Special Report

CMU/SEI-92-SR-8

An Annotated Bibliography on Integration in Software Engineering Environments

Alan W. Brown

CASE Environments Project

Maria H. Penedo

TRW

May 1992

Special Report

CMU/SEI-92-SR-8
May 1992

An Annotated Bibliography on Integration in Software Engineering Environments

Alan W. Brown

CASE Environments Project

Maria H. Penedo

TRW

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

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 work was created in the performance of Federal Government Contract Number F19628-95-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 52.227-7013.

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.rai.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, Attn: FDRA, Cameron Station, Alexandria, VA 22304-6145. Phone: (703) 274-7633.

Table of Contents

1. Introduction **1**

References **3**

An Annotated Bibliography on Integration in Software Engineering Environments

Abstract: This paper provides an annotated bibliography on integration in software engineering environments (SEEs). The aim is to provide readers with a source of information that can be used as the basis for more detailed studies in this area.

1. Introduction

There is currently a great deal of activity within the software engineering community in the area of providing automated support for aspects of the software development process. The diverse tools, systems, and environments that have resulted all have the basic aim of supporting (or automating) some part of the software development process with the expectation that the process will be be more predictable, less expensive, easier to manage, or produce higher quality products.

While many successes have been made in individual areas, perhaps the greatest challenge is to *integrate* these successes to produce an effective and integrated automated environment that supports the complete software development life cycle. While numerous terms have been coined, we denote those environments as *Software Engineering Environments (SEEs)*. There are many reasons why the individual successes have not been reflected in the success of SEEs. However, perhaps the underlying reason is that *amalgamation is not equivalent to integration*. The difficulty that arises is in precisely stating (both in terms of quality and quantity) why integration and amalgamation are different, and how integration can best be achieved. Factors to be taken into account include:

- Scale — the size and complexity of a SEE bring problems of management, control, and consistency.
- Lack of maturity — much of the technology that is being used in SEEs is immature. As a result, a sufficient body of relevant knowledge and expertise still needs to be built.
- Diversity — a wide range of requirements must be addressed that come from many different classes of potential SEE users (e.g., developers, managers, and tool builders), many types of possible application domains (e.g., data processing, real-time, and financial services), and many different structures of organizations that may wish to use a SEE (e.g., size, resources, and work habits).
- Technology base — a SEE attempts to tie together a collection of different technological bases into a consistent system. For example, a SEE can be seen as an extended operating system, a complex database application, a very high-level programming language, a diverse collection of user interface systems, or any combination of these.

When integration requirements are combined with other SEE requirements, the problems increase. For example, it is easy to see conflicts between the need for openness, tailorability,

and extensibility of a SEE on the one hand, and the need for consistency and predictability in a SEE on the other. These trade-offs must be evaluated within the context of integration.

Not unexpectedly, the result is a wide collection of views on what integration means in a SEE, how effective integration can currently be achieved, and what is important to achieve better integrated SEEs in the future. Perhaps the best that can be said is that *all* of these views of integration are meaningful and are necessary to gain a deeper understanding of SEEs and their architectures. Appreciating the range and complexity of the problem is essential to potential purchasers of SEEs, environment integrators, tool builders, and researchers investigating, developing and enhancing SEE technology.

This document contains an annotated bibliography on integration in software engineering environments. The references cited here describe, discuss, and analyze integration issues related to SEEs. They deal with: ways of characterizing integration; concepts, models, approaches, techniques and mechanisms in support of integration; and technologically and socially related issues. Our criterion in selecting papers has been to choose papers that we believe provide insight into integration beyond a single tool or system. Therefore, it is *not* intended that this bibliography cover every tool, system, or technology that claims to support, or embody, integration in a SEE. The annotations objectively summarize key aspects of the papers; the authors intentionally avoided comparisons and evaluations.

Inevitably in such a venture, relevant references are omitted, and others are included that may not directly meet the established goals. However, our hope is that this document provides an important source of information for all those interested in issues concerning tool and environment integration.

The authors welcome comments, criticisms, or suggestions for additions to this bibliography.

References

[1] Brown, A.W. and McDermid, J.A., Learning from IPSEs Mistakes. *IEEE Software*, 8(2):23–28, March 1992.

