather, in the IPSE model a common data model is a necessary, not sufficient, criteria for tool interoperability.

Even though tools may share the common *base* data model provided by the common data modeling services (e.g., an entity-relationship model), in practice tools require their own *domain-specific* data model, specified using base data model primitives. Tool integration requires integration of the domain-specific data model level as well as the base data model level. For example, the PCTE data model provides a fixed set of *link categories* in the base data model; however, this information is insufficient for version management tools to know in general which links need to be duplicated and which links must not be duplicated for new versions of arbitrary configurations of tool-defined objects [17].

Thus, even where CASE tools agree to share the same object management services, true interoperability can be attained only among tools that have entered into some bilateral understanding concerning the semantics of the shared data. This has an adverse impact on the ability to migrate new tools into an environment that have not been developed a priori for a specific OMS schema.

2.1.3. Centralized, Explicit Large-Grained Process Support

Software process support has long been a part of IPSE framework implementations. For example, ISTAR provides software support via a contract mechanism. PACT, on the other hand, embeds software process concepts in its environment schema. In both cases, the support for software process becomes an explicit part of the environment's concept of operation. Also, in both cases, the supported processes tend to be large-grained, usually corresponding to some kind of process artifact, e.g., a change request. Because of the large-grained nature of tools, finer-grained process support was not characteristic of IPSEs.

These ISTAR and PACT efforts point to two distinct views of IPSE process management: support through enaction mechanisms such as the ISTAR contract model, and support through data modeling. Both views have provided valuable insight into the nature of framework support for software process, and a number of issues have surfaced that are pertinent to tool integration.

2.1.3.1. IPSE Process Enaction Mechanisms

One concern with IPSE process enaction is the degree to which enaction mechanisms will adequately support the dynamism of processes, and to what degree the mechanisms support

process tailorability. For example, the ISTAR implementation raises questions about dynamic changes to the underlying process model [19]. There is also ongoing research into the separation of process planning from process enaction [45], as well as research into questions about what constitutes a process task, what level such tasks should be automated, and how such tasks should be specified [46].

These concerns reflect the fact that systematic process enaction is still very much a research problem. Commercial CASE vendors are not be inclined to expend resources integrating with unstable, ill-understood enaction mechanisms developed in research laboratories and universities.

2.1.3.2. IPSE Process Data Modeling

Environment schemas that model software processes, either at the lifecycle level [34] [4] or at the development model level [17] (see Section 3.3 for a discussion of lifecycle and development processes) are better understood and thus less controversial than enaction mechanisms. However, one complication arises because process data models span at least two levels of data models: those at a lifecycle level, involving lifecycle level artifacts such as requirements documents, and those at the tool level, i.e., individual steps within a lifecycle model. Tool-level models involve finer-grained objects such as individual requirements within an enclosing requirements document. The global level might be used to describe product structures, while the tool level might be used to describe tracability relations.

Both lifecycle and tool-level data model integration are necessary to support data modeling of software processes. Unfortunately, as already mentioned, current-generation OMS do not support fine-grained object models. Lacking this, CASE vendors are liable to see only limited value-added by integration with IPSE process data models, especially since IPSE lifecycle models are far removed from interactive tool services that might benefit from finer-grained data model integration.

2.1.4. IPSE Summary

IPSE frameworks characterized by hub OMS services for data and process integration did not address many of the requirements for CASE integration. OMS performance problems derived from complex requirements and overly broad virtual operating system specifications are built-in disincentives to CASE integration. The hub OMS concept suffers from more intrinsic conceptual problems, including the mismatch of framework and tool object granularity requirements and the realization that even common OMS data models are insufficient to achieve data model level interoperability. Finally, IPSE process support mechanisms addressed only large-grained processes. Since CASE vendors were more immediately concerned with fine-grained tool process issues it appeared that IPSE process support did not provide much value-added to tool functionality.

2.2. CASE Coalition Environment

Given the failings of the IPSE model described above, it is not surprising that the CASE market evolved in the absence of a widely accepted high-level integration standard (i.e., "framework"). As markets stabilized within various lifecycle niches, demonstrable integration with other CASE offerings became a competitive selling point for tool vendors. While integration standards are debated and environment research continues, in the near-term vendor coalitions provide a pragmatic bridge between customer demands and the ideal integrated CASE solutions. Some examples of vendor coalitions include the Interactive Development Environment (IDE) Software Through Pictures (STP), Sabre-C and FrameMaker coalition, and the Verdix Ada Development System (VADS) APSE, which includes the VADS compiler and choices of CADRE Teamwork or STP and FrameMaker or Interleaf.

The key characteristic of coalition environments is the degree to which the coalition member tools continue to behave in an egocentric manner. That is, tools developed in isolation have evolved idiosyncratic concepts that have impact on software development processes, e.g., multi-user support and version control. In coalition environments, the tools do not surrender such concepts in favor of common integration paradigms, but instead continue to project their own process models. Running counter to coalition egomania is the gradual "opening-up" of tool interfaces, particularly interfaces to tool functions, that are needed to support coalitions. Coalition integrations are therefore introducing interesting questions regarding the interplay of vendor concepts of proprietary services, open interfaces, and integration-support for software processes.

CASE coalition architectures are essentially the inverse of IPSE architectures. Where IPSEs provide a central database and large-grained tools, coalition tools provide their own databases, and tool services are becoming separately accessible; where IPSE support data-oriented integration via OMS services, coalition tools define their own integration models and services, frequently relying on remote execution and other forms of control-oriented integration; and where IPSEs provide explicit support for software process, especially large-grained processes such as lifecycle process, coalition tools support finer-grained processes in an implicit, localized fashion. The following sections discuss the ramifications of these inversions.

2.2.1. Multiple, Private OMS and Tool Function-Level Access

CASE tools providing complex services, such as those services offered by design and programming tools, are likely to rely on tool-specific object management services. As a result, coalition environments introduce multiple repositories, each potentially (and usually) providing proprietary, unique, tool-specific data models and data management concepts. This situation introduces issues of repository access and consistency maintenance, duplicated data management services, and management of relationships spanning different repositories in order to support lifecycle process management.

Repository Access and Consistency Maintenance

CASE OMS are frequently highly specialized for tool-specific data processing requirements. For example, the relational model may be well suited to front-end design tools but not suitable for programming language tools, where finer-grained in-memory linked structures may be more appropriate. The multiple OMS found in coalition environments is complicated by tool repositories that are opaque to the outside world. Opaqueness can refer to any or all of:

- repository data model

- repository data management and structure

- repository access functions

Repository opaqueness exacerbates the impedance mismatch among the various tool-specific data models found in coalition environments. That is, multiple OMSs supporting different data models will introduce difficult integration problems, even if these OMSs provided "open" interfaces, e.g., a uniform model for object identification or object versioning.

Some vendors attempt to provide repository openness by publishing database schemas and generalized schema-independent database access routines. At present, though, not all tools are so open-minded about their repositories. One reason for this reluctance is that tool data management services frequently are insufficiently generalized to support a broad class of clients. In addition to concerns over publishing proprietary, highly specialized schemas, the published-schema approach introduces problems of guaranteeing the consistency of repository data. That is, the responsibility for maintaining semantic constraints is distributed across all tools accessing data through a published schema.

An alternative way vendors can provide an external view of their repository is to provide programmatic service interfaces, such as proposed in [50][51]. This kind of external repository view mechanism parallels data abstraction and information hiding and solves the data consistency problem by allowing the provider of the repository to enforce semantic constraints. The programmatic repository interface approach heralds the provision of more generalized service interfaces by tools. However, it is not always obvious at what level of detail such data should be made available[52]. In [51], for example, access is provided to individual attributes of a semantically evaluated Ada parse tree, while in TeleSoft's proposed interface, the Ada Static Semantic Interface Specification (ASSIS) [52] access is provided only to objects at a much coarser granularity (e.g., compilation units). In these cases, the design decision over access granularity will be determined by process issues, i.e., the intended use to which the published services will be put.

Overlapping Data Management Services

One consequence of distributed heterogeneous OMSs is that tool users and organizations must rely on tool vendors to provide certain critical data management services. One especially important set of services concerns version and configuration management (CM). CM services and models are themselves highly evolved [38][39], and tools frequently do not support the same models of CM, provide the same CM services, or provide similar services in a similar

way. This class of overlapping tool services is particularly troublesome because of the impact CM has on defining and supporting software processes.

