META-ENVIRONMENTS FOR SOFTWARE PRODUCTION

ANTHONY S. KARRER
Computer Science Department, Loyola-Marymount University
Los Angeles, CA 90045-2699 USA

WALT SCACCHI
Information and Operations Management Department
and
Center for Software Engineering
University of Southern California
Los Angeles, CA 90089-1421, USA

December 1994

Researchers who create software production environments face considerable problems. Software production environments are large systems that are costly to develop. Furthermore, software production environments which support particular software engineering methods may not be applicable to a large number of software production projects. These conditions have formed a trend towards research into ways which will lessen the cost of developing software production environments. In particular, the trend has been towards the construction of meta-environments from which specific software production environments can be created. In this paper, we attempt to categorize more than 60 meta-environment efforts. For each of the categories, we review research efforts which illustrate different approaches within that category. We conclude by presenting an emerging common thread of requirements which links this field together.

1 Introduction

Software engineering addresses the complex problems associated with software production. Many software engineering methods, such as structured design and object-oriented analysis, have emerged to ease these problems. Recently, there has been considerable attention paid to the application of these methods through software production environments in order to evaluate the effectiveness of each method [7, 43, 67]. A *software production environment* (SPE) is a system consisting of a software infrastructure providing a common operating environment for software tools, a set of tools, and an interface which provides users with access to the environment's capabilities.

Both software production organizations and research organizations have found that there are considerable problems in using SPE technology. One underlying problem is that software production environments are large systems and, without sufficient support, are costly to engineer and build. This result is not surprising, especially given that software production environments are large software products. Coupled with this problem, researchers have found that little is known about the requirements for a good SPE. Many researchers believe that the only way in which the requirements can be found is by developing, using, and evolving environments. Unfortunately, such

an evolutionary approach will be costly because of the time and effort required for creating new SPEs, and because as better SPEs are developed, users are likely to change the manner in which they use the SPE [67]. Worse yet, each software production project may have different requirements for an SPE [10]. These factors imply that in order to research the requirements for SPEs and to support the requirements of individual projects we must reduce the cost of creating SPEs.

Because SPEs are themselves software products, we can consider the software production process used to construct SPEs. SPE construction processes have taken many forms, with early SPEs being constructed using a simple program-debug-test process based on a single programming language. *Meta-environments* [53, 55, 80] are emerging as a means of constructing SPEs with significantly more powerful constructs than are generally available in programming languages. Meta-environments have also been called generic environments [36] and environment generators [25, 31, 95]. The distinction between these terms is somewhat vague. As such, we will use the term *meta-environment* to include generic environments, environment generators, and other approaches to environment construction.

We find a meta-environment consists of a *construction model*, a *transformation* of instances of the model into an SPE, and a *process* for using the construction model and transformation to create an SPE. An *environment specification* is an instantiated construction model which specifies the intended environment. A *meta-environment construction process* is the collection of tasks, agent roles, resources, tools, and their relationships which specifies the procedures used to create viable software production environments.

In general, meta-environments are designed to overcome two major issues. The first issue is that SPEs are themselves large software systems and are extremely costly to create without the support provided by meta-environments. The second issue is that a high level of assistance in reducing the cost of environment creation often requires assumptions about the form of the resulting environment. These assumptions reduce the ability of a meta-environment to support many different kinds of software engineering methods.

In the rest of this paper, we examine the trend for creating meta-environments. First, we attempt to categorize current meta-environment approaches. For each of the categories, we review research efforts which illustrate different approaches within that category. We conclude by presenting the common thread which links this field together.

2 Related Research

Software engineering environments were initially constructed as monolithic systems, using a programming language as the construction model. Two of the more notable examples of this approach are the programming support environments for Smalltalk [37] and Interlisp [88]. Both environments support a single method of software production: interpretive, incremental program development. While the monolithic approach can be applied to construct environments which support virtually any software production method and programming language, the cost of productng an environment using this approach is generally prohibitive for most research organizations.

Since the monolithic approach, a variety of methods have been suggested for lessening the cost of construction and improving the utility of environments. We have categorized the approaches to environment construction into the following five classes:

- *Environment frameworks* support a set of low-level services including object management,

control management and sometimes user-interface management.

- *Customizable environments* provide high-level core environment capabilities and a means of customizing those capabilities.
- *Process modeling* provides a meta-model which can be instantiated to specify the activities, developers, resources, artifacts, and their relationships which together form the processes of the environment.
- *Process programming* provides a programming language oriented towards the description of processes as the basis of constructing process-driven environments.
- *Tool integration* provides the means to combine tools into an integrated set of environment capabilities.

In the following sections, we further define each of these categories. We also present several research efforts in each category which serve both to give further insight into the basic category and also to illustrate the main issues which separate researchers within a category. Most research efforts in meta-environments actually combine technology from more than one category. For example, many environment frameworks provide support for tool integration. Nonetheless, we have categorized and presented these efforts in order to illustrate the meta-environment trend as a whole.

2.1 Environment frameworks

One of the recent trends in the commercial and government sectors has been the development of environment frameworks which provide a set of computational services as a basis for environment construction. As a starting point for a survey of this research, it is a good idea to mention that considerable attention is being directed to the definition of the kinds of services which environment frameworks should address. Much of this work has been done by creating reference models including the Conceptual Environment Architecture Reference Model (CEARM) [70], the ECMA Reference Model [23], and the NIST Reference Model [94]. [48] used one of these reference models to survey current environment framework research. This survey showed the usefulness of a reference model as a conceptual framework for comparing the capabilities of frameworks. Much of this section is derived from that study.

These reference models suggest that frameworks should address Object Management Services, User-Interface Management Services, and Life-cycle Process Management Services (also called Environment Management [70] and Task Management [23]). The Object Management Services (OMS) provide for persistent objects and relationships as opposed to files and directories traditionally supported by operating systems. The Life-Cycle Process Management Services (LCPMS) provide a means for enacting change upon the current state of life-cycle objects. Most LCPMS are based on the operating system process. However, the LCPMS combines operating system processes with additional modeling services to provide a more powerful control mechanism. The User-Interface Management Services (UIMS) provide basic mechanisms for defining user-interfaces and associating objects graphically depicted with objects and actions within the environment.

These reference models suggest that environment frameworks must provide construction models for modeling data, control, and user-interface. While other meta-environment approaches require,

for example, some kind of translation of a language or environment specification into an environment, a similar transformation of the framework's models is not needed since the framework models are used as the architectural basis of the environment. The examples that we present next show a variety of data and control models.

Atherton Technology's Software BackPlane [68] is an integration and portability framework designed to assist in building SPEs independent of tools, methodology, or languages. It provides a generic operating system, an object-oriented OMS and a Macintosh-like user interface. The Software BackPlane's OMS provides an object-oriented software interface and a set of predefined classes which give the database its form. The OMS uses the concept of Generic Messages to invoke methods for new/free, open/close, checkin/checkout, and others which are applicable to virtually all objects. This approach allows users to invoke similar tools using identical messages.

Some tool integration is addressed through the object-oriented integration techniques for defining new classes or attaching existing tools to previously defined classes as methods. Tools created without using the Software BackPlane as its persistence mechanism store their data using the native operating system. Translation of their data and rehosting to the Software BackPlane can be accomplished by creating a method which does both a translation from a file into a set of objects and tool invocation over those objects.

Atherton has also provided a high-level specification of an environment construction process called the "Environment Customization Plan". This process suggests the creation of user-defined methods as a form of process programming, tool definition, data definition, and user/role definition.

Gaia [90] embodies an approach similar to the Software Backplane by providing an object-oriented OMS as its cornerstone. It lacks some of the integration techniques present in the Software Backplane. It also lacks a plan for using object-oriented technology to instantiate an environment.