The authors argue that much of the current work on Integrated Project Support Environments (IPSEs) — particularly the work on Public Tool Interfaces (PTIs) — is of limited utility because it focuses too heavily on the latter. To support this argument they present an approach for quantifying the degree of integration achieved in an IPSE and apply that approach to some existing PTIs and IPSEs. Integration is defined in terms of five orthogonal dimensions: interface integration (consistency of user interface); technical integration (consistency of development methodology); tool integration (data sharing); team integration (isolation and communication between users on a team – e.g., through private and shared work areas); and management integration (derivation of management information directly from technical information produced by software engineers).

The authors claim that various degrees (or levels) of integration may be defined for each of the five dimensions. As an example, they propose the following levels of tool integration (from highest to lowest):

- Method level (e.g., sharing of knowledge about how the data is used in the process)
- Semantic level (e.g., sharing of data structure definitions and the meanings of operations on those structures)
- Syntactic level (e.g., sharing of data structures)
- Lexical level (e.g., sharing of formatted data)
- Carrier level (e.g., sharing of byte streams)

[2] Brown, A.W. and McDermid, J.A., On Integration and Reuse in a Software Development Environment. In F. Long, editor, *Software Engineering Environments*, Ellis Horwood, Chichester, England, 1991.

This paper explores the relationship between integration and reuse (i.e., reuse of existing tools). Integration is characterized in the same way as in the authors' 1992 paper [1]. The authors argue that the objectives of integration and reuse are inherently conflicting. They believe that in principle, reuse (without major re-engineering) can only be achieved between intrinsically compatible software tools. As a result, tools that exhibit higher levels of integration (e.g., at the semantic level) would have to be designed with a shared understanding of the data and the operations on the data. Hence, we could not expect to be able to reuse one pre-existing tool in the context of another unless both had been developed with common design goals.

The conclusion is that two fundamental objectives of IPSEs (i.e., high levels of integration and reuse) cannot both be achieved simultaneously. On the other hand, the authors note that the level of integration required between stages in development is much less than that required within stages. This suggests that there may be some opportunity for reusing toolsets that are already highly integrated,

then obtaining the necessary lower levels of integration between technical stages using some mechanism such as an IPSE infrastructure.

[3] Cagan, M., The HP SoftBench Environment: An Architecture for a New Generation of Software Tools, *Hewlett-Packard Journal*, 41(3), June 1990.

This paper describes the concepts and tool integration architecture of the Softbench Environment, an integrated environment designed to facilitate rapid, interactive program construction, test, and maintenance in a distributed computing environment. Its architecture is geared toward the integration and encapsulation of CASE tools.

The review is quite comprehensive, discussing issues of the approaches embodied in SoftBench in general terms, before describing the SoftBench approach in particular. The tool integration aspects described are used for:

- Tool communication, in support of tool collaboration in presenting a task-oriented environment to programmers
- Distribution of data, execution, and display, used by the tools to allow users to make effective use of the computational, file storage, and presentation capabilities available on the network
- Encapsulation, which provides the above mechanisms to tools without requiring access to the tools' source code

[4] Clemm G. and Osterweil L., A Mechanism for Environment Integration, *ACM Transactions on Programming Languages and Systems*, 12(1):1–25, January 1990.

This paper discusses the tool integration philosophy, the architecture and lessons learned with the use of the Odin environment integration system. The Odin approach is based on a central object store and includes features such as the typing of software objects, the composition of tools out of modular tool fragments, the optimization of the storage and rederivation of software objects, and the isolation of tool interconnectivity information in a single centralized object. Odin can be thought of as an interpreter for a high-level command language whose operands are the various large-grained software objects in the data repository and whose operators are tool fragments. Odin enables tool integrators to specify how the tool fragments and object types relate to each other.

There are two central constructs: a derivation graph, which specifies type and tool interconnections, and a derivation forest, which specifies how actual instances of types are related to each other. Those concepts are described, together with the Odin specification language and Odin request language. The experience acquired by projects using Odin to integrate tools is also described.

[5] Fernstrom, C. and Ohlsson, L., Integration Needs In Process Enacted Environments, *Proceedings of the 1st International Conference on the Software Process*, Redondo Beach, CA, USA, October 21-22, 1991, IEEE Computer Society Press, Los Alamitos, CA, 1991.