Difficulties in Achieving Lifecycle Integration

Another consequence of distributed heterogeneous OMSs concerns the integration of tool artifacts into a larger-scale data model than provided by individual tools, i.e., lifecycle process models. A lifecycle model incorporates tool artifacts spanning several tools and lifecycle stages. Lifecycle integration includes not just large-grained artifacts such as entire design documents, but finer-grained integration as well, for example, tracability of individual requirements through to code artifacts. While some tools have provided support for certain lifecycle models through specialized data export services, e.g., DOD-STD-2167A documentation generation, more generalized and flexible mechanisms are wanting. In particular, the one-way information flow from tool to repository implied by export filters is very inflexible and turns a repository into a depository. Further, fine-grained object integration continues to be difficult.

2.2.2. Control-Oriented Tool Integration Medium

The lack of standard integration services has encouraged vendors to be more active, and creative, in the definition of integration models. While in some cases such models will be very traditional, such as published database schemas, in other cases vendors are providing innovative and complex integration services. The multiplicity of integration services and models is on the one hand providing a good basis for vendor experimentation with tool integration in a more autonomous tool context than in IPSE, but on the other hand it makes tool integration esoteric and expensive, and usually makes it impossible for third-party tool integration solutions.

While vendor coalitions are apt to make use of all possible avenues for tool integration, including data sharing, the most significant difference between coalition and IPSE integration is the degree to which coalitions exploit remote execution and other forms of interprocess communication. That is, the difficulties in achieving data integration is resulting in a technology push on control integration techniques.The significant result of this technology push is to entice CASE vendors to provide interfaces for the remote execution of tool services. Thus, while IPSE data integration encourages the development of black-box tools, coalition control integration encourages the development of tools with open interfaces.

The Frame Technology Live Links [28] mechanism is an example of an innovative integration service which addresses data integration among multiple OMS. Live Links combines data interchange standards with remote procedure execution to achieve a simulation of a homogeneous repository (i.e., a simulation that the data in a FrameMaker document resides in one place). In order to support Live Links, integrating tools are encouraged to provide programmatic access to tool functions, e.g., for displaying and editing "linked" objects in FrameMaker documents. Similarly, FrameMaker provides programmatic access to several of its services to facilitate the integration of FrameMaker into tool coalitions.

However, while the demands of coalition integration are resulting in publication of tool schemas and services (so-called "open systems"), not all vendors interpret the concept of open-

ness in the same way. One distinction is the degree to which vendors consider their tools the necessary centerpiece of an integration solution (the "me centered" approach) as opposed to just one piece of a composable integration solution (the "me, too" approach). While composable "me, too" solutions argue for loose coupling between tools, "me centered" tools tend to require a significant buy-in to their services on the part of other tools.

For example, me-centered tools tend to provide services that require them to act as master in master slave interactions with other tools, while me-too tools support a broader class of peer-level intertool communication. The basic distinction between master/slave and peer-level communication is the degree to which tools make their services available for remote execution by other tools. Tools that only behave as masters tend to view themselves as being the environment driver application, or as a central integrating agent (i.e., playing the role of IPSE OMS). On the other hand, tools that provide remote execution facilities offer a more flexible interface for development process integration and intertool cooperation.

2.2.3. Localized, Implicit Fine-Grained Process Support

As already discussed, multiple private OMSs makes it difficult to achieve the large-grained, explicitly modeled lifecycle process integration envisioned by IPSE OMS. However, the discussion above about master/slave and peer-level tool interactions is derived from an emerging trend for tools to provide programmatic interfaces to tool services, and this in turn points to an evolving support for finer-grained, more interactive software processes. That is, traditionally interactive tools such as FrameMaker are now providing interfaces for the non-interactive execution of interactive services. As a result, CASE architectures are becoming more amenable to a finer-grained, discrete-function level integration. One visible consequence of this is that vendors are able to demonstrate coalition integrations that are dramatically interactive. A more subtle consequence is that this fine-grained tool-function level integration leads to a number of issues concerning support for fine-grained software processes.

The term "fine-grained process" refers (vaguely) to those aspects of the software development process which are related to small-scale tasks as opposed to discrete transitions in lifecycle phases. For example, creating a new version of an object can be considered a fine-grained process activity. Such small-grained tasks can correspond to the execution of a small number of services in a set of integrated tools. For integrators, knowing which services to integrate often reduces to the question of knowing which fine-grained processes the integrated toolset is intended to support. Viewed from the other direction, the set of function-level integrated services in a coalition environment is a specification of a fine-grained process model. In this sense, the specification is implicit since it is embedded in the tool services; one consequence of this is that the specification becomes fixed, i.e., non-extensible, non-tailorable.

Besides being implicit and fine-grained, process support in a coalition environment is localized in each of the tools. The significance of this lies in the degree to which tools model their own concept of various fine-grained processes. That is, as user requests for tool services cascade to other tools, it is possible that the user's focus will move among several tools automatically. This is the case with FrameMaker's Live Links, where an editing operation may result in the

launching of an interactive CASE tool. Such a seamless migration of user focus through various tools has the potential for creating a conceptually cohesive environment, but it can also result in cognitive dissonance if each tool provides a completely different set of process-related services. For example, if each tool provides different models of multi-user, versioning and configuration management services the result of rapid, automatic context switches among tools can be disorienting.

2.2.4. Summary of Coalition Environments

Coalitions among CASE vendors provide a pragmatic bridge between the near-term demand for integrated CASE solutions and the long-term ideal of generalized plug-compatible integrated CASE environments. One result of the press for coalitions is that vendors are increasingly sensitive to the need to provide access to their tool services, and this is resulting in a trend for more "open" tool architectures. One important class of tool services being provided by vendors allows for remote execution of tool services. Remote execution supports more highly interactive integrations, and supports fine-grained tool support for the software development process.

On the negative side, the software processes supported are ad hoc, are implicit in the integrated tools, and are not tailorable. The multiplicity of integration services provided on a per-vendor basis leads to integrations that are esoteric, expensive and possibly beyond the reach of third party integrators. Also, distributed heterogeneous OMS makes data modeling of software processes, such as lifecycle processes, difficult to achieve without the active support of the CASE tools.

2.3. CASE Federation Environment

While CASE coalitions represent the most mature technology on the integration product scale, the search for more generalized support for CASE integration is being pursued by computer manufacturers, software environment researchers, CASE vendors and entrepreneurs. The most prominent approaches fall into two categories:

- repository standards, such as IBM's AD/Cycle [11], DEC's CDD/Plus [48], and the National Institute for Standards and Technology (NIST) Information Resources Dictionary Standard (IRDS) [7] and IRDS extensions [5].

- service broadcast models, such as in FIELD[18] and Hewlett-Packard's Soft-Bench Broadcast Message Server (BMS) [10].

While neither approach is sufficient for achieving CASE federation, both approaches are evolving towards each other and in unison could provide the foundations for CASE federation.

Federation environments are not yet a reality, and much of what follows is speculative. However, there are visible trends that support the conclusion that IPSE software development environment concepts are merging with CASE coalition technology. The federated environment offers to generalize the coalition view of partial access to tool functions and heterogeneous repositories into a generalized model of distributed service domains. Federated environments

will also offer a more balanced view of data and control integration, allowing tool integrators flexibility in making trade-off decisions concerning the cost/benefits of choosing one form over another. Finally, federation framework mechanisms offer the opportunity of making the implicit, hard-coded, fine-grained process support of coalition environments more explicit, abstract and tailorable. These ideas are discussed in the following sections.

2.3.1. Cooperating Service Domains

In both the repository and message broadcast models services are accessible only after they have been defined via some abstract interface (method interfaces or protocol specifications). This requirement encourages the definition of families of service interfaces whose semantics are explicitly described — in short, the process of abstraction.

Both the message broadcast model and repository model encourage the abstraction of tool services into definitions of abstract service domains. In [10] and [18] protocols are defined in terms of discrete areas of tool functionality, such as editors and debuggers. The CASE Interface Standard (CIS) specification [6] defines service domains in terms of object types and operations ("methods") associated with these types. Both models represent abstractions of tool functions into tool services, i.e., the separation of the implementation of a function from its abstract specification.

Designing a protocol for a class (or domain) of services requires maintaining a delicate balance between the desire to exploit the unique capabilities of a particular tool in a class against the desire to be able to reuse the protocol for other tools within the same class. A similar trade-off is made in the ECMA environment framework reference model [27], where service domains are partitioned into common integration services rather than tool services.