CAIS-A [60, 18, 71], the latest version of CAIS, is defined in the military standard report MIL-STD-1838A, under the control of the Ada Joint Program Office (AJPO) within the U.S. Department of Defense. CAIS-A is a set of Ada interfaces which are designed to act like a high-level virtual operating system. It provides an object management system (OMS) and process control based on the OMS. The OMS is a distributed database providing a node model which is similar in flavor to an entity-relationship-attribute (ERA) model with type inheritance. The node model is used to formalize the structures and capabilities of the OMS. Processes in CAIS-A form a tree of parent-child processes. All process information is kept in the OMS. Processes can communicate through special OMS-based queues.

The Portable Common Tool Environment (PCTE) [89] is a Public Tool Interface (PTI) which can be used as a basis for integrating tools as part of the development of an SPE. The PCTE PTI consists of software interfaces to services of an OMS, LCPMS, and UIMS. The Object Management System is a distributed database based on an ERA model with multiple type inheritance. The object types are modeled by a single overall schema which is actually comprised of a set of possibly overlapping Schema Definition Sets (SDS). PCTE models the static context of processes which are used to control process invocation. Inter-process communication is achieved through the OMS, message queues, pipes and signals. Monitoring of process execution is also available. PCTE has defined a set of user-interface management primitives, as well as extensions to support production of real-time software through the PCTE+ and PACT versions of PCTE.

The Sun Network Software Environment (NSE) [1] is a network-based object manager and tool integration facility. NSE is primarily a Unix environment with additional support for object management, configuration management, version management, distribution, environment management,

target generation, and environment front-ends. The object manager is oriented around a specific view of version and configuration control based on an optimistic concurrency control mechanism. Tools are integrated into NSE by using NSE to control their objects (files) and by adding the tools to the NSE environment front-end. More recently, an object-based tool integration facility called ToolTalk has been introduced to support this capability

The NSE object manager is not a general purpose object manager in the same vein as PCTE, CAIS, and the Software BackPlane. It provides no extensibility of the base classes. It does support an extension to Unix process management by allowing triggers associated with NSE commands to perform some of the environment tasks.

The Software Life-Cycle Support Environment (SLCSE) [84] is a framework which provides a common database, a set of common schemata, and tools. The common schemata and tools are designed to support a MIL-STD 2167A model for documenting the software production life-cycle. The 2167A model is based on the waterfall model of software production based primarily on the production of documents after each step in the life-cycle. The SLCSE framework also supports specification of alternative schemata and alternate life-cycles.

RDPE3 [65] is a programming environment which is primarily oriented around the support of adaptation and extension of environments. RDPE3 uses an extended object-oriented, fine-grained programming paradigm to represent program fragments. Thus, RDPE3 focuses on a smaller component size as compared to PCTE, CAIS-A, etc. Furthermore, RDPE3 suggests a different integration scheme than those suggested by other frameworks, one in which the function of the integrated components must be accessible.

The ALMA [55] generic environment provides the capability to define a life-cycle support database schema based on the entity-relationship model. While the ALMA environment has capabilities similar to other frameworks, it specifically presents a plan for the use of the framework as both a meta-environment and a software production environment.

The Distributed System Factory (DSF) project is developing a multilayer infrastructure of software services which can integrate multiple data, control, presentation, and process models that can span a distributed wide-area network [77]. The DSF infrastructure provides a distributed hypertext (DHT) framework to integrate heterogeneous software object repositories (e.g., those with different data models) [63] with forward and reverse software engineering tools [16], graph-based editors and user interfaces [49], process-driven user interfaces with partially order tool invocation sequences [59], and a knowledge-based process modeling and simulation environment [58]. Using these mechanisms and components, it is possible to semi-automatically generate multirole, process-driven software production environments for a hierarchy of concurrent production processes [31, 59].

The EUREKA Software Factory (ESF) [28] is a large-scale pan-European research effort aiming at the development of an integration framework for a meta-environment that can accomodate the interoperation of various tools and the interactions of people working together with these tools according to a process meta-model [2]. The heart of the ESF architecture is a Software Bus (SWB), which is an abstract communication channel hiding distribution and supporting the exchange of data and control information using abstract data type interfaces. User-interaction components (UIC) and service components (SC) communicate across the SWB. UIC, which have no persistent data, communicate with the user and make requests on service components across the SWB. SC may not have a user-interface. One UIC and one SC taken together are essentially a tool in current environments.

The ESF kernel, K/2r [2], is not tied to the use of a central OMS as the basis of integration.

Rather, the ESF project uses process models as the basis of tool integration, although different schemes [19, 30] and modeling formalisms [21, 27] are being investigated. In this way, the ESF and DSF projects share many common goals, although their approaches differ.

Each of these environment frameworks provides services which are needed in virtually all software production environments. Access to an implementation of these services assists environment construction. In general, environment frameworks are intended to maintain independence from any particular software production method. While environment frameworks provide some support to environment construction, the services they provide are generally low-level functions. There is a considerable gap between these services and the services provided by a full software production environment.

Existing environment frameworks seem to be split on the kind of data and control models to provide. The object-oriented camp with Atherton and Gaia suggest a single model for both data and control. The extended ERA camp with PCTE, CAIS-A, SLCSE, ALMA, DSF-DHT, and ESF-K/2r suggest a data model based on the ER or object-based model, and a control model based on concepts which are similar to operating system processes.

2.2 Customizable environments

Customizable environments are based on the existence of a high-level, fixed core of services or capabilities. Different customizable environments will provide different methods of customization and different core capabilities. The construction method provided by a customizable environment allows the user to specify a set of extensions or changes to the core capabilities. This specification is then combined with the core capabilities to form an environment. This approach is very similar to the approach used in most tool generators, such as those used for parser generators [92], language-directed editor generators [16, 64, 76], or extensible graph editors [47, 69, 79].

Customizable environments differ from frameworks in that they trade ease of instantiation for less generality. In other words, customizable environments will support a model which can be used to describe only a small portion of the entire environment. The remainder of the environment is fixed by assumptions built into the customizable environment. In particular, customizable environments will fix the process supported by the environment, but will allow particular parts of the process to be subsequently modified.

Several researchers have developed customizable environments centered around the use of structured editors. These customizable environments provide a construction model which can be used to describe the grammars for the languages which will be manipulated in the environment. The fixed part of the environment is the process of using a series of editors. For example, the Mentor programming environment [22] is founded on an abstract syntax tree representation of the program as the common information base. This syntax tree can be annotated by means of other abstract syntax trees in specialized languages. By allowing arbitrary annotations of nodes in the syntax tree, Mentor allows the extension of the environment.

GANDALF [64], the Synthesizer Generator [76], and SOFTMAN [16] provide customizable structure/text editors which allow an environment to be constructed as a suite of tightly integrated editors. Editors are customized through the specification of (a) the grammars of the languages which they edit, and (b) the attributes or action routines which support incremental semantic checking. GANDALF also provides for the specification of commands which invoke external tools integrated around centralized parse tree representations, while SOFTMAN provides for the integration of

generated multilanguage editors with an OMS.

TRIAD [51] is a customizable environment which supports a set of form-based, methodology-driven tools designed to operate over an extendible attribute grammar. This grammar is used to encode process information as well as syntax information. The grammar is used by TRIAD to construct various portions of the environment including a syntax-directed editor.

DRACO [62], Popart [92], GEM [38], KIDS [78], and Centaur [53] are generators of grammar-based or notation-based meta-programming environments. A meta-programming environment is an environment which assists in the creation of parsers and related tools which manipulate a particular language. Each generator accepts the specification of a language, then assembles a library of routines which can be used to operate over a parse tree or similar internal form of a program in the manipulated language. DRACO, Popart, KIDS, and Centaur are designed to support the development of domain language definitions using either (a) patterns, transformations, rules, or annotations for DRACO, Popart, and KIDS, or (b) algebraic and syntactic specification formalisms for Centaur. In addition, Popart is used in conjunction with Common Lisp Framework [4] and a generic process engine developed at USC/ISI [5], which accomodates the generation of rule-based programming and transformational implementation environments. The DRACO and KIDS environments provide similar capabilities, although they lack integration with a process engine. However, DRACO also supports the integration and composition of reusable components or tools, as described later.