This paper describes the interaction between elements of a process-enacted environment and elaborates on the need to achieve integration and at the same

time independence of these elements. Tool execution and communication via process control is also considered a form of environment integration. The paper discusses the underlying concepts of process-enacted environments from the users' standpoint, the means of implementation, and the specific requirements process enactment puts on tools. It then presents an experimental SEE, developed by Cap Gemini Innovation within the context of the Eureka Software Factory (ESF), which exhibits a number of the required characteristics. The authors do not characterize tool integration per se, but they do discuss the general problem of achieving integration while retaining independence between tools.

[6] Fleming, R. and Wybolt, N., CASE Tool Frameworks, *Unix Review*, 8(12):24–32, December 1990.

This paper is an extended version of [27]. The main additions are a historical context for the work on integration by reviewing the Ada environments work in the Stoneman report, followed by a more detailed examination of possible CASE integration technologies (such as databases, data interchange formats, and message passing). A number of standards efforts appropriate to tool integration are also reviewed.

[7] Garlan, D. and Ehsan, I., Low-Cost, Adaptable Tool Integration Policies for Integrated Environments, *Proceedings of the 4th International Workshop on Computer-Aided Software Engineering*, Irvine, CA, ACM SIGSOFT 15(6):1–10, December 1990.

This paper demonstrates how tool integration based on message passing can be extended/adapted to allow dynamically configurable policies of tool interaction. This consists of augmenting the message server of the Field environment [16] with a mechanism for determining how to decode messages sent between tools. The result is that when the message server determines that a particular tool is interested in a message, a set of policy rules is consulted to determine what action to take. Such policy rules are independent of the tools, and can be amended without a need to change either the message server or the tools.

The Forest system — an implementation of this mechanism — is described, together with its facilities for multiple-user policy definition support. The approach is also compared to similar approaches; its main benefit is to provide the capabilities of more general and costly approaches with little effort.

[8] Harrison, W., Kavianpour, M., and Ossher, H., Integrating Coarse-Grained and Fine-Grained Tool Integration. *IBM Research Report No. RC17542*, IBM T.J. Watson Research Center, Yorktown Heights, N.J., 1991.

This paper discusses integration of coarse-grained and fine-grained systems. The granularity refers to both the data items being recorded and manipulated, and the control operations between objects.

The problem of coarse and fine-grained integration is examined in isolation, taking PCTE as a typical system that provides coarse-grained data integration through objects that are at the file level, and weak linkage between processes through message queues. The fine-grained example used is an Object-Oriented Database (OODB), which allows much smaller-sized objects (at the

single program-statement level), and has control through message invocation of the operations encapsulated with the data objects.

The paper then describes in more detail how the PCTE system could be extended with an OODB to provide both coarse and fine-grained integration. The aim is to provide a migration path to the fine-grained support that the authors believe will be the future for environments.

[9] Lewis, G.R., CASE Integration Frameworks, *SunTech Journal*, 3(5):50–51, November 1990.

This paper describes a short, but useful summary of the possible approaches to tool integration together with a discussion of the reasons why a tool integration standard is important, and requirements on practical tool integration standards. It discusses data and control integration aspects, including:

- Data linkage, i.e., establishing semantic relationships between pieces of information maintained by different tools
- Data interchange, i.e., transferring information between tools in some mutually agreed representation
- Data sharing, i.e., tools accessing data in a mutually accessible place
- Interprocess control, i.e., a tool causing some action to be performed in another tool
- Meta control, i.e., a co-ordinating agent to sequence tool actions
- Advanced data linkage support, i.e., using general underlying data interchange and interprocess control mechanisms

Useful diagrams summarizing the different ways in which tools can exchange information are provided.

[10] Mi, P. and Scacchi, W., Process Integration in CASE Environments, *IEEE Software*, 8(2):45–53, March 1992.

This paper defines process integration as a way to represent development activities explicitly in a software process model to guide and coordinate development and to integrate tools and objects; their implementation strategy is to realize process integration using existing CASE environments or tools.

