A Survey of Database Support for Process-Centered Software Development Environments

Dimitrios Tombros, Andreas Geppert
Institut für Informatik, Universität Zürich
{tombros|geppert}@ifi.unizh.ch

Technical Report 95.28

Abstract. Database management systems (DBMSs) are a basic component of integrated project support environments (IPSEs). They are used for the modeling, storage and management of data produced by various software tools during the development process. In environments where the development process is explicitly supported (process-centered software development environments - PCDEs), processes can also be modelled, persistently stored and managed. In the present paper, we survey how DBMSs are integrated in the architecture of PCDEs and in particular how DBMS mechanisms are used to provide parts of their functionality. The systems we consider are Adele/Tempo, ALF, Arcadia, Epos, Marvel/Provence, Merlin, Oikos, SPADE, and several commercial systems. A number of open issues with respect to database support in these PCDEs are identified and discussed.

Contents

1 Introduction

The development of software systems has proven to be a complex task which involves the coordination of various agents - human and computational - performing more or less related activities and operating on a large variety of software artifacts among which complex dependencies exist. *Software engineering environments* (SEEs) have been developed to support various aspects of the software development process. Their architecture can be broadly characterized as following one of two trends: they are either constructed by the integration of various separately developed tools, or they are environments conceived from scratch as a coherent whole. The first category refers to *CASE coalition environments* and the second to *integrated project support environments* (IPSEs) [25]. Their main difference lies in the approach used as a basis for integration. CASE-coalitions consist of heterogeneous tools using private repositories and are integrated by providing transparent data manipulation mechanisms independent of the location of data, i.e. functions are migrated to where the data resides. IPSEs on the other hand are based on data, presentation and control integration services on top of a common data repository [47]. This may be in the simplest case a common file structure, or more usually an object management system (OMS) or a DBMS[1], in which development data is persistently stored and can be shared between system components in a controlled way. In this paper, our focus of interest lies with a special category of IPSEs which have been the subject of active research in recent years. In these so-called *process-centered software development environments* (PCDEs), the definition and management of software processes plays a central role.

A software development project has to proceed according to predefined guidelines and possibly obey specific constraints. In PCDEs, *process models* (PMs) are used to formally define process guidelines and constraints, so that the processes can be executed, analyzed and validated. PM enactment refers to the execution of a process instance according to the process definition. As the PM is enacted, coordination of the developers involved and automation of certain tasks such as the invocation of external tools may take place.

In PCDEs, OMSs or DBMSs can be used to store both data produced during the enactment of the process models (i.e. production data or artifacts) and data about the process itself (i.e. the process representation and state). Furthermore, when using a DBMS, the additional functionality it provides compared to an OMS can be put to use to implement tasks for which additional PCDE components would have to be realized (e.g. concurrency control). Thus, it is not surprising that in various projects the use of DBMSs is being researched, substituting custom made OMSs or file systems used in earlier SEEs.

In section 2 we define PCDEs, DBMS and OMS, describe the requirements for using DBMSs in PCDEs, and briefly review existing concepts and solutions. In section 3 we survey example systems focusing primarily on the support provided by the DBMS/OMS for the representation of PMs and the integration of the DBMS/OMS within their architecture. We consider the following research prototypes: Adele/Tempo, ALF, Arcadia, Epos, Marvel/Provence, Merlin, Oikos, and SPADE. We also briefly describe a number of commercially available systems. Finally, in section 4 we conclude this paper with some general observations about the current situation and a discussion of remaining open issues.

---

1. The distinction between OMS and DBMS is somewhat blurred; the two concepts are defined in section 2 of this paper.

2 Definitions and Requirements

2.1 Software Processes and Process Models

Various proposals have been made to define the terminology used in software process engineering [e.g., 53, 80]. We are not going to attempt to construct our own definitions but will use the existing ones to provide a framework for understanding the terminology used throughout the paper. In any case, there seems to be wide consensus that software development processes can be viewed as a kind of software and as such are based on a set of definable requirements, and posses an architecture and design. Also, the development of software processes follows a life cycle currently heavily emphasizing on prototyping and early testing, a fact that is due to the lack of a complete theoretical foundation and to the highly complex and dynamic nature of the processes themselves.

A *software process* is a set of partially ordered *process steps* intended to reach a goal. The process steps can either be composite sub-processes or elementary *activities*. An activity is thus the elementary structuring unit of a software process. A process step can have resources (human and computerized) allocated to it which are required for the step to be performed. During the process products are created or modified, which are called *artifacts*.

A further concept needed for our survey is that of a *process model* (PM). According to [80], this is a more or less formal abstract description of a software process. The level of formality of this description has to be specified leading to a taxonomy of PMs, which can be classified as either prescriptive or descriptive and also as oriented towards expression, guidance, analysis or enforcement of a particular process. The level of abstraction of a PM determines the details of definition and implementation that are described as part of the PM. In any case, various aspects of the software process have to be modelled including the data types being manipulated, the activity types and structure, the resources used, and the organization within which the process is performed. In the process modeling literature, these aspects are referred to as sub-models of the PM. In order to establish a connection between different terminologies we note here that the term (sub-)model corresponds to the term *(sub-)schema* used in the database community, while the notion of a process meta-model, which is a conceptual framework for expressing and composing PMs, corresponds to the notion of a database data model which defines the elements available to the database schema designer. We will thus use each term depending on the context it appears.