Another approach to customizable environments is to provide a model which supports the specification of the objects to be manipulated during a software life-cycle process. The SOFTMAN environment [16] provides core capabilities which enforce a particular model of incremental software development and verification across life-cycle activity boundaries. It allows customization based on the specification of the formal attributes of life-cycle objects and specification of tools which operate on these objects. Thus, the specification refers to objects beyond those directly tied to the programming language. However, the primary tools in a SOFTMAN environment are customizable structure editors interfaced to a constraint-based OMS. The combination of the tools and the object repository can then track and verify the correctness attributes of objects that have been created or modified [14].

ISHYS [33] provides an interface to a hypertext-based information storage structure and to a structured documentation process. ISHYS supports the specification of the type, attributes, and composition of hypertext-based life-cycle documents based on the DIF software hypertext system [34]. It also supports the specification of a process model which describes the production and use of the documents by collaborating agents. The document formats and process model support multiple forms of group-based or team-based interaction and communication structures within a project and support multiple simultaneous projects.

META/GA [95] is a Life-Cycle Support System generator based on the specification of the format of information which must be processed in the generated environment. META/GA generated environments support a specific life-cycle approach to the creation of information processing systems. However, the META/GA user, the environment constructor, is able to specify the type of information which will be processed. This customizable environment attempts to help standardize the type of information and leave the life-cycle as general as possible.

The Metaview project [11, 82] takes an approach that is a hybrid of framework approaches and customizable environments. The models provided are based on an ERA model extended with constraints, transformations, and a graphical interaction model. These models are used to describe

the data manipulated by the environment as a whole and by generic tools. So both the environment framework as well as tools which operate over the environment are specified using this model.

There is growing interest in developing structure- or language-based editors for manipulating graph-based or graphic software specification notations. EDGE [69], TGE [49], MetaEdit [79], VSF [43, 72], and GEDL [47] are recent examples. In contrast to structured-text editors and related functionality generated by GANDALF or Synthesizer Generator, these graphic meta-editors provide multiple layouts of graphic information [47, 69], allow the specification or redefinition of user interface functionality [49, 47], and support a variety of object modeling, software process [49, 47] and method notations [43, 72, 79]. Each meta-editor employs a language-based specification notation for describing the iconic syntax and composition semantics to be supported in the target editors that are generated or instantiated. Finally, TGE, VSF, and GEDL have specification-level interfaces which allow them or their derived editors to be integrated with one or more repositories and environments, such as SOFTMAN's use of TGE-based graph editors [16, 77]. Subsequently, this provides another dimension for the extension of resulting meta-environments.

Overall, the primary advantage of customizable environments is the speed and relative ease of constructing high functionality environments. However, most customizable environments have lessened the burden of environment construction by adopting a particular approach to software development. This means that a customizable environment may only support certain kinds of software processes or development methods. Environments centered or driven by software production process models thus represent an alternative to this situation.

2.3 Process Modeling

In environments based on process models, a software process meta-model is used to circumscribe the family of processes the environment supports. Thus, a process model-based construction method must provide a *process meta-model* as the basis of implementing a specific life-cycle model. Instances of a meta-model describe active and inactive objects in an environment. Further, an instance of a meta-model allows the execution of a process-centered environment.

The GENESIS (from UC Berkeley) [73], Marvel [46], and Merlin [27] environments are among those that provide production rules and rule interpreters as the means for specifying and enacting software processes. A production rule has (1) a precondition, (2) an action part which is executed when the precondition is true, and possibly (3) a postcondition which becomes true once the action is executed. Environment construction using these systems consists of specifying the production rules for all of the activities or events which the environment should support.

PRISM [56], IPSE 2.5 [91], and its successor, PSS [12], provide process meta-models similar to Marvel and GENESIS. However, these efforts suggest a particular execution of the process models, which provides the environment to the user. In these environments, a process model execution facility serves as the means of invoking tools and presenting choices to environment users.

MELMAC [21] is a software process management environment based on the execution of high-level Petri-net representations of software process models [26]. The MELMAC research effort has led to the development of a prototype capable of enacting a software process model on top of a framework similar to the ones described in subsection 2.1. A major part of this prototype is the graphical user-interface, which depicts the Petri-nets and allows both the specification of process models and the enaction of process models.

The Software Designer's Associate (SDA) project [52] is a software design environment in which

the executable process descriptions are specified as a process model consisting of activities, composite activities, products and tools. SDA is unique in that it only addresses the early phase of design and that it uses a model-based approach to the design and construction of an environment.

The DSF project has been developing and using the Articulator [57], which is a knowledge-based environment for modeling, analyzing, and simulating software production processes. The Articulator uses a hierarchical object-oriented representation of agent roles, products (or resources), and tasks, which can be either partially ordered or triggered through rule-based mechanisms. Modeled agents communicate process task status and resource availability to one another through queries and messages. These queries can entail direct or deductive retrieval of process enactment information, or trigger either the symbolic execution of new processes or replay processes already enacted. Multiple concurrent processes can be enacted and agents can interact across processes. In addition, the environment maintains persistent information about the state of enacted processes as process transitions occur. This information is used to track the execution history of planned activities, and to facilitate incremental process rescheduling and replanning when unplanned or dynamic changes in the process occur [58].

The Articulator has been integrated with a process-based user interface and a process model interpreter (together called PBI) to provide a framework for developing process model-driven environments [59]. The Articulator, PBI, and an open software environment or set of software tools can the be used to construct and instantiate process-driven software production environments. That is, process models created with the Articulator are automatically transformed into process programs that can be enacted by PBI's process interpreter. When the process models specify tool bindings to process pre- or postconditions, data objects, and object repositories, then process-driven software environments can be automatically produced. In one study [59], the SOFTMAN environment [16] was used as the base environment, and a software life-cycle process model that guides a user in their invocation of SOFTMAN tools was developed using the Articulator. The SOFTMAN process model was then transformed and loaded into the PBI process interpreter. This interpreter drives the process-based user interface by structuring access to which SOFTMAN tools can be invoked on the designated objects. This in turn depends on the development process history and process enactment status at that time. In order to do this, the pre-existing top-level user interface for SOFTMAN had to be disabled and redirected to the new process interface, while the tool-specific interfaces and object repository for SOFTMAN remained intact and unmodified. With the new PBI interface, a user is presented with a visual representation of the development process currently assigned to them. Users can then select the next process task or action to perform. In this way, we can say that a *process integration* interpreter and user interface enable knowledge-based process models to guide user invocation of tools within a software production environment, such as PBI-SOFTMAN [59].

The Articulator has also been integrated with the Matisse team programming environment [32] and the SynerVision process execution engine [42] in place of PBI. The resulting composite environment is called SMART [31]. SMART supports the modeling, analysis, enactment, measurement, and improvement of software processes utilizing encapsulated CASE tools that communicate messages and tool invocations across the SoftBench BMS [13], and that manipulate data stored in networked repositories. Since SMART can configure commercially available CASE tools, and as most Unix-based CASE tools have SoftBench encapsulations, then the number and configuration of process-based CASE environments that can be generated using SMART is very large. Finally, the Articulator has similarly been integrated with the generic process engine developed at USC/ISI [5],

which supports the refinement of generic process descriptions into concrete instantiations during its enactment. Thus, the Articulator environment successfully demonstrates the power of software process meta-models as a basis for integrating and interoperating independently developed tools within generated process-driven environments.

Last, the SPADE environment [6] is similar in scope to the Articulator in that it supports the design, analysis, and enactment of software process models. Its process representation is based on extended Petri-nets, and the environment includes functional mechanisms for accommodating dynamic changes to a process description during its enactment. However, it does not utilize a process simulation facility, nor does it include inferential mechanisms for diagnosing and repairing process faults, as does the Articulator.

2.4 Process programming

Process programming environments use a programming language to describe the processes that form the capabilities of the environment. The goal of process programming-based systems is to provide a highly flexible, process-oriented environment construction mechanism. Process programming differs from process modeling in the form of the meta-model provided for environment construction. However, we can view process programming as process modeling where the meta-model is a programming language.