The advantages of process modeling and enactment are discussed, and a particular software model and enaction mechanism is presented. Different process-based SEE user interfaces are illustrated, distinguishing between a software developer and a process manager's needs.

These ideas are then illustrated with the Softman experiment — a migration of an existing CASE environment into a user-interactive process driven environment, by instantiating their process model into the Softman process model and integrating Softman's object model and tool set. Details of the resultant system are described.

[11] Morris, E., Feiler, P., and Smith, D., Case Studies in Environment Integration, Technical Report CMU/SEI-91-TR-13, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, May 1991.

This paper provides a useful set of observations with respect to integration, based on case studies of four software engineering environments. The systems studied are: the Boeing Automated Software Engineering Environment (BASE), the Rome Airforce Laboratory funded Software Life Cycle Support Environment (SLCSE), the Verdix VADS APSE, and Hewlett Packard SoftBench environment [3]. It examines the environment standards, implementations, and technology that proved useful in the integration of tools into these environments.

The observations are concerned with the current state of tool and environment integration and trends in integration; they deal with aspects of: tool maturity, vendor egocentrism, conservative integration, virtual interfaces, user interface, data interchange, process support, framework support, data granularity, and programming in the large. Those observations were also based on surveys with environment and tool builders at two workshops. Key trade offs for SEE builders and users are also identified.

[12] Nejmeh, B., Characteristics of Integrable Software Tools, INTEG_S/W_TOOLS-89036-N, Version 1.0, Software Productivity Consortium, Inc., Herndon, VA, 1989.

This paper discusses four dimensions of integration in a software development environment: data integration; presentation integration; interoperability integration; and process integration. They are variations of Wasserman's definitions [25].

This paper also describes characteristics of *integrable* tools, i.e., characteristics that make a tool easy to add to Software Development Environments, and discusses those characteristics as they relate to the four dimensions of integration.

The integration characteristics were classified into four categories: environment sensitivity (e.g., location independence), characteristics to allow a tool to peacefully co-exist with other tools; interoperability (e.g., data import), characteristics to facilitate interoperability among tools; extensibility and adaptability (e.g., object schema extension), characteristics to facilitate tool extension or adaptation; and standards characteristics (e.g., portability) related to conformance to standards.

[13] Oliver, H., Adding Control Integration to PCTE. In *Software Development Environments and CASE Technology*, A. Enders and H. Weber, editors, number 509 in Lecture Notes in Computer Science, pages 69–80. Springer-Verlag, Berlin, 1991.

The main theme of this paper is the need to combine different forms of integration in a software engineering environment, particularly data and control integration mechanisms. The focus of the paper is a set of experiments carried out at Hewlett-Packard aimed at re-implementing SoftBench (which provides control integration) on top of PCTE (which provides data integration). Details of these experiments are given, followed by some thoughts on the likely usefulness of the resultant system.

[14] Penedo, M. H., and E. D. Stuckle, PMDB – A Project Master Data Base for Software Engineering Environments, *Proceedings of the 8th International Conference on Software Engineering*, London, August 1985.

This paper describes a model of the entire life cycle in terms of data and relationships; the model is denoted the Project Master Data Base (PMDB) Model. The PMDB model is a conceptual or logical life cycle schema to be used as a common data interoperability model. The model can also be used as part of a SEE user interaction paradigm, since it reflects users' perspectives of the data used in their project activities. Key concepts that influenced the design of this model were the need to: support large projects' needs, model the full life cycle from proposal to delivery, provide a generic model, avoid redundancy, and exclude implementation issues.

This paper reports the initial results of the PMDB work. It presents the PMDB model, an entity-relationship model consisting of 31 types (e.g., requirement, interface, software component, resource, task, person, milestone); approximately 200 attributes associated with the types and approximately 200 relationships between types. The paper also discusses technical issues in the definition and design of an environment database implementing such model.

[15] Perry, D. and Kaiser, G., Models of Software Development Environments, *IEEE Transactions on Software Engineering*, 17(3), March 1991.