A final set of concepts needed are those related to the execution of software processes. In the software process engineering literature, a distinction is often made between generic and customized PMs. Customized PMs are more detailed and contain domain-specific and project-specific characteristics. Such PMs are *instantiated*, i.e. transformed into an *enactable* PM consisting of the process definition, process inputs, assigned enactment agents and resources, an initial enactment state, an initiation agent, and continuation and termination capabilities [53]. The exact content of the instantiated PM depends on its nature, i.e. is it simulation-oriented, guidance-oriented, or enforcement-oriented. A distinction can also be made depending on whether the PM is instantiated before its execution begins or the instantiation is incrementally done during its enactment.

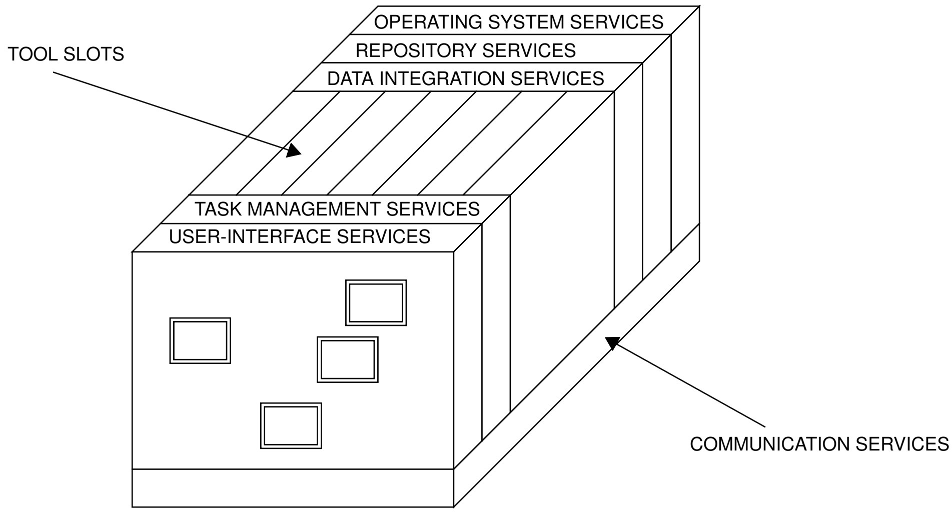

Figure 1 IPSE structure according to the ECMA reference model

2.2 Conceptual PCDE Architecture

The conceptual architecture of IPSEs in general and PCDEs in particular can be described by the ECMA reference model [47]. This model follows a service-oriented view of IPSEs abstracting from concrete implementation details and concentrating on the capabilities of the environment. An IPSE is a collection of software and hardware components which can be conceptually described as a framework providing services to various end-user tools. The services provided by the framework build a common infrastructure and are at least roughly equivalent to the combined capabilities of an operating system with its communication infrastructure, and an object management system. Additional services that may be provided by an IPSE include a common presentation layer providing a uniform interface to end-user tools, policy enforcement services (e.g. access control), and environment administration services.

An IPSE framework supports implicitly or explicitly the notion of a software development process. Non-process-oriented IPSEs are built around a fixed implicitly defined development process. In process-oriented IPSEs (PCDEs) the choice of the specific process used in a particular development project is left open and its definition and execution is supported by a set of process management services: *development*, *enactment*, *visibility*, *monitoring*, *transaction*, and *resource* services. For more detailed descriptions of the possible content of these services see [52].

2.3 DBMS and OMS

Database management systems (DBMSs) are general purpose software systems used for the secure, reliable, and long-term management and use of large, integrated, multi-user data sets. They support application-independent data structuring, retrieval, and manipulation by means of *data models* (e.g., the relational model). Data structures are defined in the *database schema*[2], and concrete data instances are stored in the *database*. A *database system* (DBS) is a DBMS together with a concrete database and its schema.

2. In behaviorally object-oriented data models also type-specific operations can be defined.

DBMSs also implement a *transaction model* that allows multiple users to access the database concurrently, and that ensures that the database can be recovered in case of media failure, system crash, or transaction abort.

The term OMS usually refers to the components of SEEs which provide the capabilities to manage project data. These are either custom-made or general-purpose systems adapted to a specific SEE. Although it is difficult to agree on a set of minimal requirements, the functionality provided in current systems usually includes persistent storage of sharable data. OMSs however lack more advanced functionality encountered in DBMSs such as query facilities, transaction management, and consistency control.