The Arcadia Research Project [87] consists of a collection of loosely coordinated research and development projects conducted by the Arcadia Consortium. Like ESF, Arcadia could also be categorized as an environment framework, but the architecture of the Arcadia framework is centered on the capability to execute process programs. In this manner, users interact with an Arcadia-based environment by executing parts of the process. The Arcadia environment is a process-centered environment. The form of the environment which the users see is wholly determined by the processes which are executed to form the environment.

Arcadia is currently a collection of many partially integrated components including a UIMS, an OMS, an Ada-based Process Programming Language (APPL/A), a Measurement and Evaluation (M&E) system, and a Process Administration System. The UIMS is based on Chiron [50], an Ada UIMS based on a very strong separation of functionality from interaction. The OMS in Arcadia is viewed as possibly many different object managers combined through a common underlying type model where data is combined through interoperability mechanisms. Triton [41] and PGraphite [93] are two OMSs which have been used as a basis of object management in the Arcadia project. APPL/A [86] is a process programming language which consists of extensions to Ada that include the notion of persistent relations associated with the OMS. The Process Administration System is a run-time support interface for executing processes based on a corporate metaphor. This metaphor has been suggested as a basis for the specification of environment characteristics.

Environment adaptation is the primary focus of the Arcadia architecture. Data and processes can be directly defined using the software interfaces provided by the Arcadia infrastructure. Further adaptation can be built up by creating processes which act like virtual architectures. In other words, a process could be defined which would allow the specification of user-role definitions. This mechanism would not be part of the Arcadia infrastructure, but it would be part of the virtual environment created by the definition of the process which allows user-role specification.

In contrast, the Adele-2 environment [61] utilizes a special-purpose persistent database and programming language, Adele, to describe and implement enactable processes. Adele-2 provides

support for multiple work environments (i.e., tool, policy, and method configurations) that are coordinated and coupled through an activity manager and a task manager. Coordination and coupling, as well as activity and task specification, are all realized using the Adele-2 language.

Process WEAVER [29] and SynerVision [42] are commercially available process execution engines. Each provides a process programming language based on the Unix shell command langugage. Their use is similar in scope to Adele-2, but they both lack support for a persistent object store, other than the underlying file system. However, the SMART environment [31] can semi-automatically generate SynerVision process program code via transformations on Articulator-based software process models [57], while Process WEAVER allows modeled process states to be attributed with Unix shell commands, which can subsequently be executed under user direction.

Last, both OIKOS [3] and OPM [85] propose object-oriented approaches to process programming. The idea in these two efforts is to show how object-oriented views of process programming and software development environments can be united into a single view in which both the process model and programming environment can be dynamically modified. Thus, object-oriented process programming seeks to provide an object-oriented process model (or meta-model), process programming language, and meta-environment framework where the environment data, control, and presentation models all share a uniform object-oriented representation and interpretation scheme.

2.5 Tool integration

A primary concern for environment creators is the integration of existing tools as part of the capabilities of an environment. Because tool integration provides the ability to invoke other tools, the technology described in this section is often applied in each of the previous meta-environment construction approaches. For example, a process model-based environment can be based on a special tool which is in charge of executing a process model and in turn invokes other tools. However, we have split tool integration into a separate category because a large body of efforts have a distinct model, transformation, and process from any of the previous categories. In particular, tool integration meta-environments provide a model based on an interface between a set of connected tools. The execution or use of the interface to the tools is the transformation of the model into an environment.

Basically two paths towards easier integration have been suggested. One path is the development of standards which tools must use in order to be integrated. Some standards are merely standards of data-exchange, while others are standards which affect all data storage. The other path is to build technology which can use tools developed prior to or outside of the standards. In the remainder of this section, we will consider examples of both approaches.

2.5.1 Standards

A Tool Integration Standard (ATIS) [9] and its more recent name, a Component Integration Standard (CIS), is a proposal for an object-oriented approach to the integration of tools that provides a set of interfaces that support schema-driven dispatching of behavior. ATIS/CIS is presented in terms of a hierarchy that specifies and interrelates abstract data types, their properties, and their methods. In other words, ATIS/CIS proposes an instance of an object-oriented model which defines a set of classes and their properties. This model is designed to provide services to tools which promote integration and make tool development easier.

The IRDS [39] is an ANSI Standard (ANSI X3.138). The IRDS standard defines a meta-model using an ERA model. Instances of the meta-model are models which describe data. IRDS allows access to schema definitions through access to instances of meta-entities. The meta-entities define the types of entities, relationships, and attributes.

The IRDS/ISO is the IRDS Rapporteur Group of the ISO/IEC JTC1/SC21 WG3 (International Standards Organization, International Electotechnical Commission Joint Technical Committee 1 / Subcommittee 21 Working Group 3). The IRDS/ISO uses a model similar to the relational model as an approach to representing data. Information is stored at two levels, the definition level and the IRD level. The definition level defines the tables and the functions which manipulate the tables. The IRD level contains the application data. This is analogous to class and instance levels of an object-oriented model.

CDIF [24] is a definition of a common data transfer format which supports many data models and the transfer of data between CASE tools or frameworks. The CDIF approach is based upon the concept of a Meta-Meta-Model. In order to describe the conceptual models for each of these domains, a model (i.e., the meta-meta-model) which can describe data models for different application domains was developed. The Meta-Meta-Model is similar in form to an Entity-Relation-Attribute (ERA) model. Various proposals for how to best implement CDIF-based data transformations, however, seem to focus on batched monolithic transformation of large data sets. In contrast, the DHT approach noted earlier also incorporates modeling formalism similar in power to a meta-meta-data model, but does not require transformation of data sets to support access to heterogeneous data repositories [63]. In addition, DHT implements a persistent hypertext-based object management service using a persistent programming language, while CDIF lacks such a service and programming language.

The P1175 reference model for interconnections between computing system tools [44] is a tool-integration standard being developed by the IEEE Computer Society's Task Force on Professional Computing Tools. The approach taken in P1175 is to define reference models for tool-to-organization and tool-to-platform interconnections, and a language for the transfer of data between tools. The tool to organization interconnection defines roles, life-cycle phases, and support elements which relate to a tool. The tool to platform interconnection describes various interconnections which must be supported by the platform and by the tools to effectively integrate the tools into the platform. These interconnections consist of the Data Base Manager, Communication Network Manager, O/S Service Manager, and the User Interface Manager interfaces.

The Object Management Group (OMG) [81] is an international organization of more than 50 information systems vendors, users, and research organizations with diverse backgrounds. The basic technical approach being followed in OMG is an object-oriented layer of services which exists above various implementations of object managers, user-interface facilities, and environment managers. This layer defines a set of network transparent protocols for message-oriented common service requests.

The Product Data Exchange using STEP (PDES) is a US organizational activity that supports the development and implementation of STEP (Standard for the Exchange of Product Model Data - ISO TC184/SC4). STEP is a neutral mechanism capable of completely representing product data throughout the life-cycle of a product. This representation is suitable for neutral file exchange and as the basis for sharing product databases and archiving.

The Engineering Information System (EIS) [54] is a vast undertaking addressing heterogeneity of hardware and software platforms, data formats, tools, site-specific policies and methodologies,

and interfaces primarily oriented around the computer-aided engineering (CAE) domain. Within the CAE community, the EIS program is intended to produce a consolidated approach to a broad set of functional and other requirements. This approach consists of proposed standards and guidelines for services which enable and accelerate a trend toward uniform engineering environments and information exchange. The EIS framework contains automated services which support and control the data and activities of the engineering process. The framework is composed of an OMS, Application Object Model (AOM), and Engineering Environment Services (EES). The OMS is composed of a meta-model, schema management, execution control, object identification support, and data access facilities. The AOM uses the OMS to create higher-level constructs which provide object-oriented capabilities. The EES is a large collection of types, operations, and default policies that support the engineering and administration processes built on top of the AOM. The EES addresses configuration management, access control, audit trails, backup and archival, inter-EIS exchange, user environment services, e.g., login, and rule processing (data-driven triggers). The UIMS is a family of guidelines and candidate standards for an interface to a CAE system. The UIMS design not only concentrates on interfaces for interactive EIS applications and tool adapters, but also provides interfaces for tying in external tools.