The process of services abstraction in both approaches has several beneficial implications:

- modularization of tool services into more cohesive, complete service packages

- counters tool egocentrism by emphasizing what services are provided, not who provides them

- separation of service interface from the service itself supports alternative service implementations, perhaps coexisting, within an environment

- facilitates recognition of commonly provided services and reorganization of these services into common framework services

The cumulative result of these implications is the evolution towards environments providing cooperating sets of services. That is, common (or "horizontal") services such as CM and documentation can be shared by many tools, while specialized (or "vertical") services can be provided by domain-specific tools, such as design tools. The assiduous adherence to a service design philosophy that balances service functionality against generality can ensure that these vertical and horizontal service sets are independent of particular tools, and instead represent abstract domains of services.

2.3.2. Flexible Control and Data Integration

CASE repository standards emphasize data integration while broadcast models emphasize control integration. Both models are evolving towards each other, and the resulting frameworks promise tool integrators flexibility in choosing among integration mechanisms.

CASE repository standards represent an architectural approach to tool integration very similar to that of the Stoneman (and hence IPSE) models. One variation is that these models make an attempt to separate an enterprise's data model from the repository data storage model, thus providing more systematic support for distributed OMS than provided in earlier IPSE. Another, more significant variation from past realizations of Stoneman is that the data models are object-oriented. The significance of object-orientation is the association of functional behavior with OMS object types. This, in turn, provides a data model basis for tool integration at the level of remotely-executable tool services, where tool services are accessed through references to repository objects. Note that object-orientation does not itself support remote service execution, nor does it imply a particular implementation. Instead, the data model provides uniform interfaces that can support a variety of remote execution implementations.

The service broadcast models represent a fundamentally different approach to tool federation. Rather than envisioning a centrally shared data modeling service, broadcast models envision a centrally shared message broadcast network and tool communication protocols. The conceptual model supported by this form of tool federation is one where tools are thought of as nodes in a distributed network of CASE service nodes, with tool services accessed by requests broadcast in a well-defined protocol. The message broadcast model is particularly striking in the way it encourages the view of services as distributed functions. While object-oriented repository models associate tool services with data objects that appear to be centrally located, the message broadcast model separates the service from the data and treats all tool services as requests to a network of distributed services. This explicit view of distribution makes the broadcast model more flexible in support of providing redundant sources of services as well as in the selective broadcast of certain service requests to a limited service audience.

It is noteworthy that while efforts derivative of ATIS, such as CATIS and CIS, are expanding their specifications to include support for a wider variety of control integration strategies [32], Hewlett-Packard is also engaging in an effort to re-implement the SoftBench BMS, which is the heart of the SoftBench concept, using the PCTE framework in order to support a wider variety of data integration strategies. In this latter example, even though the PCTE repository will still suffer from the earlier-discussed IPSE OMS shortcomings, the combination of message broadcast and data sharing mechanisms will provide a tool integrator with a basis for making trade-off decisions regarding whether to employ data integration, message-based coupling, or some combination of the two.

Such trade-off flexibility is useful when considering the integration of tools within tool families versus the integration of tools in different families. "Tool family" is meant to denote a class of tools that share significant functionality. For example, coding/debugging/testing tools can be considered members of a tool family, as can requirements/structured design tools. In the case

of coding/debugging/testing tools, it is quite likely that the tools will benefit from tight data sharing of a common intermediate language as well as control integration to allow a rapid, seamless migration of user focus among the tools (i.e., an enhanced code/test/debug cycle). A similarly "tight" integration may be cost-justifiable among the design tool family. However, it may not be necessary, or cost-effective, to require such extensive integration among tools that belong to different families.

Another way a tool integrator can use such flexibility is to decide whether a service should be performed by moving control to where the data resides (i.e., control integration) or moving the data to where the control resides (i.e., data integration). Such a decision can be based upon performance constraints as well as on software process issues. For example, data integration may require too much data transfer while extensive control integration may result in significant operating system process initiation overhead.

2.3.3. Abstract Process Definition and Enaction

Abstract service domains can represent virtual interfaces to process abstractions in the same way that the implicit integration of tool services in coalition environments represents a kind of process encoding. For example, within a horizontal service domain such as configuration management, services can be provided that support various abstract models [38]. Such services can be viewed as a virtual machine basis for encoding those software process elements that depend upon configuration management and multi-user coordination. Similar examples can be drawn from various vertical service domains, e.g., the design domain can be partitioned into various design methods, with services associated with each method, and design processes derived from or specified using design method primitive services.

Also, while it is true that process enaction mechanisms are not well understood and there are many research approaches still being explored, it is also true that the control integration mechanisms and fine-grained service access inherent in federated environments provide a fertile substrate for developing process enaction mechanisms, particularly with respect to fine-grained process enaction. The broadcast model is noteworthy in this regard because it provides an obvious mechanism for migrating process-flow aspects of control integration into the framework. That is, it is easy to envision process enaction mechanisms that use the broadcast model.

2.3.4. Summary of CASE Federation Environments

While federation environments are not yet a commercial reality, several research and development efforts, as well as activity in the formal and de facto standards/consortium arena, promise advances in software development environment technology that will build upon, and generalize, the successes of vendor coalition environments as well as IPSE efforts.

The significant advances include a changing perception of environment functionality from a tool-based view to a service-based view. This changed perception is a result of the need to describe the interfaces among tools that are coordinating on a much finer grain of services

than is the case even in coalition environments. These interface descriptions raise the level of abstraction in tool interface descriptions and provide a mechanism for identifying discrete horizontal and vertical service domains. Such well-defined domains offer promise of greater support for plug-compatible tool integration and the possibility of third party tool integrations. Further, the well-defined domains offer an abstract encoding of environment services that can be used as a virtual machine for specifying enactable process models.

Finally, generalizations of successful data and control integration techniques found in IPSE and coalition environments support flexible tool integration. The combination of integration flexibility and abstract service specifications provides a context in which technology can evolve in different domains (tool and environment) without adversely affecting the validity of the underlying integration framework.

3. Types of Tool Integration

The trend towards CASE federation is both pushing and being pushed by an evolving under-standing of the technical issues of tool integration. As already discussed, new integration mod-els are being provided by CASE vendors that address the inherently decentralized character of CASE-based environments. These models build upon existing integration mechanisms (e.g., remote procedure call, object management systems, exchange standards such as Post-Script, etc.) and are generating a wide variety of approaches to achieving CASE integration. Each of these new approaches, as well as traditional approaches, has consequences on the quality of integration when perceived from many different perspectives. Examples of such per-spectives include the environment builder, who is looking for generality of integration tech-niques, the CASE vendor, who is looking for platform availability of integration mechanisms, and the CASE user, who is looking for coherent support for software development processes.

In order to discuss tool integration in a coherent way, the complexities of the problem space need to be partitioned in some way. Several problem decompositions have been proposed. For example, Nejmeh describes the characteristics of tools that are related to integrability [20]. More commonly discussion has focused on three classes of integration: presentation, data, and control integration. By and large the discussions of these classes of integration focused on integration mechanisms. Further, such mechanisms were considered extrinsic to the tools themselves. However, as CASE vendors and environment integrators gained more experi-ence with CASE integration, it became apparent that the focus on presentation, control and data integration needed to be refined.

Wasserman broadened the focus to include two additional classes of integration: process and platform integration [21]. In Wasserman's paper, process integration addresses the integration of software development tools with process management tools, while platform integration ad-dresses tool integration with a virtual computing service layer for network and operating sys-tem transparency. Ian Thomas (Hewlett-Packard) and Brian Nejmeh (INSTEP) in a soon to be published paper took this view further by recognizing that integration occurs between pairs of entities, and that the relationships between these entities has a number of properties that can be used to characterize the integration. Further, in Thomas and Nejmah's model, 1) process integration generalizes Wasserman's definition of process to address conceptual integration with an organization's software development process model, and 2) framework integration generalizes Wasserman's platform integration to include use of any common substrate servic-es. Finally, Martin Cagen (Interactive Development Environments) proposed a further decom-position of process integration into three distinct kinds: enaction models (which addresses Wasserman's concern), lifecycle models and development models.

Our refinement of this model is twofold. First, we argue that framework and process integration are orthogonal to control, data and presentation integration (and to each other). Process inte-gration defines the design constraints on tool integration solutions imposed by an organization (or user community), while framework integration defines implementation constraints imposed

---

by integration mechanisms. Put another way, process integration defines "what" gets integrated while framework integration defines "how" integration works.