2.4 Existing Concepts and Solutions

A large amount of work has been performed in the area of appropriate data and transaction models for design applications, including CASE. The traditional data models (such as the relational model [32]) and the traditional transaction model of flat, short, and serializable transactions [60] have been successfully used in traditional database applications. Nevertheless, in the eighties researchers began to investigate non-standard applications (of which CASE is a typical example), and soon recognized that practically all DBMS-concepts had to be modified or extended in order to make DBMSs suitable for such application domains.

First attempts tried to extend the relational model, e.g., XSQL [61]. It was soon recognized, however, that the relational model even in this extended form is not well-suited for SEEs [23]. Instead, more flexible data modeling facilities, including a broader set of built-in data types, user-definable data types, flexible relationship mechanisms, complex objects, version structures, and more subtle constraint definition facilities are needed.

For these reasons, object-oriented data models have been proposed, with CASE and other engineering applications as a driving force. One of the first full-fledged design DBMSs, the DAMOKLES system [43], offers powerful data modeling concepts, such as user-definable types, complex objects (modeling the part_of relationship), explicit and attributed ER-like relationships [28] with arbitrary arity, object versions with acyclic graphs, and long fields for large unstructured data items.

Other representatives of this kind of DBMS include Cactis [63] and GRAS [69] which offers graph structures as present in attribute grammars for data modeling, SEED [59] addressing the issue of modeling vague or incomplete information, and the Software Information Base (SIB) [37] which uses TELOS [88] as its data model. All these systems focus on the structural aspect of data modeling (and therefore are called *structurally object-oriented* [44]), while another group of systems (called *behaviorally object-oriented*) support classes, inheritance, and type-specific behavior, but not complex objects. In addition, much research has been devoted to the support for versions [68]. Relevant issues in this respect include version modeling facilities, semantics of versions and the operations defined for them, consistency of version graphs, and efficient storage of versions.

Behavior in database systems (even in object-oriented ones) is still "passive" in the sense that they react to explicit requests from users or applications. Active database management systems (ADBMSs) [e.g., 46] have been a topic of recent database research; they are able to react to predefined situations in the database and its environment and to perform corresponding actions. This kind of reactive behavior is typically specified in terms of event-condition-action (ECA) rules or *triggers*. The semantics of ECA-rules are "*when* the event occurs and *if* the condition holds, *then* execute the action".

Another major issue in database support for SEEs is transaction management. It has been recognized that short, atomic, and conflict-serializable transactions are not appropriate units of work in this domain. Instead, in SEEs units of work may last over long periods of time (maybe months), typically have an internal structure, and are often characterized by the need to let developers cooperate. Furthermore, the traditional correctness criterion of conflict serializability (implemented by the classical read/write strict two-phase locking protocol [24]) is by far too restrictive, prevents cooperation, and would force designers to wait overly long for required data.

Consequently, the use of long-lived, nested transactions with relaxed serializability has been proposed [e.g., 6]. Other proposals extended the existing concept of a "database user" so that project structures can be modeled, e.g., as acyclic graphs of teams and designers. Hierarchies of private, public, and semi-public databases have been introduced in order to provide users and teams with workspaces. Finally, the checkout/checkin mechanism has been developed that allows users to transfer design data between the various databases in a project organization (see [95] for an approach that combines project structures, private and public databases, and the checkout/checkin mechanism).

Another approach for combining developer teams, cooperation, and transactions assigns transactions to the various groups and the individual designers, and explicitly specifies correct and incorrect sequences of design steps in the form of state automata [89]. Also in order to dynamically restructure long, interactive transactions, allowing more flexible transaction schedules, the *split* and *join* transaction operations have been proposed. For a survey of some of the advanced transaction models, see [11, 48].

A further important issue in (software) engineering database systems is *consistency maintenance*. Again, the functionality offered by traditional systems and models has been recognized as insufficient. Extending proposals include support for triggers in the DBMS [73], integration of consistency constraints with the transaction model [72], and consistency maintenance for version graphs of design objects (i.e., change notification and change propagation [68]).

In the recent past, database researchers have started to investigate *workflow management systems* (WFMSs) [56]. A WFMS supports the computer-based modeling and enactment of (any kind of) processes, where the processing entities can be software programs (including database applications) or human beings. Workflows are rather similar to software processes, and indeed some authors subsume software processes under the notion of workflow [e.g., 76].

2.5 Requirements for DBMS Support in PCDEs

In this section we examine in some detail the specific requirements PCDEs pose on their underlying DBMS. Other surveys pertaining more to non-process-oriented SEEs can be found in [23, 49, 79, 90]. A recent short survey of database requirements for process technology is in [50]. The requirements in particular can be derived from:

- the nature of the data that is produced and manipulated in these systems, and
- the nature and context of the operations that have to be performed on this data.

Data representation. The data created and managed in PCDEs consist of both, software engineering artifacts (henceforth called production data) and process descriptions. Software production data can have any of the following types:

- source code, compiled code and executables,
- requirement specifications and design data,
- planning and scheduling information,

- configuration management graphs (see below), and
- test cases and test documentation (e.g. test-protocol results).