This paper has a dual focus: first, it presents a general model of software development environments (SDEs) in terms of their structures, mechanisms, and policies; second, it explores the application of this model (the SMP model) to the classification of SDEs. Tool integration is discussed as it relates to structures, mechanisms and policies that characterize integrated SDEs. For instance, the authors note that the more complex structures required by integrated environments enable more sophisticated policies, but make it harder to integrate new mechanisms and policies into the environment. The authors classify SDEs according to the scale of development supported. They define four classes of SDEs, ranging from small-scale (i.e., supporting individual development) to large-scale (i.e., supporting development across multiple projects). Each class of SDE is analyzed in terms of its prevalent structures, mechanisms, and policies. This same kind of analysis is also applied to an existing taxonomy of SDEs that is based on historical trends.

[16] Reiss, S., Connecting Tools Using Message Passing in the Field Environment, *IEEE Software*, 7(4):57–66, July 1990.

This paper describes an approach to tool integration based on message passing, and the system Field developed as a testbed for this approach. Reiss's thesis is that the approaches to tool integration based on a central database are too complex and monolithic to provide either the speed or flexibility required in a software development environment. He believes that providing a central message server facility allowing tools to communicate via message passing is simple, flexible, and adequate to support most forms of tool collaborations.

The Field environment is described in some detail. It is a system supporting three purposes: a programming environment for teaching undergraduates, a research programming environment, and a testbed for developing new tools. It combines the selective broadcast communication mechanism, an annotation editor, and a set of specialized interactive analysis tools.

The Field environment has been influential as the catalyst for a number of commercial products, including Hewlett-Packard's SoftBench, Sun's ToolTalk, and DEC's FUSE.

[17] Schafer, W. and Weber, H., The ESF-Profile, in *Handbook of CASE*, P. Ng and R. Yeh, Editors, Van Nostrand Reinhold, New York, 1989.

This paper presents an early description of the profile of the Eureka Software Factory (ESF) project from the authors' perspectives. ESF is a large European research effort aiming at developing flexible concepts for the integration of software development support tools that have been built at different sites, and the application of those concepts to build customized software factories.

This paper characterizes the ESF factory concept, which includes a reference architecture, a reference model, and an instantiation procedure that together allow ESF instances to be derived. The ESF reference architecture fixes the spectrum of possible ESF instances, whereas the reference model defines different interfaces on different abstraction levels for each architectural component. According to the authors, the result is that any new tool can be plugged in at the most suitable level of abstraction and does not need to be adapted to one specific standard interface.

Five degrees of integration are defined, which are partially ordered in that one may achieve a certain degree of integration only if certain lower degrees of integration have already been achieved:

1. Machine-integrated, enabled by common network services

2. Object-integrated, enabled by a common object store and a common I/O manager

3. Process-integrated, enabled by the specification of a software development process

4. Method-integrated, enabled by a combination of object-integrated and process-integrated, to achieve sharing of information at the semantic level

5. Method-uniform, enabled by a single development method throughout the life cycle

The ESF reference model is claimed to be a natural extension of the ISO/OSI model. To support integration of existing tools, the ESF reference model defines a hierarchy of abstract levels of communication between tools. ESF intends to define/reuse and standardize those interaction protocols, also called Software Buses [18]. Tools are integrated at different levels according to those interaction protocols.

[18] Schuelke, F. and Holtkamp, B., The Software Bus — Communication Aspects, University of Dortmund Technical Report, Dortmund, Germany, March 1991.

This paper presents a communication-oriented view of the Software Bus, which is the global integration mechanism in a Factory Support Environment (central concept of the Eureka Software Factory (ESF) program). The Software Bus goes beyond communication to be a mechanism that enables the plugging of

components that make up a software factory instance. Four major service areas are to be supported by the Software Bus: platform management, component management, service management, and tool management.

The paper identifies needs and requirements for the Software Bus. Emphasis is placed on the communication capabilities of the Software Bus and how MUSE, a University of Dortmund system for managing the interoperation of components in a distributed environment, can be used as a Software Bus. The paper also describes the software factory reference framework (SFRF), which defines the means of integration for Software Factories. It is defined as a four-stage, multi-layer framework consisting of the following stages:

1. *interworking* of human users or user groups

2. *interaction* of human users and the computing system

3. *interoperation* of different software capabilities

4. *interconnection* of different hardware platforms

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

This paper describes the Portable Common Tools Environment (PCTE) and some aspects of the PCTE Added Common Tools (Pact) environment with respect to the use of the PCTE interfaces. The history of the PCTE effort is described, together with an overview of the PCTE interface itself. PCTE aimed to define a public tool interface to be used as a portability interface and integration support for the ESPRIT software tools. The Pact environment was built on top of the PCTE interface; it includes a set of common services and a set of integrated tools. The Pact common services include version management, data query and manipulation primitives, and documentation structure management.