Second, we argue that intertool integration is still characterizable via the now classical control/ data/presentation partition. However, the control/data/presentation relationships between tool entities does not describe integration mechanisms (mechanisms are described in tool/framework relations), but rather describes the conceptual models of integration occurring between integrated tools. That is, while the framework provides integration mechanisms, the tools define the semantics of the integration. The significance of this distinction lies in the degree to which intertool semantic integration is migrating into framework services. This paper argues that the trend towards environment federation is reflected, and in some sense gauged, by the migration of integration concepts provided by tools in support of coalition integration into generalized framework services.

The unified view of integration depicted in Figure 3-1shows the three classes of entities involved in integration: framework, process and tools. The services illustrated as belonging to the framework are taken from the ECMA software development environment reference model [27] and illustrates the relationship of control, data and presentation mechanisms with intertool control, data and presentation relationships.

The following discussion will focus in turn on each of the kinds of integration illustrated in Figure 3-1. Each kind of integration will be described and, where possible, the relationships among the kinds of integration to commercial or experimental integration techniques or frameworks will be discussed.

3.1. Framework Integration

Wasserman describes platform integration in terms of a "set of services that provide network and operating systems transparency" to CASE tools. This focus is too narrow when taken in the context of current integration practice. Rather, tools may be integrated with sets of services that provide something less than network and operating system transparency; conversely, some CASE integration services go far beyond mere host systems transparency. The current trend is to consider integration in the context of something more general than tool portability services; this "something" is frequently referred to as a "framework."

3.1.1. Variants of the Framework Concept

The term "framework" is ambiguous, having different meanings depending upon the context of a particular discussion. A framework can refer to sets of standards, a conceptual model or schema, the generic or common part of a family of application specific environments, a system of programmatically accessible services, or generalizations and extensions to native operating systems. This ambiguity stems from the confusion of several distinct concepts: host services,
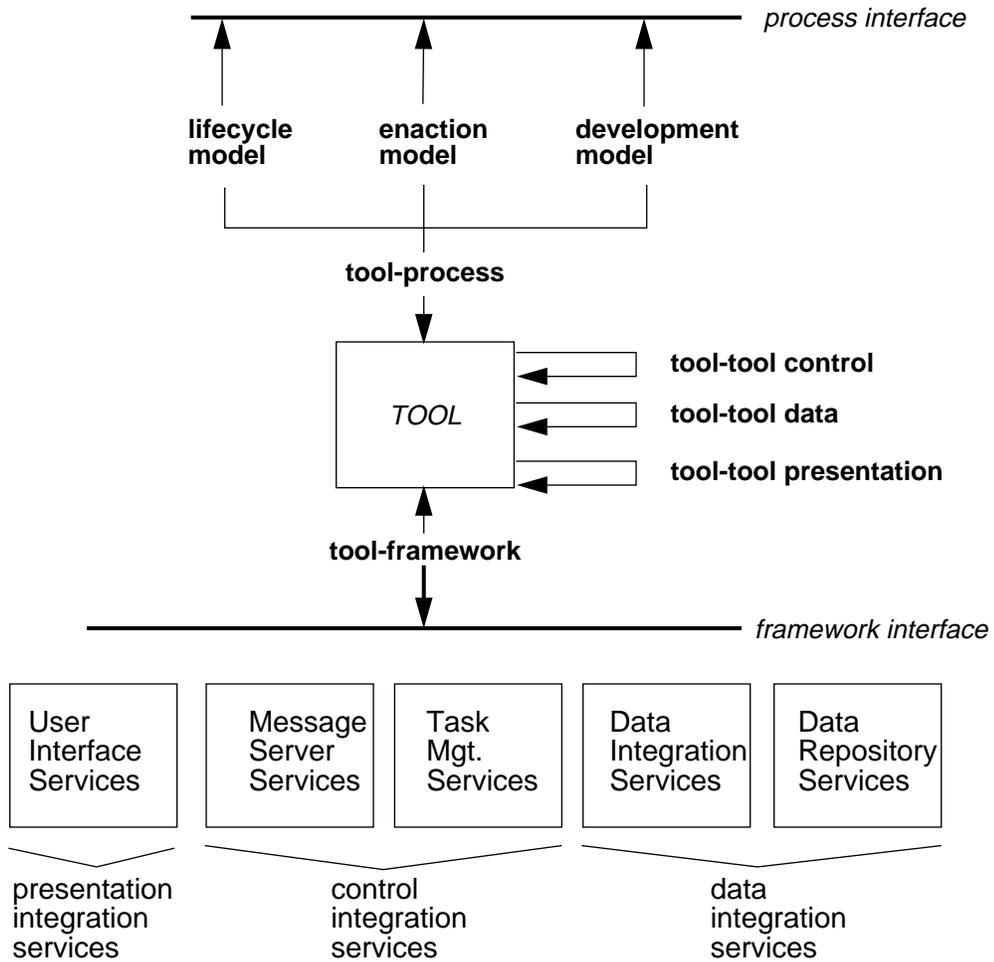
Figure 3-1 Types of Integration

platform services, framework services, and environment services. The following relationship describes the way these concepts are related:

$$host \subseteq platform \subseteq framework \subseteq environment$$

For example, a host represents the native operating system services available on a machine, e.g., UNIX; the platform represents portability/transparency services, e.g., POSIX; the framework represents services that provide a high-level conceptual model for tool integration, e.g., CIS, and the environment represents a framework that is populated with cooperating CASE tools.

Of course, all of these levels (host, platform, etc.) can be coalesced or re-interpreted; a typical example is the denotation of UNIX as a software environment. Alternatively, one can view a system as spanning several levels; for example PCTE can be thought of as combining platform and framework services. What is significant, however, is that it is possible to evaluate in-

tegration as occurring at any of these levels, and in each case a different set of integration issues may surface. For example, environment-level integration may require integration with a process model or with other tools, while framework integration may require integration of a tool with a common data model.

It is not surprising to find that framework integration has been the focus of continuous research and development since the time of Stoneman because it is so fundamental to tool integration. In practice, frameworks tend to focus on some combination of these objectives:

1. to provide specialized services for application development

2. to support construction of portable integrated tools

3. to support the incorporation and integration of tools developed for different frameworks

4. to provide a maximum degree of platform independence

Of course frameworks need not focus on one objective to the exclusion of the others; nevertheless, there is tension among these objectives. For example, environments that are intended to maximize 1 and 2 will probably ignore 4; examples include InterLISP [14] and FSD [22]. Environments that are intended to focus primarily on 1 will produce frameworks such as Sun's NSE[23] and Apollo's DSEE [24]. Environments that focus on 2 and 4 result in frameworks such as CAIS and PCTE, while a focus on 4 alone produces a minimal framework based upon standards such as POSIX and X. Support for 3 results in frameworks such as SoftBench and specifications such as ATIS/CIS.

3.1.2. Tool and Framework Integration Services: Migration to Framework

In an effort to characterize and relate framework products an ECMA environment framework reference model task has been undertaken to identify and define environment framework services [27]. The results of this effort are directly applicable to understanding issues of tool-framework integration. One particularly important ECMA reference model technique is the three schema approach of defining framework services in terms of conceptual, internal and external levels. The conceptual level describes what a service is, the internal level describes how a service is implemented, and the external level defines how the service is made available.

Several points of interest need to be made about the 3-schema definition of services. First, the separation of interface from implementation seems to imply that tools sharing the same framework interfaces but different implementations can nonetheless be integrated. In practice, i.e., in residual IPSE and current-generation coalition environments, this is not the case (although federated environments may provide a cleaner separation of service interface from implementation), and instead the conceptual model, interface and implementation details are more tightly coupled than strictly necessary. As a result, tools frequently need to integrate with all three schema levels of a framework service.

Second, the conceptual level provides a description of the semantics of a service which is necessary but not sufficient for tool integration. In practice, tools need to augment the conceptual level with tool or domain-specific semantics. To illustrate this and the three schema approach, consider data integration in the PCTE framework. In PCTE, two tools share data by sharing the same schema definition set (SDS). Internally, the SDS is a PCTE framework mechanism. Externally, the SDS is accessed via programmatic interfaces and shell commands. Conceptually, the SDS is a *view* into an entity-relationship-attribute (ERA) database. Tools integrate with all three levels of PCTE SDS's; in addition, tools need to agree on an interpretation, or meaning, of the data model described by the SDS view. This fourth level of integration is beyond the scope of the framework services, and instead represents a bilateral semantic interface between the integrated tools.