Among these data, different kinds of relationships may exist which also have to be adequately modelled. Examples of such relationships include inheritance, composition, versions, interchangeability, usage, etc. Process representations have to include all the information required for process modeling and process enactment. These comprise the definition of the required inputs, assigned enactment agents and tools [53], description of process activities, roles for modeling activities requiring human interaction, and historical data on the performance of past processes. In addition to the previously mentioned production data relationships, we may have various constraints among process activities and relationships between activities and responsible agents. As a consequence, powerful enough data definition and data manipulation languages must be provided.

Process representation. Further requirements for the data model depend on the formalism used for the representation of development processes. Currently, there are various process programming paradigms used in PCDEs. Among the most common we note procedural programming (e.g. APPL/A [101]), rule-based process descriptions (e.g. Marvel [13]), and petri-net or automata-based formalisms (e.g. SPADE [9]). Each of them offers specific advantages and disadvantages so that usually a combination of paradigms is used in a specific environment. In any case, the data model should allow the description of process elements at a level semantically close to the process representation formalism used. In addition, the DBMS should allow the efficient management of the process descriptions.

When petri-net formalisms are used, for example, the DBMS should provide support for their representation of places, tokens, transitions, and arcs. In rule-based process programming, different forms of rules are used to describe the software process: event-action triggers, condition-action production rules, or ECA-rules.

When triggers or ECA-rules are used, the underlying DBMS has to provide mechanisms for the definition and notification of "events" as a means for reacting to occurrences of specific anticipated or unanticipated situations in the database itself or its environment. The use of the event concept also allows the efficient monitoring of process activities. In order to effectively support rule-based process programming, the provided event types should include at least database operations, predefined points in time, and user-definable events (e.g. external tool operations). The definition of composite events by using appropriate operators allows extra flexibility and power. Database systems which provide ECA-rules are subsumed under the general term of *active database systems* [46]. In order to use the underlying active DBMS as a basic process execution engine, it has to provide functionality corresponding to the one discussed in the third class of ADBMS described in [46]. Such ADBMSs can be used for control and integration of heterogeneous application systems and provide correspondingly powerful rule execution mechanisms allowing action execution partly outside local control.

Process evolution. Software processes may evolve during their execution, due to modifications in the requirements and specifications of the development products, performance measurements indicating process shortcomings or inefficiencies, and changing conditions in the environment. Thus a PCDE should efficiently support the evolution of processes during their execution while keeping consistency constraints unchanged. In order to support process evolution, the system should allow dynamic "on-the-fly" changes of process model elements during process execution. Elements that may change include activity definitions, product descriptions, resources used, consistency constraints, and policies that have to be enforced [83]. This means that the DBMS must allow the modification of schemata during the lifetime of a database (e.g. flexible type modification, multiple type versions). The changes to a running process depend on and may be limited in their scope by the contents of the model itself or by model-external factors. They should, however, not be limited by the process representation formalism. A meta-process should thus exist providing a framework where the consistency of the model can be defined and maintained. Information about this meta-process may be stored in a meta-database.

Concurrency control and recovery. In order to ensure the operational integrity of the data, transactional (including recovery) mechanisms should be provided in some form. However, such mechanisms should not affect the degree of possible parallelism and retrieval time adversely. When accessing software engineering data, the data may be locked for long periods of time during which a developer may perform various changes. Additionally, some of the requirements set on ACID transactions might have to be relaxed. Non-standard transaction and recovery mechanisms [48] have to be used in such cases, often combined with the notion of private user workspaces (checkout/checkin). However, a transaction model which offers correctness guarantees in one specific development process may not be adequate or may be too restrictive for others (e.g. production vs. prototyping). Thus ideally, a DBMS should allow the definition and use of different cooperation strategies depending on needs of a specific process (or a part of it) and not impose a particular one on all processes.

Versions and configuration management. Due to the evolution of the descriptions of development processes and of software products, configuration management functionality has to be provided in a PCDE. Often multiple versions of the artifacts used in PCDEs have to be stored and accessed in parallel. Different versions of the same artifact can be used either in the same development process or in other development processes (e.g., when reuse is practiced). Also, different versions of a particular process can be enacted at any time. Configurations are built consisting of various artifacts of different versions and have to be efficiently managed. The DBMS has to allow and support the derivation of new versions, removal of subversions, automatic merging of versions, and maintain version histories. According to [50], even different versioning paradigms should be supported, since these depend on the process model used.

Multi-language interoperability. The DBMS used in a PCDE has to allow access from different programming languages. This stems from the fact that the various tools used during software development in an organization may be written in different programming languages and provide different programming language interfaces and different API styles. In order to satisfy this requirement, two facts have to be considered:

1. The DBMS should be accessible from different programming languages by providing the appropriate programmatic interface, i.e. the choice of provided DML should not limit the usability of the DBMS to specific application domains.
2. Programs written in different languages should be able to share common data, ideally by using a common data model or alternatively by the provision of pairwise conversions, i.e. an adequate degree of heterogeneity must be provided.