Any standardization approach to tool integration will work only if a significant number of tools use the standard. At this point, most of these standards are unaccepted, either by the national or international standards organizations, e.g., ISO, NIST, or by software tool vendors. One of the problems facing standardization efforts is the vast number which apply to tool integration. We have only presented a small subset to give some flavor of this approach. The IEEE P1175 effort cites around two hundred standards efforts, many of which overlap or compete with one another.

Another problem facing standardization efforts is the large number of existing tools which cannot be brought up to par with most standards efforts. While some standards do address this problem by considering approaches to data transfer which do not require changing storage formats of existing tools, many of the standards fall short in this area. The following section considers various technologies which address issues similar to the standards efforts, but do not assume prior knowledge of their technology.

2.5.2 Integration technologies

Tool integration environments began with the Unix operating system. The Unix pipe facility allows the combination of tools into larger tools through the interconnection of tool outputs to tool inputs through standard typeless I/O facilities. However useful this simple piping mechanism may be, it falls short of the kind of tool integration that other efforts have demonstrated.

DRACO [62] is an approach to the construction of software by organizing reusable software components or tools according to a specific problem area or domain. DRACO uses a domain language for describing programs in each domain. Statements of programs in these domain languages are then optimized by source-to-source program transformations and refined into other domains. A single reusable component corresponds directly to each object and operation in the domain language. Given the domain of software production environment construction, DRACO effectively provides a tool composition language based on a module interconnection language paradigm. GENESIS (from UT Austin) [8] is similar in purpose to DRACO, but specialized to a domain for constructing and generating of special-purpose database management systems.

Toolpack [67], and later Odin [17], support the construction of an environment based on the

specification of software objects, tools, and relationships between objects and tools. Environments integrated through the use of Odin can be thought of as collections of tools which are satellites around a large structured repository of software data. Both Toolpack and Odin are ancestral predecessors to the Arcadia environment described earlier.

The Hewlett-Packard SoftBench product [13] is a tool integration framework comprised of several components including the Broadcast Message Server (BMS) that functions as a software bus, a Motif-based user-interface, the Encapsulator [20], and some integrated CASE tools. In the HP integration model, each tool makes changes to global information and informs other tools about its actions via the BMS. The BMS uses a broadcast paradigm via the software bus which is different from the point-to-point paradigm offered by object-oriented systems. The broadcast nature of the BMS communication allows the set of tools managed by a BMS and interested in a particular message to be extended without requiring any change in the tools that send the messages. The Encapsulator [20] provides a means of integrating tools into the HP SoftBench user-interface and BMS. The encapsulation consists of the Encapsulation Description Language (EDL) which describes a user-interface and corresponding communication across the BMS. The communication information consists of messages which it will respond to and messages it will generate in response to user-interface events. The message model [45] which is suggested as a basis of the environment definition is very primitive and will be too limited for environments consisting of a very large number of tools. A "multi-cast" messaging paradigm which transmits strongly typed messages may provide an alternative. Accordingly, a combination of SoftBench and SUN's ToolTalk tool integration mechanisms have been proposed as the basis of a new "standard" for open tool integration, as part of the Common Development Environment (CDE) now being investigated by the Open Software Foundation (OSF).

Field [74, 75] provides an integration framework very similar to the BMS using a communication mechanism called selective broadcasting. Field has extended the HP Softbench approach by passing all data through the message server rather than relying on a common database. Additionally, Field supplies an editor which provides consistent access to source code in multiple contexts and a set of analysis tools.

Forest [35] has extended the Field communication mechanism with an additional decision mechanism based on policies. Policies are rules that determine how and when tools are invoked. This approach represents a hybrid between process modeling mechanisms and frameworks.

The Common Lisp Framework (CLF) [4] is an incremental integration, development, and evolution environment. Tools and applications are programs written in an extension of Common Lisp called AP5. AP5 provides a persistent virtual database of relations, objects, and rules. The rules connect tools through triggering upon detection of changes to data or calling procedures.

The Scorpion Meta-Environment [80] uses a specification of tool topology using a module interconnection language and a specification of the level of trade-off between evolution support and efficiency. This approach is different from the previous approaches in that it primarily addresses the use of a tool integration technology based on the module interconnection language rather than addressing the integration technology itself.

Last, the Matisse team programming environment [32] builds upon concepts previously demonstrated in the CLF and Odin. Matisse provides a multiuser programming support environment where user functionality and data object representation are managed by an intepreted, rule-based, persistent programming language. In this manner, shared objects or software components can be cached into a user's local address space from a networked object repository, to realize a user-

configurable or user-extended workspace.

Overall, tool integration construction methods address the important problems of integrating externally developed tools into an environment. However, because tool integration construction is usually based on a limited control policy, they represent only a partial solution to the problem of environment construction. Generally, tool integration methods must be used in conjunction with other environment construction methods in order to support the construction of a full environment. Thus, it should be no surprise to find that many of these tool integration mechanisms are being extended or combined with environment frameworks or process support technologies.

2.6 Summary of meta-environment research

In Table 1, we briefly summarize the categories of meta-environments in terms of their models and transformations. For some of the categories, we have shown subcategories which arise based on differing approaches. Given these various approaches, we will now consider the more general question of what requirements must a meta-environment address.

3 The Meta-Environment Problem

In the previous section, we presented a variety of approaches to lowering the cost and improving the efficiency of environment construction. While we have found that the approaches taken are not similar, the problems they address are. In particular, the common thread among these efforts is that all attempt to address problems associated with environment construction. Thus, to present this trend, we will attempt to give further insight into the problem area and, in particular, to the functional requirements which form the common thread.

Considerable work has been done in the area of defining the requirements for software production environments [66, 82, 83]. That is, a meta-environment must be capable of producing environments which satisfy these requirements. In this section, we will first present the kinds of roles environment builders must play and the processes they must perform which are the basis for meta-environment technology. These roles, processes, and related research are used as the basis of the requirements presented in the next section.

3.1 Meta-environment roles

One of the best ways to understand the problems which are to be addressed by meta-environment technology is to look at the various roles played by environment builders. A complete meta-environment process must account for the roles of the environment integrator, component builder, and component model manager among others.

The *environment integrator* is responsible for creating an environment specification and controlling the associated construction of an environment. This role has been the focus of this paper so far. In particular, developers acting in this role utilize meta-environment technology for environment construction, build a specification of the environment in terms of the meta-environment's construction model, and then generate and refine the specified environment. In general, the generation part of the process is an automated, or at least semi-automated, task. However, the task of specifying the environment's characteristics in terms of the construction model is a demand often requiring

Category	*Model*	*Transformation*	*Examples*
Frameworks	ERA data model, Process-based Control model	Used directly	PCTE, CAIS, SLCSE, ALMA, DHT, ESF K/2r
	Object-oriented data and control model	Used directly	Atherton, Gaia, RDPE3
Customizable	Grammar-based	Language-oriented environment generated	Mentor, GANDALF, Synthesizer Generator TRIAD, GEM, EDGE, TGE, VSF, GEDL
	Life-cycle objects	Generates environment which is specific to the information domain	Softman, ISHYS, META/GA, MetaView
Process modeling	Process meta-model	Executed directly	ESF, Genesis, Marvel, Merlin, PRISM, IPSE 2.5, PSS, MELMAC, SDA, Articulator, PBI-Softman, SPADE, SMART
Process programming	Process-oriented programming language	Compiled to generate environment	Arcadia, Adele-2, OIKOS, OPM, Process WEAVER, SynerVision
Tool integration	Standards and tool interface	Tools used directly according to tool interface	ATIS, IRDS, IRDS/ISO, CDIF, P1175, OMG, PDES, EIS
	Tool integration technology and tool interface	Tools used directly according to tool interface	Unix, DRACO, Toolpack, Odin, HP SoftBench, Field, Forest, CLF, Scorpion, PBI, Matisse

Table 1: Summary of meta-environment research

the environment integrator to be a software process architect, tool integrator, data modeler, control modeler, etc.