The experiences with the Pact project are discussed, including some areas that led to changes and additions to the PCTE interface itself. Four areas of integration were identified: the common services, the use of the object base, user interface uniformity, and tool composition.

[20] Thomas, I., Tool Integration in the Pact Environment, *Proceedings of the 11th International Conference on Software Engineering*, Pittsburgh, PA, May 16-18, 1989, IEEE Computer Society Press, Washington, DC, 1989.

This paper describes the PCTE Added Common Tools (Pact) environment, a software engineering environment built on the PCTE interfaces. The paper describes the Pact architecture, which includes a set of common services and a toolset built using those common services. The Pact common services include version management, data query and manipulation primitives, and documentation structure management. The toolsets discussed in the paper are: configuration management, project management and document preparation tools. It also includes a rationale for the architecture and lessons learned about the benefits and disadvantages of a common service approach to environment architectures based on Public Tool Interfaces.

[21] Thomas, I., Goals and Requirements for Integration Frameworks, *Proceedings of the 4th International Workshop on Computer-Aided Software Engineering*, Irvine, CA, ACM SIGSOFT 15(6):201–203, December 1990.

This short paper looks at the requirements for a tool integration framework. It is based on the author's experiences of developing the PCTE interface, and the requirements that were imposed on it. In summary, the requirements identified for a tool integration framework are: completeness of the interface; a clear migration path for existing tools; multi-language development support; a separation of policy issues from mechanistic ones; support for distributed development; scalability of services for large-scale development; and clear identification of the scope of services and the class of end-user they are intended to support.

[22] Thomas, I. and Nejmeh, B., Definitions of Tool Integration in Software Engineering Environments, *IEEE Software*, 8(2):29–35, March 1992.

This paper presents a conceptual framework for helping to determine how well a tool is integrated within a software engineering environment. It defines integration as a property of the relationship between elements of an environment. However, it focuses on integration relationships between tools.

Those properties are defined in the context of four well-known types of integration: presentation integration, data integration, control integration, and process integration. Those properties, described together with support mechanisms that can improve them, are:

- From the perspective of presentation, *appearance and behavior, and interaction paradigm*
- From the data integration perspective, *interoperability, non-redundancy, data consistency, data exchange, and synchronization*
- From the control integration perspective, *provision and use*
- From the process integration perspective, *process step, event, and constraint*

The authors note that they expect this set of proposed relationships and properties to be extended and refined as the understanding of tool integration grows.

[23] Wallnau, K.C. and Feiler, P.H., Tool Integration and Environment Architectures, Technical Report CMU/SEI-91-TR-11, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, May 1991.

Both an overview of approaches to tool integration and a review of the different types of tool integration are provided in this paper. The paper compares the centralized approach of many Integrated Project Support Environment (IPSE) solutions, to the more ad hoc, "point-to-point" approaches taken by many CASE tool vendors in forming tool coalitions. The conclusion of the paper is that these two approaches have advantages that can be combined in future systems. This combined approach, called "federated CASE", can come about by recognizing the need for tool integration at three distinct abstract levels: the tool mechanisms level, the tool semantics level, and the tool process level.

[24] Wasserman, A.I. and Pircher, P.A., Visible Connections, *Unix Review*, 4(10):62–73, October 1986.

This paper looks at the need to provide an “open architecture”, believing that this implies that system components must be connected in visible ways. Examples of open architectures are discussed, including Unix and others based on a common file format. The need for software architectures to have equally “visible connections” is then discussed. The openness is described at four levels: the environment level, tool level, data repository level, and file interface level. The *Software through Pictures (StP)* product is then described in terms of its openness at each of these four levels.

[25] Wasserman, A.I., Tool Integration in Software Engineering Environments, *Proceedings of Software Engineering Environments: International Workshop on Environments,* Chinon, France, September 18-20, 1989, F. Long, editor, number 467 in Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1990.