Distribution and heterogeneity. Multiple interconnected workstations, often with different operating environments, are used during the software development process. Thus a requirement on DBMSs used in PCDEs is their ability to operate in distributed heterogeneous environments. This usually has repercussions on the DBS architecture which has to support location and hardware transparency.

Efficient storage. A final requirement we consider stems from the nature of the data stored and manipulated in PCDEs. The DBMS should allow the efficient storage and retrieval of different kinds of objects of highly varying sizes and types. Although the retrieval time is not highly critical, it should be in the same order of magnitude as when using a conventional file-system based storage.

From the above discussion it should be clear that DBMSs/OMSs are part of the essential infrastructure of a PCDE. All existing PCDEs (as will be obvious from the survey in section 3) use such a component to support software process modeling and enactment to some extent. The tightness of the DBMS integration in the PCDE architecture (degree of coupling) depends on both, the process modeling formalism and the DBMS/OMS functionality.

3 State of the Art

In this section we survey a number of existing PCDEs with regard to the representation of process models, their enactment, and the support provided by the underlying DBMS/OMS. We also compare their capabilities with the previously listed requirements.

3.1 Adele/Tempo

3.1.1 General Characteristics and Architecture

The Adele/Tempo [14, 16, 17, 18, 51] environment developed at the Computer Sciences Research Laboratory and Applied Mathematics Institute at Grenoble is composed out of a software engineering database (resource manager), an activity manager (responsible for control integration) driven by so-called *temporal event-condition-action* (TECA) rules, and a process manager based on the activity manager (providing communication between process agents, working environments, and configuration management). These components are integrated through a language ("Tempo") describing the data schema and the behavioral model of the system.

3.1.2 Process Representation and Enactment

Dynamic aspects of the environments constructed on top of the Adele kernel system are defined using the Tempo language by constructing process models composed of various software process types. A software process type corresponds to an activity and can be defined recursively. Process resources are described using the concept of *roles* which customize an object's contextual behavior by adding temporary attributes and methods to its definition.

The flow of the software process is controlled by temporal constraints described through TECA rules. These rules are used in the data model to express integrity constraints on object relationships (pre- and post-conditions on origin or destination method execution) and in the activity model to express the execution order of activities, their synchronization, and constraints on the use of software resources (pre- and post-conditions of user actions).

Process activities in Adele are defined as *work environments* (WEs) which include in their definition the object instances the step will operate on, along with the activity descriptions, tools, users, etc. A WE can be regarded as a long transaction. Changes to the affected objects are visible only within a WE. A SEE is seen as a federation of collaborating WEs for which roles are the unit of collaboration. Roles may collaborate by using a library of semantic relationships.

3.1.3 DBMS Support

Adele uses a proprietary OMS providing a data model which is an extension of the ER-model; it supports complex objects called aggregates, binary relationships between an origin and a destination object, and multiple inheritance. Versioning functionality is provided. Each object may have a version branch (i.e. a sequential list of revisions) which is a first class object in itself. Methods can be defined as part of an object type and are written in a simple shell-like imperative language. Different schemata can be used in different logical database partitions.

Triggers (event-action associations with conditions defined as part of the event) can be defined for an object or relationship type. The event is a predicate over the past or present state of the system or the object base (i.e. object state, database operations and activities underway) and the action is one of the methods of the object type or its ancestors. Events can be composed through conjunction and disjunction operators and are checked each time a method is called. The defined classes of triggers (event-action association) are shown in Table 1:

Trigger class	Execution time	Corresponding HiPAC coupling mode [82]
pre-trigger	before method execution	immediate
post-trigger	after method execution	deferred
after-trigger	after transaction commit	decoupled
error-trigger	after transaction abort	decoupled

Table 1 Trigger classes in Adele OMS

3.1.4 Evaluation of DBMS Support

Adele/Tempo provide various concepts to support the construction of PCDEs. The main strengths of the Adele OMS used can be summarized as follows:

- active relationships can be used to describe contextual behaviour,
- triggers can be used to express integrity constraints and development policies.

However, the following limitations exist in the current functionality of the Adele OMS:

- limited support for design transactions,
- only limited types of active behaviour (method execution events) are supported,
- TECA rules have to be mapped onto Adele trigger definitions instead of being directly executed by the DBMS. This limitation is known to the authors and addressed in [85].

3.2 ALF

3.2.1 General Characteristics and Architecture

ALF [27, 91] is a meta-CASE environment for generating process-centered software development environments supporting different life-cycle models, design and development methods. The approach used is to provide the means to describe process activities and to integrate mechanisms which implement these activities. Process models, described in a special formalism called MASP, can be used to instantiate an environment and can be enacted in the instantiated environment. An ALF-based development environment is built by plugging compiled MASPs and environment-specific tools into a predefined framework containing process management services. The bottom layer of all environments is a set of PCTE [106] interfaces providing object management, execution, communication and distribution mechanisms.