The *component builder* is responsible for creating, providing, or automatically generating new components which will be used in SPEs. This role has not been considered to this point. In general, the environment integrator does not create all of the environment components from scratch. Instead, the environment integrator relies on existing frameworks, process execution mechanisms, tool integration technology, generated or acquired tools, and schemata. The component builder is responsible for creating this technology, as well as the other software components which may be incorporated into them. This may entail selecting from existing components available within some reusable component repository, or extracting (and restructuring if necessary) candidate components from existing software systems [15, 59]. Note that there can be many component builders building products which are completely incompatible. The fact that they assume the same role does not imply that they work together. Nonetheless, once the component builder has created or acquired a new component, that component is potentially available for use in an environment or meta-environment.

The *component model manager* is responsible for the creation and evolution of models of component characteristics. Presently, the models are very simple and the description of characteristics is in natural language. However, as meta-environment technology becomes more prevalent and sophisticated, there will be a need for more sophisticated models to allow for automated and semi-automated selection of environment components as part of the environment integrator's role. An emerging line of research has begun to explore and develop such models using module/component interconnection formalisms and interaction protocols. Thus, the component model manager can be seen as a mediator between the component builder and the environment integrator, where the model is the language of discourse between these two roles.

Given these roles, we will now present a set of requirements which meta-environments must attempt to satisfy.

3.2 Meta-environment requirements

Environment specification: The fundamental requirement that a meta-environment must satisfy is that it should be able to support the construction of software production environments in accord with their specifications. This implies that a constructed environment must support desired software production methods and the process which controls the application of these methods. Furthermore, a constructed environment must have the characteristics of a "good" environment: it must be fast, provide a high level of functionality in a consistent and coherent manner, provide a consistent "look and feel" graphical user interface, etc. [66, 82, 83]. A meta-environment should provide a construction model able to express policies or methods for dealing with security, integrity, reuse, process, etc. Furthermore, the construction model should be general enough to express any of the mechanisms which can be chosen within any of these categories. This includes the ability to specify data models, data repositories, tool bindings and control message invocations, graphic and textual presentation displays, and multiple process model descriptions or notations, while accommodating one or more operating systems, computing hardware platforms and network communication protocols. Finally, either an (i) empirically tested, (ii) experience-based, or (iii) analytically robust process (or meta-model) should define the procedures for creating an environment.

Ease of use: The construction method supported by a meta-environment should be easy to use in

the following ways:

- *Multiple levels of support.* The users of a meta-environment are likely to have different levels of expertise in dealing with the meta-environment. For this reason, a meta-environment should support multiple levels of interaction which give greater power to expert environment constructors, e.g., via process programming, and easier interaction for nonexpert environment constructors, e.g., by specifying parameter values in a process model.
- *Understandable.* A meta-environment should provide support in understanding the construction model through visualization, query, and browsing of process, control, and data model representations that will be embodied in the resulting environment. In addition, the process by which a specific environment is constructed should be able to be recorded, analyzed, replayed, simulated, reconfigured, and reused.
- *Selection assistance.* The number of possible software production methods and environment specifications which provide support for these methods is likely to be very large. A meta-environment should assist users in the selection of desired software production methods and corresponding environment specifications. Meta-environments might include reusable or extensible "starter kits" [40], or complete working examples of the kinds of environments that can be readily produced.
- *Automatic error checking.* A meta-environment should support automatic error checking in order to prevent the creation of environments with significant bugs in compilation, execution, or execution semantics. Language-directed text or graphic editors that can detect and prevent the entry of syntactically or semantically incorrect descriptions are typical tools to support this. In addition, mechanisms for analyzing, simulating, replaying, or repairing environment specifications or process models will be helpful.

Evolution support. Alterations are likely to occur in response to evolution in both the needs of a project and available technology. Both the environments created by a meta-environment and the meta-environments themselves must be capable of supporting evolution. A meta-environment should support evolution in the following ways:

- *Data continuity.* Changes to environments are likely to be required during the lifetime of the project which an environment supports. Therefore, a meta-environment should prevent the loss of project data during the evolution of environment specifications and the corresponding software production environment.
- *Incremental specification.* It is likely that only part of the complete environment specification will be available or known at the start of a project. A meta-environment should support the incremental construction of an environment based on partial specifications.
- *Environment versions.* As the environment evolves, it is important to control its evolution in a systematic manner. Therefore, a meta-environment should support version and configuration control over environment specifications.
- *Open for new technology.* When new technology such as better tools, frameworks, user interfaces, or methods becomes available, the meta-environment and the environments constructed by the meta-environment should be able to take advantage of this new technology. This requirement affects both the form of the environments and the meta-environment itself. The

environment should be open such that new technology can be included. A meta-environment should be able to make the new technology available to environment specifications, and thus integrate and interoperate heterogeneous software data objects, components, tools, repositories, and process modeling notations.

Adopt Existing Capabilities. One of the primary means for making significant gains in environment construction capabilities is to adopt existing technology, such as currently available tools, frameworks, and user interfaces. A meta-environment should support the use of existing technology, where appropriate, in order to leverage the power of the construction method and correspondingly, the constructed environments. A meta-environment should also support, where possible, the use of environment capability generators or tool-building tools, such as editor generators or other application generators. Environment capability generators do not have the same characteristics as the capabilities themselves, but meta-environments should provide an effective means of utilizing this power.

4 Conclusions

Research on meta-environments for software production is following a number of alternative paths, representing a focus on either environment frameworks, customizable environments, process modeling, process programming, or tool integration. These paths represent both competing and complementary alternatives to the challenging problem of how to rapidly produce standardized or customized environments for engineering software applications. Clearly, no one path, nor any single meta-environment architecture, represents the best choice in all circumstances.

While we have examined dozens of efforts aimed at developing meta-environments, it should be clear that most of these efforts combine techniques and mechanisms employed in other categories. Further, we may expect to see a trend toward new or increased combination of techniques and mechanisms across these efforts. Accordingly, we think that further study in this field and further development of meta-environments is an important stepping stone in the creation of effective software production environments.

Finally, we have attempted to summarize and synthesize an emerging set of requirements that should be satisfied or addressed by meta-environments in the time ahead. These requirements outline a bold agenda for research and development in the area of meta-environments. Meta-environments have emerged as a key strategy for reducing the cost, time, and effort of constructing software production environments. If these requirements can be met, researchers will be able to use meta-environments to learn a great deal more about the requirements for environments themselves. Furthermore, meta-environments will enable the production of a new generation of large-scale software applications that are engineered using domain-specific environments constructed from meta-environments. Thus, the ultimate payoff from meta-environments will lie in the domain-specific environments and applications that can most readily be produced and supported.

Acknowledgements: Preparation of this report was supported in part through contracts and grants to the USC System Factory Project from ATT Bell Laboratories, Hewlett-Packard, Northrop Corporation, the USC Center for Operations Management, Education and Research (COMER), and others. No endorsement implied.

References

[1] E. W. Adams, M. Honda, and T.C. Miller. Object management in a CASE environment. In *11th Int. Conf. Software Engineering*, pages 154–162, May 1989.

[2] R. Adomeit, W. Dieters, B. Holtkamp, F. Schulke, and H. Weber. K/2r: a kernel for the Eureka Software Factory support environment. In *Proc. 2nd. Int. Conf. Systems Integration*, pages 325–336. IEEE Computer Society Press, June 1992.

[3] V. Ambriola, P. Ciancarini, and C. Montangero. Software process enactment in OIKOS. In *Proc. Fourth ACM SIGSOFT Symp. Software Development Environments*, pages 183–192, December 1990.