Finally, in the above PCTE example, all three service levels (conceptual, interface and implementation) are provided by the framework (PCTE). In some cases, however, it is possible that tools themselves provide integration services constructed upon other, lower-level, framework services. In these cases, the tool can provide the conceptual level for a service while the framework provides the mechanisms and interfaces. Where such tool-provided conceptual-level services become widely accepted, it is possible to find migration of these concepts into the framework. For example, analogues to Frame Technology's Live Links services and Interleaf's Active Document services [31] are beginning to be found in commercial integration frameworks [35].

3.2. Process Integration

Although environment support for software process has been a significant component of past environment efforts [15] [33], the recognition that process plays a role in the integration of tools is a more recent realization [21] [32]. Given the variety of tools, tool services, and integration mechanisms available for integration, an increasingly important question that needs to be asked by the tool integrator is how the tools (or tool services) are intended to be used. Some integration strategies that make sense under one software development process may not be reasonable under alternative process models. Deciding which tool services to integrate, and how to integrate them, is dependent upon the organization's software processes that are being automated or supported. Conversely, tools themselves may support or impose process models that may have an impact on the organization's processes.

Process models at two levels of granularity are discussed, below. Lifecycle models address the course-grained processes derived from a macro-level view of the product development cycle over the lifetime of a product. Development models address the finer-grained processes involved in the day-to-day activities and interactions of managers, developers, testers, etc., in the development of a product. Note that the distinction between lifecycle and development processes is not always clear since the development process is ultimately concerned with the production of lifecycle artifacts described in the lifecycle model. Finally, several recent research projects are exploring innovative techniques for specifying and controlling software processes

within an environment. These techniques are combined into a generic software process enaction category, which in the future may become significant in tool integration.

3.2.1. Lifecycle Process

Lifecycle process refers to the artifacts, the relationships among artifacts, the processes that generate artifacts, and the sequencing of these processes during the entire lifecycle of a product. There are two aspects of lifecycle process integration of interest to tool integrators: integration with the model as a whole, and integration with tools that manage other, related lifecycle artifacts. In the former case, the integration issues typically concern the interaction of a tool with a repository data model that reflects the lifecycle model; in the latter case the issues typically concern tracability relations. Both are discussed, below.

3.2.1.1. Repository Structure Integration

One obvious way of modeling the lifecycle process is in terms of the products produced at various stages of the process. The Software Life Cycle Support Environment (SLCSE) [34] provides an example of this concept. In SLCSE, a data model representing the DOD-STD-2167A lifecycle model provides the structure for tools to integrate their by-products. Figure 3-2 depicts a trivial lifecycle model that will be used to illustrate some of the points raised below.
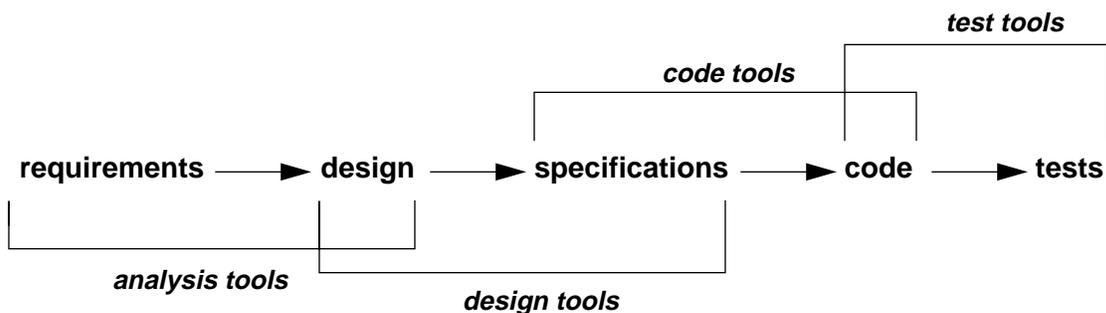

Figure 3-2 Lifecycle Context Integration

At its simplest, this form of integration may require that tools generate artifacts to conform to various internal format and external structuring requirements. For example, design tools may be required to produce documentation formats that satisfy a standard such as DOD-STD-2167A or generate objects that may not be a part of the logical processing requirements of a tool but fit the structural requirements of the lifecycle model. In addition to these product-oriented lifecycle artifacts, there may be additional process-oriented artifacts, such as change request objects, that may require some integration efforts.

More complicated issues arise where lifecycle model details address various states and transitions between states of artifacts in the model. For example, individual artifacts may pass through several intermediate states ranging from "preliminary" to "final" versions with different

semantics associated with each state, e.g., visibility and access control. Lifecycle artifacts may also exist within a family of related artifacts, for example, platform variants or variants reflecting maturity states, such as "experimental," "development" and "released," or sequential versions within states, such as "release 1.5." In these more complex scenarios, issues of access control, object sharing, data compression, consistency, etc., all need to be addressed by the tool integrator. Further, because CM systems are frequently used to support this class of lifecycle model issues, tool integration with the lifecycle model may require close coordination of the integrated tool with a set of CM services.

3.2.1.2. Fine-Grained Lifecycle Integration: Lifecycle Consistency

While the kinds of lifecycle relationships described above can be thought of as course grained since they deal with artifacts at the granularity of, say, a requirements document, another class of relationships concerned with much finer grained objects is also present in lifecycle integration. The need for fine-grained lifecycle relations arises from two requirements: tracability and dependency management. Tracability reflects the desire to preserve design rationale in the form of relations (e.g., these modules are derived from this data flow bubble which satisfies these requirements). Dependency management reflects the need not only to manage the manufacturing process of a system but also to maintain consistency among tools which share logical artifacts.

Fine-grained relationship integration becomes complex in CASE environments where tools make use of private, tool-specific repositories because different data models and management schemes make object-level coordination difficult. In addition to issues of forming and maintaining relationships between objects in heterogeneous repositories, dependency management also introduces issues of cooperation of tool services for update notifications and re-derivation of logically shared objects.

To illustrate the last point, consider the following example. CASE tools with specialized repositories are likely to maintain private, tool-specific forms of logically shared artifacts. Thus, a design tool will have an internal representation of the system interface specifications that will eventually be imported into a coding tool's specialized repository. These tools share a logical object type, but have redundant copies of the object represented in separate repositories. If changes are made to the system interface specifications within a coding tool, these changes should be reflected backwards to the design tool, which may in turn need to interact with the analysis tool. It is conceivable that changes would need to be propagated in the other direction as well. Each such propagation may require execution of tool services to import, process, and possibly notify other tools of the need for further propagation.

Continuing with the example, it is also clear that lifecycle integration may call for varying degrees of consistency maintenance. That is, at various development stages it may be important for various parts of the lifecycle model to be inconsistent. For example, exploratory changes to a design within the context of a programming tool may be reasonable. In such circumstances, automatic change propagation may not be desired, and instead a less intrusive notification

mechanism such as electronic mail to concerned parties may be more appropriate. Mechanisms that support inconsistency are discussed briefly in [42].

This example illustrates how closely related lifecycle and development processes can be, how tool architectures can impact the integration strategies required to satisfy a process requirement, and how tightly coupled these strategies are to the availability of integration mechanisms and tool services to support these strategies.

3.2.2. Development Process

Development process refers to the model employed by an organization to support the day-to-day activities of environment users, particularly those activities that imply coordination among multiple environment users and distinction of various user roles. While the concept of development process is closely related to ongoing research in the area of support for collaborative work, all multi-person software development efforts enforce, through management stricture and/or automated mechanisms, some notion of control on the development process.

The difficult issues that are raised by development model integration concern the alignment of tool-defined development models with those of the customer organization. Consider again the lifecycle model described in Figure 3-2. Imagine a development model that stipulates different user roles for designer and developer. Further, in this hypothetical development model, only the designer is permitted to modify system specifications. In this scenario, it might be important that the coding tool honor this access control constraint. While this would be straightforward in an environment where tools share a common repository (and hence common access control), in an environment in which each CASE tool imposes its own development models and is ignorant of external models such an integration would be far more difficult.

One widely used vehicle for expressing a model of, or constraints on, the software development process is a CM system. Modern CM systems can be used not only to enforce role-based access control, as in the previous example, but also to support developers in a variety of ways. Organizations are increasingly adopting the use of CM systems in their environments, and, similarly, CASE tools with significant data management requirements are likewise implementing embedded CM services. Thus, an important aspect of development model integration is the coordination and cooperation of CM services that may be distributed across various tools and within the environment framework, and that may support different underlying CM models [38].

3.2.3. Enaction Mechanisms

Enaction mechanisms refer to services that automate the execution of process models. Although forms of process enaction have been found in earlier environments, e.g., ISTAR and DSEE, on the whole, this form of integration is less well explored, at least in terms of the ultimate form such enaction mechanisms might take. While research efforts such as Arcadia [12] are exploring executable process specifications, other efforts are aimed at a more fundamen-

tal understanding of what units of task activity are and at what level they are reasonably automated [43].