3.2.2 Process Representation and Enactment

The MASP (meta-Model for Assisted Software Processes) formalism provides description mechanisms for software processes. A PM is described as a hierarchy of MASPs. Each MASP description is composed of the following parts:

1. A *schema* (see below).
2. A set of *operator type definitions* in which pre- and postconditions (logical expressions) are used to describe activities performed by tools and other complex activities (MASP-activities).
3. A set of *orderings* expressing restrictions on the execution order of the operators. Two operators can be executed sequentially, alternatively, concurrently, or grouped together. An operation can be executed arbitrarily often, repeatedly, or simultaneously with other occurrences of itself.
4. A set of ECA-rules defining automatic reactions to predefined events that occur during the software process. The events can be either database operations (read, write, update, delete, and locking of objects, attributes or links) or user defined abstract events. The rules have the following form:
   ```
   IF ON event EVALUATE condition EVALUATED {true | false} THEN action
   ```
5. A set of *characteristics* describing constraints on the software process states that have to be enforced during process enactment. Characteristics are logical expressions associated with an optional event part. They have the following form:
   ```
   [ON database_operation database_object] EVALUATE logical_expression
   ```

A MASP interpreter is used for the enactment of the development process. The interpreter is a PCTE process and uses the aforementioned production rules. It invokes PCTE OMS operations in order to request services from PCTE and to invoke CASE tools which are stored in the PCTE object base.

Several agents in an environment (which play the same role) can access the same objects, invoke the same operations and have the same objectives. Working contexts which support these roles cooperate by sharing instances through PCTE views.

3.2.3 DBMS Support

As already mentioned, ALF is based on PCTE. The typed ER-model provided by PCTE has been extended with multi-valued attributes and triggers (event-action associations). The model allows the definition of new attribute types based on existing types, and of new object types by subtyping and extending existing object types. Links (unidirectional associations) can be established between an origin and a destination object. Relationships are defined by a pair of inverse links.

There is no single global schema. Types are grouped into self-contained schema definition subsets (SDS). Sharing of data that belong to different SDSs can be achieved by importing type definitions explicitly. Two SDSs with the same definitions can see the same data instances (have the same view). Shared instances are subject to PCTE access control and can be used in PCTE short transactions. A process executes in the context of a working schema, which is an ordered set of SDSs. PCTE was extended by adding triggers which directly support the triggering mechanisms used in MASP characteristics.

3.2.4 Evaluation of DBMS Support

The strengths of the ALF OMS can be summarized as follows:

- use of standardized components for object management, cooperation and distribution (PCTE), and
- support for process model evolution, especially through an inconsistency checking tool.

However, the following limitations and shortcomings still exist in the current state of the ALF project:

- limited object model based on an ER-extension with multi-valued attributes and triggers,
- no support for long-lasting transactions and nested transactions,
- the event concept is not well-integrated with the OMS and no event algebra has been defined. There is no well-defined connection between rule execution and transactions.

3.3 Arcadia

3.3.1 General Characteristics and Architecture

The Arcadia project [101, 102] has as its main goal the construction of integrated, flexible and extensible PCDEs. There is no single Arcadia environment, but rather a series of environments constructed out of various partly interchangeable components. The basic architecture of these environments consists of a user interface management system providing a common interface to various tools operating on the underlying object management system. The third important component is a process program interpreter controlling software process activities. Arcadia environments can have many different configurations. A typical representative of each component will be presented here, while other possible components will be just briefly mentioned.

3.3.2 Process Representation and Enactment

The process programming language used in Arcadia is APPL/A. It is an extension of Ada featuring persistent programmable relations between objects, concurrent triggers that react to operations on relations, predicates which can be enforced as constraints on relations, and special statements to program high-level transactions. APPL/A uses the Ada type system, its modularization concept of packages, and the rendezvous task communication paradigm. Process definitions in APPL/A are procedural, but rules can be used to specify consistency constraints.

3.3.3 DBMS Support

Various object managers are used in Arcadia environments (e.g. Triton [62], PGraphite [108]). Triton is an OMS with a client/server architecture based on Exodus [26], which consists of a storage manager and a persistent programming language called E [96]. The type system of Exodus is comparable with that of C++. Triton augments Exodus by providing APPL/A relation and simple trigger support (see below). In Triton, trigger events are method or function invocation and execution completion. A dynamic loader provides some flexibility with respect to the dynamic change of object methods. Multi-language access is achieved by providing a remote procedure call protocol, with Triton residing in an address space different from that of other (non-C) language programs. Data sharing among multiple languages is possible by using as a common data model a subset of E to which the type systems of other languages have to be mapped.

3.3.4 Evaluation of DBMS Support

The main advantage of the Arcadia approach is the use of an architectural concept optimized for experimentation with various interchangeable components. The underlying object management system is used only for the persistent storage of objects. Arcadia-based environments are language-oriented, as the process programming language has to provide a very large part of the functionality needed for both, process control and data modeling. Functionality usually provided by full-fledged database management systems has to be explicitly coded in APPL/A, which may be a very tedious task.