This paper proposes five types of tool integration and discusses levels of integration within each type. The types of tool integration are:

1. Platform integration, i.e., the set of system services that provides network and operating systems transparency to applications

2. Presentation integration, i.e., tools that share a common “look and feel” from the user’s perspective

3. Data integration, i.e., support for data sharing across tools

4. Control integration, i.e., support for event notification among tools and the ability to activate tools under program control

5. Process integration, i.e., support for a well-defined software process

The paper presents a structure for building tools that advocates isolating the functionality of the tool from its internal mechanisms so as to support adaptation to emerging standards. This paper also presents an overview of the integration techniques employed in IDE’s Software through Pictures environment, using the concepts described in the paper.

[26] Wileden, J., Wolf, A., Rosenblatt, W., Tarr, A., Specification Level Interoperability, *Proceedings of the 12th International Conference on Software Engineering*, Nice, France, 1990.

This paper describes an approach for component interoperability denoted as Specification Level Interoperability (SLI), which provides support, as indicated by its name, at the specification level. The approach is a high-level, representation-independent approach for combining software components written in different languages or that run on different machines.

The paper discusses some representation-level interoperability (RLI) mechanisms and the SLI approach, which consists of four components:

1. A unified type model, i.e., a notation for describing the entities to be shared by interoperating programs

2. Language bindings, which connect the type models of the languages to the unified type model

3. Underlying implementations, which realize the types used by the different interoperating programs

4. Automated assistance, which eases the task of combining components into an interoperable whole

The SLI approach does depend on RLI mechanisms. However, the paper claims that, if fully realized and properly used, SLI can be a type-safe, extensible mechanism for tool/component interoperability.

The paper also describes a prototype realizing the model and experience with the use of the prototype. It consists of a unifying type model called UTM-0, bindings for Lisp and Ada, an implementation strategy, and a UTM-0 automated generator.

[27] Wybolt, N., Perspectives on CASE Tool Integration, *ACM Software Engineering Notes*, 16(3):56–60, July 1991.

This paper provides a short overview of key technical, political, and economic issues surrounding CASE tool integration. It looks at some of the common questions of CASE integration such as “where does the data reside?” “How do the tools communicate?” and “Who carries out the integration, and who maintains it?”. Some of the technologies proposed to answer these questions are then reviewed, before concluding by looking at some of the real-world practicalities of providing an integrated environment.

[28] Wybolt, N., CASE Repositories and Tool Integration — A Reality Check, *Proceedings of the 4th International Conference on Software Engineering and its Applications*, Toulouse, France, December 1991.

This paper discusses data repositories in the context of tool integration and enumerates a number of potential problem areas from the perspectives of the CASE tool supplier and end-user. It then surveys alternatives to the repository and its problem areas, including the integration of links and message switches. In link technology, the data repository is replaced by a network-wide database of links between objects that may reside in the same or different databases (e.g., NSE’s Link Service Facility). Message switch technology embodies a mechanism whereby tools send and receive messages (e.g., HP’s SoftBench).

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

This paper discusses the current barriers to integration of CASE tools and describes some actions being taken to overcome them. It addresses the issues, problems and resolution efforts related to CASE integration and standardization from the users’ perspective. For discussion purposes, CASE tool integration is presented according to five areas: single-vendor (internal) tool integration, multiple-vendor (external) tool integration, operating environment integration, development process integration, and end-user integration.

Issues related to standardization are also discussed, and a summary of key standardization efforts and their status is presented. The author sees standardization efforts as a possible path to tool integration, but conjectures that market-driven de facto standards may become the cornerstone of future CASE tool evolution.

Acknowledgements

The authors would like to acknowledge A. Goldstein, who contributed to an earlier draft of this bibliography (under STARS sponsorship), and P. Oberndorf and I. Simmonds, who provided additional recommendations.

This work was supported in part by the Software Engineering Institute, sponsored by the U.S. Department of Defense, and in part by the Defense Advanced Research Projects Agency/Information Systems Technology Office, DARPA Order 7314, issued by the Space and Naval Warfare Systems Command under contract N00039-91-C-0151.