[4] R. Balzer. A 15 year perspective on automatic programming. *IEEE Transaction on Software Engineering*, 11(11):1257–1267, November 1985.

[5] R. Balzer and K. Narayanaswamy. Mechanisms for generic process support. In *Proc. First ACM SIGSOFT Symp. Foundations Software Engineering*, pages 21–32. ACM Software Engineering Notes, Vol. 18(5), December 1993.

[6] S.C. Bandinelli, A. Fuggetta, and C. Ghezzi. Software process model evolution in the SPADE environment. *IEEE Trans. Software Engineering*, 19(12):1128–1145, 1993.

[7] V.R. Basili and H.D. Rombach. The TAME project: Towards improvement-oriented software environments. *IEEE Trans. Software Engineering*, 14(6):758–773, June 1988.

[8] D.S. Batory, J.R. Bennett, et al. GENESIS: An extensible database management system. *IEEE Trans. Software Engineering*, 14(11):1711–1731, 1988.

[9] H.R. Beyer, K. Chapman, and C. Nolan. The ATIS Reference Model. draft, June 1990.

[10] B.W. Boehm. Software engineering environments in the United States (Plenary Talk). In *Fourth ACM SIGSOFT Symp. on Software Development Environments*, December 1990.

[11] G. Boloix, P.G. Sorenson, and J.P. Tremblay. On transformations using a metasystem approach to software development. *Software Engineering J.*, 7:425–437, 1992.

[12] R. F. Bruynooghe, J. Parker, and J.S. Rowles. PSS: A system for process enactment. In *Proc. First Int. Conf. Software Process*, pages 128–141, 1991.

[13] M. Cagan. The HP SoftBench environment: an architecture for a new generation of software tools. *Hewlett-Packard J.*, pages 36–47, June 1990.

[14] S.C. Choi and W. Scacchi. Assuring the correctness of configured software descriptions. *Proc. 2nd. Int. Work. Software Configuration Management, ACM Software Engineering Notes*, 17(7):67–76, 1989.

[15] S.C. Choi and W. Scacchi. Extracting and restructuring the design of large systems. *IEEE Software*, 7(1):66–73, January 1990.

[16] S.C. Choi and W. Scacchi. SOFTMAN: An environment for forward and reverse computer-aided software engineering. *Information and Software Technology*, 33(9):664–674, November 1991.

[17] G. Clemm and L. Osterweil. A mechanism for environment integration. *ACM Transactions on Programming Languages and Systems*, 12(1):1–25, January 1990.

[18] Common ADA Programming Support Environment (APSE) Interface Set. *Introduction to CAIS*, September 1989. MIL-STD-1938A.

[19] J. Cramer, H. Hunnekens, W. Schafer, and S. Wolf. A process-oriented approach to the reuse of software components. Technical Report 43, Dortmund University, March 1990.

[20] H. Davidson. Encapsulator: The plug-in Compatibility tool for SoftBench. SoftBench Technical Note Series SESD-89-11 Revision 1.1, Hewlett-Packard, Software Engineering Systems Division, 3404 E. Harmony Road, Fort Collins, Colorado 80525, June 1989.

[21] W. Dieters and V. Gruhn. Managing software processes in the environment MELMAC. In *Proc. Fourth ACM SIGSOFT Symp. Software Development Environments*, pages 193–205, December 1990.

[22] V. Donzeau-Gouge et al. *Programming Environments Based on Structured Editors: the Mentor Experience*, pages 128–140. McGraw Hill Book Co., New York, 1984.

[23] A. Earl. A Reference Model for Computer Assisted Software Engineering Environment Frameworks. Hewlett-Packard Laboratories, Fort Collins, CO. USA, May 1990.

[24] EIA. *CDIF Organization and Procedure Manual*, cdif-doc-n2-v1 edition, January 1990. Electronics Industry Association Project No.: EIA/PN-2329, USA.

[25] R.J. Ellison. Software development environments: research to practice. In *Int. Work. Advanced Programming Environments*. Springer-Verlag, June 1986.

[26] W. Emmerich and V. Gruhn. Software process modeling with FUNSOFT nets. Technical Report Technical Report 47, University of Dortmund, 1990.

[27] W. Emmerich, G. Junkerman, et al. Merlin: knowledge-based process modeling. In *Proc. First European Work. Software Process Modeling*, pages 181–186, Milan, Italy, 1991.

[28] ESF - Eureka Software Factory. *ESF Technical Reference Guide*, 1989.

[29] C. Fernstrom. Process WEAVER: adding process support to Unix. In *Proc. 2nd. Int. Conf. Software Process*, pages 12–26. IEEE Computer Society Press, 1993.

[30] C. Fernstrom and L. Ohlsson. Integration needs in process enacted environments. In *Proc. First Int. Conf. Software Process*, pages 142–158, 1991.

[31] P.K. Garg, P. Mi, T. Pham, W. Scacchi, and G. Thunquest. The SMART approach to software process engineering. In *16th. Int. Conf. Software Engineering*, pages 341–350. IEEE Computer Society Press, May 1994.

[32] P.K. Garg, T. Pham, et al. Matisse: a knowledge-based team programming environment. *Int. J. Software Engineering and Knowledge Engineering*, to appear, 1994.

[33] P.K. Garg and W. Scacchi. ISHYS: Designing an intelligent software hypertext system. *IEEE Expert*, 4(3):52–63, Fall 1989.

[34] P.K. Garg and W. Scacchi. A hypertext system to manage software life-cycle documents. *IEEE Software*, 7(3):90–99, May 1990.

[35] D. Garlan and E. Ilias. Low-cost, adaptable tool integration policies for integrated environments. In *Proc. Fourth ACM SIGSOFT Symp. Software Development Environments*, pages 1–10, December 1990.

[36] J.L. Giavitto, A. Devarenne, G. Rosuel, and Y. Holvoet. Adage: New trends in CASE environments. In *Proc. Int. Conf. Systems Development Environments and Factories*, May 1989.

[37] A. Goldberg and D. Robson. *Smalltalk-80: The Language and Its Implementation*. Addison-Wesley, 1983.

[38] A. V. Goldberg and K. J. Lieberherr. GEM: a generator of environments for metaprogramming. In *COMPSAC 85*, pages 86–95, 1985.

[39] A. Goldfine and P. Konig. A technical overview of the information resource dictionary system. Technical Report NBSIR 88-3700 (Supersedes NBSIR 85-3164), U.S. Department of Commerce, National Bureau of Standards, Institute for Computer Sciences and Technology, Gaithersburg, MD 20899, January 1988.

[40] M.L. Griss. Software reuse – from library to factory. *IBM Systems Journal*, 34(4):548–566, 1993.

[41] D. Heimbigner. Triton reference manual. Technical Report CU-CS-483-90, Department of Computer Science, University of Colorado, July 1990.

[42] Hewlett-Packard. *Developing SynerVision Processes*. HP Palo ALto, CA, part number: b3261-90003 edition, May 1993.

[43] M. Heym and H. Osterle. Computer-aided methodology engineering. *Information and Software Technology*, 35(6/7):345–354, 1993.

[44] IEEE Computer Society's Task Force on Professional Computing Tools. *A Standard Reference Model for Computing System Tool Interconnections*, p1175/d6 edition, February 1990. Draft.

[45] B.T. Jenings. The HP SoftBench Message Model: Concepts and conventions used by the HP SoftBench Tools. SoftBench Technical Note Series SESD-89-21 Revision 1.2, Hewlett-Packard, Software Engineering Systems Division, Fort Collins, C), USA, September 1989.

[46] G.E. Kaiser and P. Feiler. An architecture for intelligent assistance in software development. In *Proceedings of the 9th International Conference on Software Engineering*, pages 180–188, February 1987.

[47] A.S. Karrer. *Generating Graph Editors*. PhD thesis, Computer Science Department, University of Southern California, May 1993.

[48] A.S. Karrer and M. Penedo. A survey of native environment framework architectures. TRW Corporation Note Arcadia-TRW-90-002, El Segundo, CA, August 1990.