In the current version of Triton the trigger mechanism is relatively primitive. The types of actions generating events are limited and no composite event definitions are possible. Furthermore, no transaction management facilities are provided; instead all needed transaction management features have to be programmed in the process programming language. A further problem is the need to map the type systems of various languages to the common data model provided by E.

3.4 EPOS

3.4.1 General Characteristics and Architecture

EPOS [34, 35, 42, 77, 78] is a multi-user process-oriented software development environment providing basic facilities such as process modeling and configuration management (CM). It has a layered architecture consisting of a common user interface to various programming tools, of an activity manager, and a planner. The process modeling language SPELL is based on and extends the underlying structurally object-oriented database system EPOSDB.

3.4.2 Process Representation and Enactment

The process modeling language SPELL is an extension of the EPOSDB DML/DDL and consists of two layers. The lower level adds behavioral and reflexive object-orientation to Prolog (types and meta-types). Classes have an instance-level and a class-level part with (public and private) attributes, procedures, and triggers (on procedure calls) defined in each. Various types of single inheritance are provided for class-level attributes and class/instance-level procedures (`redefine`, `append`, `concatenate`). Attribute access and binding is dynamic and is implemented by the SPELL interpreter. The upper level of SPELL adds tasking by making *task objects* concurrent. Tasks are connected in a typed network with chaining and decomposition.

The basic concepts used for PM are *activities* and *products*, which are described as SPELL types. Process artifacts are represented by model entities (objects) with mutual relationships. A PM consists of relevant model entities and relationships, together with their process schema with types and meta-types. The activity model consists of a set of task types (subclasses of

`TaskEntity`), and the product model of a set of data types (subclasses of `DataEntity`). Software process steps (activities) are instances of task types. A task has the following properties:

- pre- and post-conditions (static and dynamic) which are logical predicates,
- code defining the steps of task execution (in Prolog),
- decomposition relationships to subtasks in the task network,
- formal input and output parameters.

PM instantiation is done by the *planner* component, which is implicitly and incrementally called by the *execution manager* responsible for PM enactment. The planner generates subtask networks for each composite task (hierarchical planning). It applies backward chaining and hierarchical decomposition combined with domain specific knowledge to build a plan in order to achieve a given goal. Planning is based on the process schema and a model of the product structure. The generated plan has to satisfy the requirement that every pre-condition of every node (task) in the plan is matched by a post-condition of another node. The execution manager schedules, interprets, executes, controls and records tasks while cooperating with external tools and users. The EPOS type system is shown in Figure 2.

3.4.3 Process Model Evolution

Explicit support for the evolution of PMs is given in EPOS by a *meta-process* consisting of *meta-activities* used to manage software processes. These meta-activities are:

- provision of a process support environment (PSE),
- analysis and design of a generic PM schema,
- customization of the generic schema with project and application specific information,
- instantiation of the PM and subsequent process enactment, and
- continuous performance assessment during the process enactment.

This meta-process description provides the framework for PM evolution which can refer to either the instantiated PM/enacting process, the generic PM schema or the meta-process itself. The entire PM underlies a life-cycle and CM control. A component called the *evolver* manages type evolution by following rules for changing and creating types. Such rules are specified as type-level procedures.

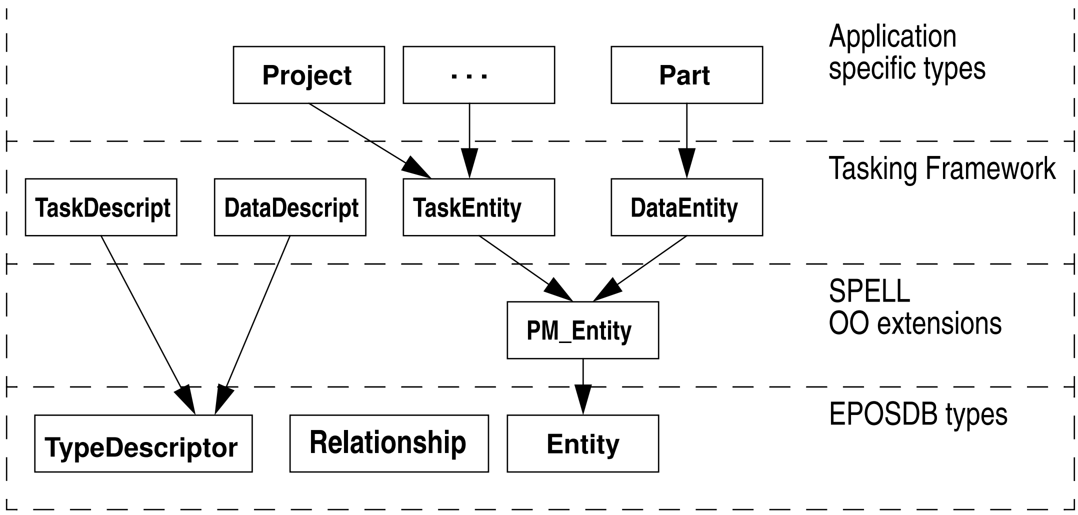

Figure 2 EPOS type system layers

3.4.4 DBMS Support

The underlying database system EPOSDB has a data model similar to the *object-relation*[3] model suggested in [97]. Objects have a unique and immutable identity. All database operations are performed within the context of long (non-serializable) and nested transactions with checked-out workspaces. Nested transactions can be used as substructures of long transactions. Transactions may survive multiple application sessions and can be interactively started, committed or aborted.