Despite this uncertainty, the trend towards federated CASE environments is generating tool architectures and tool integration framework technology which can support a much finer-grained process enaction model than is the case with environments populated by large, monolithic, predominately interactive CASE tools. As part of the trend to CASE federation process, enaction mechanisms will gradually migrate from the tools to the framework.

3.3. Intertool Integration: Control, Data and Presentation

Various aspects of control and data integration were discussed earlier in this report when describing the IPSE, Coalition and Federation environment concepts. This section reprises the significant implications on control and data integration posed by the evolution of decentralized, federated environments. Then a more detailed discussion of presentation integration is provided because presentation integration clearly demonstrates the migration of tool-originated conceptual models of presentation semantics into framework mechanisms.

3.3.1. Control Integration

This form of integration refers to the ability of one tool to execute functions provided by another tool or, more generally, to support remote execution of tool services. The recent rise in the prominence of control integration mechanisms is partly a result of the seemingly intractable problems with generalized data integration, and the favorable balance of perceived value-added to two or more tools loosely coupled via some control integration strategy versus the cost of implementing this form of integration. Whereas data integration aims at moving data to where control resides, control integration can be use to invert this and instead move control to where the data resides. This is the way Live Links achieves data integration by use of control integration mechanisms.

Although control integration may seem like a recent technology, environment and tool builders have in the past provided control integration mechanisms in the form of data-driven and event-driven triggers. Data-driven triggers cause actions as a result of a change in the state of a database (OMS or repository). The CAIS-A attribute monitor mechanism is an example of support for data driven triggers. Event-driven triggers cause actions as a result of some activity that occurs in an environment. The NSE notification mechanism is a limited example of event-driven triggers; more recent and general event triggers can be found in FIELD and SoftBench.

What is significant about the recent uses of control integration techniques is the affiliation of control integration with finer-grained access to tool functions than was found in IPSE environments. This trend is being amplified by the market pressures for CASE coalition integration, and it is becoming increasingly common to find CASE architectures that support the programmatic, non-interactive execution of otherwise interactive tools. The combination of control integration standards such as RPC and open CASE architectures (with respect to programmatic

interfaces) is a major driving force in the development of environments based upon federation of tool services.

Programmatic access to separately selectable tool services is having a positive impact on support of fine-grained processes in CASE environments. The combination of fine-grained service support and advances in event- and data-driven trigger support sets the stage for the development of more reactive and proactive environments. Greater reactiveness and proactiveness, when coupled with a trend away from egocentric tool architectures, sets the stage for a greater degree of seamlessness between tool boundaries, as control is passed implicitly among tools as a result of user activities rather than as a result of explicit user-level tool invocations. Framework support for this kind of seamlessness is fast becoming a commercial reality [35], though commercial tools may lag in their use of such services.

3.3.2. Data Integration

Data integration has long been the centerpiece in the research and development of tool integration frameworks. Since the time of Stoneman, environment architectures featuring central repositories have been commonplace. This basic architecture is still found in general purpose environments [3] as well as language-centered environments [22] [25]. The underlying assumption of these architectures is that tool integration is best achieved through common data representation and models [30][34][4]; the central repository provides a mechanism for tools to access the artifacts produced by other tools, and for expressing the relationships between these artifacts.

For reasons that were discussed in Section 2 of this report, the central repository research ideal has yet to become commonplace in industry. Instead, data integration is commonly achieved, if at all, through an eclectic assortment of mechanisms, including:

- format mechanisms: data interchange through use of non-proprietary external formats, such as PostScript, and proprietary external formats, such as Frame Technology's Maker Interchange Format (MIF)

- storage mechanisms: data interchange and data sharing through external files, clipboards, cut-buffers, CASE-specific data repository services, common databases, and an assortment of intertool conventions

- carrier mechanisms: data interchange through interprocess communication, such as UNIX pipes and sockets, and Sun Microsystems' Remote Procedure Call (RPC) and External Data Representation (XDR)

Tool vendors in coalition environments have achieved rather dramatic successes in tool integration by making use of any and all of these mechanisms. These successes, coupled with the rise in prominence of control integration techniques, have caused a reappraisal of old assumptions concerning the primacy of data integration.

Yet some set of common repository services will be required by an environment in order to support generic activity controlling processes, such as configuration management and project management [26]. General recognition of this requirement continues to drive efforts to define a repository concept to support data integration in CASE environments. The requirement for common repository services, when coupled with trends toward greater tool autonomy and the desire to support tool migration into frameworks as an alternative to a priori tool/framework integration, has resulted in interesting advances in repository and data integration concepts.

One concept is the separation of the repository data model from the underlying data management services. In this way the repository, or services making use of the repository, can access objects through the data model, while other environment tools can bypass the data model and use the data management services directly. This concept supports tool migration by allowing different levels of data integration. For example, data management services can be provided by the native file system; in this model, native host tools can still achieve some degree of integration with tools using a more sophisticated data model than provided by the file system. Separation of data model and data management can be found in the ECMA framework reference model, the CIS specification and in Sun Microsystems' NSE.

Another related concept is the separation of relationship management services from data management services, or the provision of alternative relationship (or "link") management services. This concept is derived from the need to provide tracability and configuration management relations among tools that manage their own data, as is frequently the situation with highly complex CASE tools. The ECMA software environment framework reference model reflects this recent concept by noting that relationship services need not be an intrinsic part of an underlying data model. A proposal to the CIS committee [29] and, less ethereally, the NSE link services are examples of attempts to provide generalized relationship services in the context of distributed, heterogenous repositories. SofTool's CCC [44] is an example of CASE tool (as opposed to framework) provision of relationship management services. Frame Technology's *Live Links* and FrameMaker's *Active Documents* [31] are examples of specialized link services combined with control integration to provide a degree of transparency to relationships among heterogeneous repositories.

At this time it is not clear where the ultimate responsibility for some of these services will lie, i.e., with the framework or with tools. However, one clear trend is to have certain data integration services, such as data management, reside in the framework while other services are provided by tools or are specialized from framework services to the semantics of the data models implemented in tool repositories, e.g., link services. Ultimately, though, the data integration services — whether residing in the framework or in the tools — will be more tightly coupled with control integration services in order to support a broader class of process-oriented data models.

3.3.3. Presentation Integration

The relative maturity of both hardware and software graphical display technology has resulted in a gradual separation of user interface concepts from underlying mechanisms. Thus, the common "look and feel" among tools that is the goal of presentation integration is largely independent of particular presentation mechanisms.[1] Thus, it is possible to write X applications that present the same look and feel as Macintosh applications.

One indication of this separation of concepts from mechanisms is illustrated by the NIST user interface reference model [53], shown in Figure 3-3. Thus, a common look and feel is achieved among applications that share the same realizations of the presentation ("look") and dialogue ("feel") layers. In an ideal world, different tools could implement different layers of the NIST stack and still achieve a common look and feel. In practice, however, the separation of conceptual models from implementation mechanisms has not been followed by a separation of interfaces from mechanisms. Thus, window systems, toolkits, etc., tend not to be easily interchangeable. As a consequence, practical achievement of common look and feel requires agreement among tools to use the same presentation mechanisms, such as OSF's Motif toolkit.
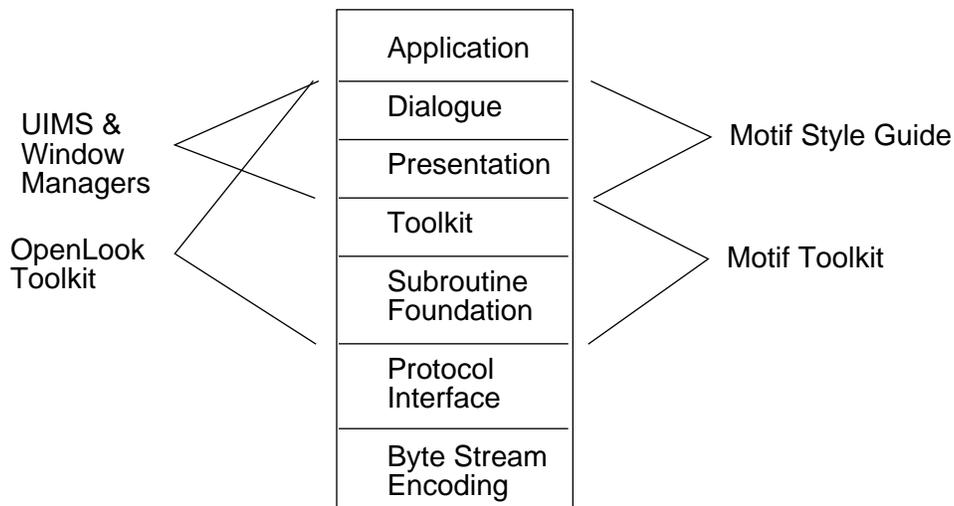

Figure 3-3 NIST User Interface Reference Model

Another approach to presentation integration that offers some promise of removing the integration dependencies on low-level implementation mechanisms (or rather, replacing dependencies on low-level mechanisms with dependencies on window system-independent mechanisms) is through the use of user interface management systems (UIMS). An example of

---

1. For the purpose of this report, presentation integration will refer to integration issues dealing with the bitmapped, window-oriented display technologies, since these are the most important technologies related to CASE integration.

such a UIMS is Serpent [47]. Serpent illustrates an attempt to separate dialogue from presentation, and separate presentation from specific presentation interfaces.