[49] A.S. Karrer and W. Scacchi. Requirements for an extensible object-oriented tree/graph editor. In *ACM SIGGRAPH Symposium on User-Interface Software and Technology*, pages 84–92, October 1990.

[50] R.K. Keller, M. Cameron, R.N. Taylor, and D.B. Troup. Chiron-1: a user interface development system tailored to software environments. Technical Report UCI-90-06, Department of Information and Computer Science, University of California, Irvine, June 1990.

[51] J.D. Kiper. *The ergonomic, efficient, and economic integration of existing tools into a software engineering environment*. PhD thesis, Ohio State University, USA, 1985.

[52] K. Kishida, T. Katayama, M. Matsuo, I. Miyamoto, K. Ochimizu, M. Saito, J. Saylet, K. Torii, and L. Williams. SDA: a novel approach to software environment design and construction. In *Int. Conf. Software Engineering*, pages 69–79, April 1988.

[53] P. Klint. A meta-environment for generating programming environments. *ACM Trans. Software Engineering and Methodology*, 2(2):176–201, 1993.

[54] J.W. Krueger. *Application Object Model for Engineering Information Systems*. Honeywell Systems and Research Center, Minneapolis, MN, October 1989.

[55] A. Van Lamsweerde et al. Generic lifecycle support in the ALMA environment. *IEEE Trans. Software Engineering*, 14(6):720–740, June 1988.

[56] N.H. Madhavji et al. Prism = methodology + process-oriented environment. In *Proc. 12th Int. Conf. Software Engineering*, pages 277–289, 1990.

[57] P. Mi and W. Scacchi. A knowledge-based environment for modeling and simulating software engineering processes. *IEEE Trans Knowledge and Data Engineering*, 2(3):283–294, March 1990.

[58] P. Mi and W. Scacchi. Modeling articulation work in software engineering processes. In *Proc. First Int. Conf. Software Process*, pages 188–201, 1991.

[59] P. Mi and W. Scacchi. Process integration for CASE environments. *IEEE Software*, 9(2):45–53, March 1992.

[60] R. Munck, P. Oberndorf, E. Ploedereder, and R. Thall. *An Overview of DOD-STD-1838A (proposed), The Common APSE Interface Set, Revision A*. Technical report, Dept. of Defense, USA, 1988.

[61] W.L. Melo N. Belkhatir, J. Estublier and L.G.I. France. Adele-2: A support to large software development processes. In *Proc. First Int. Conf. Software Process*, pages 159–171, 1991.

[62] J.M. Neighbors. The DRACO approach to constructing software from reusable components. *IEEE Trans. Software Engineering*, 10(5):564–573, September 1984.

[63] J. Noll and W. Scacchi. Integrating diverse information repositories: the Distributed Hypertext approach. *Computer*, 24(12):38–45, December 1991.

[64] D. Notkin. The GANDALF project. *J. Systems and Software*, 5(5):91–105, May 1985.

[65] H. Ossher and W. Harrison. Support for change in RDPE3. In *Proc. Fourth ACM SIGSOFT Symp. Software Development Environments*, pages 218–228, December 1990.

[66] L. Osterweil. Software environment research: directions for the next five years. *IEEE Computer*, 14(4):35–43, April 1981.

[67] L. Osterweil. TOOLPACK - an experimental software development environment research project. *IEEE Trans. Software Engineering*, 9(6):673–685, November 1983.

[68] W. Paseman. The Atherton Software BackPlane – an architecture for tool integration. *Unix Review*, April 1989.

[69] F. Newbury Paulisch and W. Tichy. EDGE: an extendible graph editor. *Software Practice and Experience*, 20(S1), June 1991.

[70] M. Penedo and W.E. Riddle. Guest editors' introduction to software engineering environment architectures. *IEEE Trans. Software Engineering*, 14(6), June 1988.

[71] E. Ploedereder, T.C. Harrison, P. Oberndorf, C. Roby, F. Belz, J.F. Kramer, and J. Clouse. *Rationale for DOD-STD-1838 (CAIS)*. Technical report, Dept. of Defense, USA, August 1988.

[72] J.N. Popcock. VSF and its relationship to open systems and standard repositories. In *Software Development Environments and CASE Environments*, pages 53–68. Springer Verlag, Lecture Notes in Computer Science, Vol. 509, 1991.

[73] C.V. Ramamoorthy, Y. Usuda, et al. GENESIS: an integrated environment for supporting development and evolution of software. In *COMPSAC '85*, pages 472–479, October 1985.

[74] S.P. Reiss. Connecting tools using message passing in the field environment. *IEEE Software*, 7(4):57–66, July 1990.

[75] S.P. Reiss. Interacting with the field environment. *Software Practice and Experience*, 20(S1):89–115, June 1990.

[76] T. Reps and T. Teitelbaum. The Synthesizer Generator. In *Proc. ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, April 1984.

[77] W. Scacchi. The Software infrastructure for a Distributed Software Factory. *Software Engineering Journal*, 6(5):355–369, September 1991.

[78] D. Smith. KIDS: A semi-automatic program development system. *IEEE Trans. Software Engineering*, 16(9):1024–1044, 1990.

[79] K. Smolander, P. Marttiin, K. Lyytinen, and V-P. Tahvanainen. MetaEdit – a flexible graphical environment for methodology modelling. In *Advanced Information Systems Engineering*, pages 168–193. Springer Verlag, Lecture Notes in Computer Science, Vol. 498, 1991.

[80] R. Snodgrass and K. Shannon. Fine grained data management to achieve evolution resilience in a software development environment. In *Proc. Fourth ACM SIGSOFT Symp. Software Development Environments*, pages 144–156, December 1990.

[81] R.M. Soley. Object Management Group Standards Manual. Draft 0.1 OMG TC Document 90.5.4, May 1990.

[82] P.G. Sorenson, J.P. Tremblay, and A.J. McAllister. The Metaview system for many specification environments. *IEEE Software*, 5:30–38, March 1988.

[83] Vic Stennig. On the Role of an Environment. *Communications of the ACM*, 1987.

[84] Tom Strelich. The Software Life Cycle Support Environment (SLCSE): a computer based framework for developing software systems. In *Proc. ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, pages 35–44, November 1988.

[85] Y. Sugiyama and E. Horowitz. Building your own software development environment. *Software Engineering Journal*, 6(5):317–331, September 1991.

[86] S.M. Sutton, D. Heimbigner, and L. Osterweil. Programmable relations for managing change during software development. Technical Report CU-CS-418-88, University of Colorado, Boulder, CO 80309-0430, September 1988.

[87] R. Taylor et al. Foundations for the Arcadia environment architecture. In *Proc 3rd ACM Symp. Software Development Environments*, November 1988.

[88] W. Teitelman and L. Masinter. The Interlisp programming environment. *IEEE Computer*, 14(4):25–33, April 1981.

[89] I. Thomas. PCTE interfaces: supporting tools in software engineering environments. *IEEE Software*, 6(11):15–23, November 1989.

[90] D. Vines and T. King. Gaia: an object-oriented framework for an Ada environment. In *Third Int. IEEE Conf. Ada Applications and Environments*, pages 81–90, May 1988.

[91] B. Warboys. The IPSE 2.5 project : process modeling as the basis for a support environment. University of Manchester, September 1989.

[92] D. Wile. Program development: formal explanations of implementations. *Communications ACM*, 26(11):902–911, November 1983.

[93] A. Wolf, J. Wileden, C. Fisher, and P. Tarr. Pgraphite: an experiment in persistent typed object management. In *Proc. ACM SIGSOFT/SIGPLAN Software Engineering Symp. Practical Software Development Environments*, pages 130–142, Novemeber 1988.

[94] W. Wong and M.V. Zelkowitz. A preliminary description of an Integrated Software Engineering Environment (ISEE) reference Model. National Institute of Standards and Technology, February 1990.

[95] Y. Yamamoto. *An approach to the generation of software life-cycle support systems.* PhD thesis, University of Michigan, 1981.