Despite their conceptual appeal, UIMS's probably will not have substantial impact on tool integration in the near future. The reasons for this appear to mirror many of the reasons why IPSE frameworks have not been widely adopted (see IPSE discussion in Section 2.1). Foremost among these reasons are a lack of UIMS maturity, which is reflected in several ways: UIMS's are not yet widely available in customer environments, where they will be needed to offer run-time services, and application developers have concerns over UIMS-imposed constraints on user interface functionality as well as application performance.

As in data and control integration, some degree of bilateral semantic integration is necessary to achieve integration objectives. For example, it might be desirable for tools that share logical repository object types to display these types in a similar way, or for tools to share an agreed-upon convention for the meaning of certain sequences of mouse button or key sequences. This latter concept is referred to by the IEEE P1201 user interface standardization committee as "drivability," and is based on the analogy of automobiles, i.e., what factors are involved that make it possible for people to drive many different models of cars.

One interesting aspect of presentation integration is the degree to which some of these semantic integration issues have become widely accepted by tool vendors, and the degree to which such de facto standardization is reflected in presentation mechanisms. For example, it is generally recognized that the OpenLook toolkit encodes more presentation and dialogue policy than the Motif toolkit, which relies more on tool adherence to a well-specified style guide [49]. While this does not imply that OpenLook is more mature than Motif, it does point out that continued experience with tool integration will result in a gradual migration of implicit semantic or policy issues of integration into common framework mechanisms.

4. Conclusions

IPSEs represent an early vision of software environment architectures to support large-scale software development efforts. The key aspects of IPSE, centralized object management and process management, make sense architecturally. However, the continued technological immaturity of OMSs as well as the conceptual immaturity of process modeling and enaction mechanisms has resulted in the development of IPSE frameworks that are not widely accepted by tool vendors. Instead, the CASE tool industry has developed in isolation of IPSE research and development efforts. This isolation has resulted in the development of sophisticated, highly functional tools that are, unfortunately, unintegrated with each other. The resulting environments, populated with unintegrated CASE tools, raise troublesome questions with respect to environment scalability and evolvability.

Market pressure for integrated CASE environments is resulting in the development of new models of tool integration. The current generation of integrated CASE environments are being created by CASE vendor coalitions. These coalitions are demonstrating integrated CASE solutions that satisfy the near-term demand for tool integration, and point the way toward more generalized solutions. In particular, coalition CASE technologies provide a basis for creating more interactive, cohesive and coherent environments based upon well-defined domains of environment services, some of which are provided by tools. However, tool architectures have not kept pace with the development of integration mechanisms that support coalition environments. In particular, CASE tools exhibit a large degree of egocentrism that makes generalized, tailorable integrations difficult to achieve.

The development of CASE architectures that are more "open" with respect to tool functions is resulting in less egocentric tools. This, in combination with advances in abstract framework mechanisms that support exploitation of open CASE interfaces, such as object-oriented repositories and message broadcast mechanisms, points to a trend for a more generalized federation of tool services than found in coalition environments. Federated environments are still more of a vision than a reality, however. At present, tool vendors still do not know which integration models and mechanisms work best, or how to combine these mechanisms into integration solutions that will be widely acceptable among customer organizations.

A key aspect in gaining vendor and customer acceptance is found in an evolving understanding of the technical issues of tool integration. Of interest to vendors are the technical aspects of framework integration, i.e., the availability of generalized, flexible integration mechanisms. Of interest to customers is process integration, i.e., the degree to which integrated tools support chosen software processes. Process and framework integration define the what and how, respectively, for achieving intertool control, data and presentation integration.

References

[1] *Requirements for the Ada Programming Support Environment: Stoneman*, Department of Defense, February 1980.

[2] *DOD-STD-1838A, Common Ada Programming Support Environment (APSE) Interface Set (CAIS), Revision A*, Department of Defense.

[3] Thomas, Ian, "Tool Integration in the PACT Environment," in *Proceedings 11th International IEEE Conference on Software Engineering*, pages 13-22, May 1989.

[4] Penedo, M. H., Stuckle, E.D., "PMBD - A Project Master Database for Software Engineering Environments," in *Proceedings of the 8th International IEEE Conference on Software Engineering*, pages 150-157, London, England, Aug. 1985.

[5] Beyer, Hugh R., *Proposal For Extending Dictionary Standards to Support CASE*, Proposed ANSI X3H4 "ATIS" Extensions, ANSI X3H4 Committee, March 14, 1990.

[6] *CASE Interface Services Base Document*, Digital Equipment Corp., Nashua, NH., September 1990.

[7] *ANSI X3H4, Draft Proposal American National Standard, IRDS*, American National Standards Institute, New York, 1986

[8] Boudier, G., Gallo, T., Minot, R., Thomas, I., "An Overview of PCTE and PCTE+," in *Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Engineering Environments,* Boston, MA., 1988, pages 248-257.

[9] Fleming, R., Wybolt, N., *Frameworks for CASE Tool Integration*, CADRE technical report CADRE Technologies, Inc.

[10] Cagen, M.R., "The H.P. SoftBench Environment: An Architecture for a New Generation of Software Tools," *Hewlett-Packard Journal*, 41, 3, June 1990, pages 36-47.

[11] Mercurio, V.J., Meyers, B.F., Nisbet, A.M., Radin, G., "AD/Cycle strategy and architecture [sic]," *IBM Systems Journal*, Vol. 29, No. 2, 1990, pages 170-188.p

[12] Taylor, N., Belz, F., Clarke, L., Osterweil, L., Selby, R., Wileden, J., Wolf. A., Young, M., "Foundations of the Arcadia Environment Architecture," in *Proceedings of the ACM SIGSOT/ SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, Boston, MA., 1988, pages 1-13.

[13] Feiler, P., Dart, S., Downey, G., *Evaluation of the Rational Environment*, Technical Report, CMU/SEI-88-TR-15, ADA198934, Software Engineering Institute, Pittsburgh, Pa., July 1988.

[14] Teitelman, W., Masinter, M., "The Interlisp Programming Environment," *IEEE Computer*, Vol. 14, No. 4, 1981.

[15] Dowson, M., "ISTAR - An Integrated Project Support Environment," in *Proceedings of the 2nd SIGSOFT/SIGPLAN Symposium on Practical Software Development Environments*, pages 27-33, December 1986.

[16] Bourguignon, J.P., "Structuring for Managing Complexity," in *Managing Complexity in Software Engineering*, ed. Mitchell, R. J., Peter Peregrinus Ltd., 1990.

[17] *Version Management Common Services*, in PACT documentation, G.I.E. Emeraude, 38 Bd Henri Selier, 92154 Suresnes, France.

[18] Reiss, S., "Interacting with the FIELD Environment," *Software Practice and Experience*, Volume 20, June 1990.

[19] Graham, M., Miller, D., *ISTAR Evaluation*, SEI Technical Report, CMU/SEI-88-TR-3, ADA201345, Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, Pa., 1988

[20] Nejmeh, B., *Characteristics of Integrable Software Tools*, INTEG_S/W_TOOLS-89036-N Version 1.0, Technical Report, Software Productivity Consortium, May 1989.

[21] Wasserman, A., "Tool Integration in Software Engineering Environments," in *Lecture Notes in Computer Science*, #467, Springer-Verlag, Fred Long, ed., ISBN 3-540-53452-0.

[22] Balzer, R., "Experiencing the Next Generation Computing Environment," in *Lecture Notes in Computer Science*, #467, Springer-Verlag, Fred Long, ed., ISBN 3-540-53452-0.

[23] *The Network Software Environment*, Sun Technical Report, Sun Microsystems, 1989.

[24] Leblang, D., Chase, R., "Computer-Aided Software Engineering in a Distributed Workstation Environment," In *Proceedings of SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, Pittsburgh, PA., April 1984, pp. 104-112.

[25] Clement, D., "A Distributed Architecture for Programming Environments," In *Proceedings of the Fourth ACM SIGSOFT Symposium on Software Development Environments*, Irvine, December 1990.

[26] Clow, G., Ploedereder, E., "Issues in Designing Object Management Systems," in *Lecture Notes in Computer Science*, #467, Springer-Verlag, Fred Long, ed.,ISBN 3-540-53452-0.

[27] Earl, A., *A Reference Model for Computer Assisted Software Engineering Environment Frameworks*, Technical Memo, May 25, 1990, Version 3.0 ECMA/TC33/TGRM/90/011.

[28] *Integrating Applications with FrameMaker*, Technical Manual, Frame Technology Corporation, December 1989, Part Number 41-00327-00.

[29] Leblang, D., Hare, D., *CIS 89-008 Draft Proposal Tool Integration Environment Interface, Working Draft, Rev. 1.3*, June 6, 1989.

[30] Buxton, J., Druffel, L., "Requirements for an Ada Programming Support Environment: Rationale for Stoneman," In *Proceedings of the IEEE Conference on Computer Software and Applications (COMPSAC80)*, Chicago, Illinois, October 1980.

[31] Hayes, F., "The Joy of Automatic Data Updates," *UNIXWorld*, November 1990, pp. 80-82.

[32] Zarrella, P., *CASE Tool Integration and Standardization*, SEI Technical Report, CMU/SEI-90-TR-14, December 1990, Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, Pa.

[33] Hitchcock, P., "The Process Model of the Aspect IPSE," Position Paper, In *Proceedings of the IEEE/ACM 4th International Software Process Workshop*, 1988, pp. 76-78.

[34] Strelich, T., "The Software Life Cycle Support Environment (SLSCE) A Computer Based Framework for Developing Software Systems," In *Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, Boston, Massachusetts, Nov, 28-30, 1988.

[35] Swaine, M., "Applications are Talking Too," *MacUser*, May 1991, pp. 239-242.

[36] Dart, S., Ellison, R., Feiler, P., Habermann, N., "Software Development Environments," in *IEEE Computer Magazine*, November 1987, pp. 18-28.

[37] Chappell, C., Downes, V., Tully, C., *Real-time CASE: the Integration Battle*, Ovum Ltd., 1989, ISBN 0 903969 48 3.

[38] Feiler, P., *Configuration Management Models in Commercial Environments*, SEI Technical Report, CMU/SEI-91-TR-7, March 1991, Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, Pa.

[39] Dart, S., *Spectrum of Functionality in Configuration Management Systems*, SEI Technical Report, CMU/SEI-90-TR-11, December 1990, Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, Pa.

[40] Kilter, U., "Performance of PCTE/OMS," in *PCTE Newsletter*, Number 6, April 1991.

[41] Jetty, J., "Emeraude PCTE Benchmarks," in *PCTE Newsletter*, Number 6, April 1991.

[42] Balzer, R., "Tolerating Inconsistency," Position Paper, In *Proceedings of the 5th International Software Process Workshop*, IEEE Computer Society Press, Kennebunkport, Maine, Oct. 10-13, 1989.

[43] Kaiser, G., "Mechanisms," Session Summary, In *Proceedings of the 5th International Software Process Workshop*, IEEE Computer Society Press, Kennebunkport, Maine, Oct. 10-13, 1989.

[44] *CCC: Change and Configuration Control Environment. A Functional Overview*, SoftTool Product Description, 1987.

[45] Huff, K., *Plan-Based Intelligent Assistance: An Approach to Supporting the Software Development Process*, Ph.D. Thesis, Computer and Information Science Department, University of Massachusetts, COINS Technical Report 89-97, September 1989.

[46] *Proceedings of the 5th International Software Process Workshop*, IEEE Computer Society Press, Kennebunkport, Maine, Oct. 10-13, 1989.

[47] Bass, L., Clapper, B., Hardy, E., Kazman, R., Seacord, R., "Serpent: A User Interface Environment," In *Proceedings Winter 1990 USENIX Technical Conference*, Washington, D.C., January, 1990.

[48] Yourdon, E., "DEC's CASE Environment," *American Programmer*, Vol. 3, No. 1, January, 1990.

[49] *OSF/Motif Style Guide Revision 1.1*, Open Software Foundation, 11 Cambridge Center, Cambridge, MA. 02142.

[50] *Reference Manual for Iris*, Incremental Systems Corporation Technical Report, Incremental Systems, Pittsburgh, PA. 1990.

[51] Evans, A., Butler, K., Goos. G., *DIANA Reference Manual*, SEI-DIANA-REF-MAN-83/SH, Tartan Laboratories Incorporated, Pittsburgh, PA., Feb. 28, 1983.

[52] *STARS Intermediate Language Assessment, IRIS/DIAIA Analysis*, Defence Technical Information Center (DTIC), STARS-RC-01430/001/00 Publication Number GR-7670-1158 Contract Number F19628-88-D-0031.

[53] *The User Interface Component of the Applications Portability Profile*, National Institute of Standards and Technology (NIST) Federal Information Processing Standards (FIPS) Publication 158, May 29, 1990, NIST, Tech-B64, Gaithersburg, MD., 20899.

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified		1b. RESTRICTIVE MARKINGS None			
2a. SECURITY CLASSIFICATION AUTHORITY N/A		3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release Distribution Unlimited			
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A					
4. PERFORMING ORGANIZATION REPORT NUMBER(S) CMU/SEi-91-TR-11		5. MONITORING ORGANIZATION REPORT NUMBER(S) ESD-91-TR-11			
6a. NAME OF PERFORMING ORGANIZATION Software Engineering Institute	6b. OFFICE SYMBOL (if applicable) SEI	7a. NAME OF MONITORING ORGANIZATION SEI Joint Program Office			
6c. ADDRESS (City, State and ZIP Code) Carnegie Mellon University Pittsburgh PA 15213		7b. ADDRESS (City, State and ZIP Code) ESD/AVS Hanscom Air Force Base, MA 01731			
8a. NAME OFFUNDING/SPONSORING ORGANIZATION SEI Joint Program Office	8b. OFFICE SYMBOL (if applicable) ESD/AVS	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER			
8c. ADDRESS (City, State and ZIP Code) Carnegie Mellon University Pittsburgh PA 15213		10. SOURCE OF FUNDING NOS.			
11. TITLE (Include Security Classification) Tool Integration and Environment Architectures		PROGRAM ELEMENT NO 63752F	PROJECT NO. N/A	TASK NO N/A	WORK UNIT NO. N/A

12. PERSONAL AUTHOR(S) Kurt C. Wallnau and Peter H. Feiler			
13a. TYPE OF REPORT Final	13b. TIME COVERED FROM `O	14. DATE OF REPORT (Yr., Mo., Day) May 1991	15. PAGE COUNT 40

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse of necessary and identify by block number)
FIELD	GROUP	SUB. GR.	CASE integration software environments
			environment frameworks tool integration

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The expanding CASE market is having substantial impact on software development environment technology in the area of environment support for tool integration.   Sharpened awareness of CASE integration requirements, particularly in the context of the large number of fully developed CASE tools, has resulted in a technology shift away from monolithic integrated project support environments (IPSE) derived from the Stoneman model in favor of highly distributed environments based upon a federation of environment services.   Federated environments promise environment framework support for the reuse of a large number of existing CASE tools and the development of highly interactive, tightly-integrated CASE environments.   The evolution of environment framework technology to support CASE federation is predicated on an improved understanding of the techniques and issues of tool integration.   One reflection of this improved understanding is recognition of the need to address integration mechanisms, tool semantic integration, and tool process integration as separate but related issues.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ■ SAME AS RPT ☐ DTIC USERS ■	21. ABSTRACT SECURITY CLASSIFICATION Unclassified, Unlimited Distribution	
22a. NAME OF RESPONSIBLE INDIVIDUAL John S. Herman, Capt, USAF	22b. TELEPHONE NUMBER (Include Area Code) (412) 268-7630	22c. OFFICE SYMBOL ESD/AVS (SEI JPO)

DD FORM 1473, 83 APR   EDITION of 1 JAN 73 IS OBSOLETE