The DBMS provides a change-oriented versioning model which enables uniform versioning of objects and relationships as well as of schema-level information. A version is not an explicit object but can be evaluated as a combination of selected incremental modifications: instead of using version graphs, logical changes can be freely combined (constrained only by version rules). Versions apply to any granularity from atomic objects to entire databases.

3.4.5 Evaluation of DBMS Support

In EPOS, extensive support is available for process modeling through an extension of the base type system. The underlying DBMS provides elaborate version and configuration management functionality. The structurally object-oriented data-model provided by the EPOSDB has however limited expressiveness.

3.5 Marvel/Provence

3.5.1 General Characteristics and Architecture

Marvel [12, 13, 20] is the result of a research project at Columbia University developed between 1987 and 1992. It started as a single user system but evolved into a multi-user, rule-based software environment focusing on the aspects of consistency maintenance and automated control in software development. Provence [74] is a PCDE using Marvel as one of its components. It focuses on creating an open PCDE architecture allowing the incremental integration of external software tools. A short description of the main components of Provence is given at the end of this section. Oz [21, 22] is another system extending Marvel and focusing on the decentralization aspects of process-centered environments. The issues researched in Oz are beyond the scope of this paper and are not described here.

Marvel has a layered client-server based architecture. Many development environments may be active in parallel, each one of them has a separate repository and is managed by a single centralized server. The server provides data integration, repository and task management services to multiple distributed clients, each one of them corresponding to a single user session.

3.5.2 Process Representation and Enactment

In Marvel an environment administrator specifies a model of the process in the form of a *project rule set*, a schema of the data called the *project data set*, and interfaces to external tools which are called envelopes (*project tool set)*. The development process is modelled by *activa-*

3. A structurally object-oriented model not to be confused with object-relational models used in e.g. Illustra and UniSQL [70].

tion rules, which control tool invocation when certain preconditions are met and specify effects of the invocation on object attribute values, and *inference rules,* which define relations among object attributes. Rules are parameterized to take as arguments objects which are instances of specific classes. Loading the specifications into the kernel is equivalent to compilation with strong typing. When loading succeeds, end-users are provided with a choice of commands reflecting the project rule set.

Forward and backward chaining is used to automatically fire rules when rule conditions are satisfied. If a condition is not satisfied, backward chaining of other rules whose effects may satisfy the original condition is applied. When the condition is satisfied, one of the rule effects is asserted which may cause further rules to fire. This may satisfy the conditions of other rules which are then fired (in a forward chaining manner).

In order to distinguish between consistency and automated execution, it is possible to define both consistency and automation predicates in a single rule. The algorithm implements a so-called *maximalist assistance* mechanism. If an automation predicate in the condition of a rule is not satisfied, Marvel tries to satisfy it by backward chaining. If a consistency predicate in the condition is not satisfied, Marvel refuses to execute the activity. The assertion of a consistency predicate in the effect of a rule mandates that either all rules in the implication set must be successfully fired, or the original activity is rolled back. The assertion of an automation predicate in the effect of a rule causes Marvel to attempt to carry out all implications of the predicate on a “best effort” basis.

Modularization and reusability of PMs is achieved in Marvel through the concept of *views*. A view consists of a subset of the project rule set, the project type set, and the project tool set. Different views might be loaded by the same user during different phases of the development process, and different users may load different views at the same time to fulfill different roles. There are two correctness criteria: (i) the type set of the view must match the assumptions of its rule set (self-consistency); and (ii) the set of rules in the view must be complete with respect to the project rule set, i.e. for each rule in the rule set, the set includes all the rules that are members of the consistency implication set of the rule.

The correctness criteria required for views must be reevaluated when the process model or the production data schema is modified. The schema must always match the process model and the process model must be always complete. Possible mismatches may arise only because of consistency predicates and not because of the automation predicates. If only automation predicates existed, there would be no restrictions in adding/removing rules at any time. An evolution step is permitted in Marvel if the consistency implications specified by the rule set after the evolution step are either weaker than the implications before the evolution step is carried out, or are independent from them.

3.5.3 DBMS Support

A *storage manager*, a *file manager*, and an *object manager* (OM) are responsible for repository services in each environment. The storage manager handles low-level disk and buffer management for control data. The file manager can either manage files requested by the OM, or just maps objects to files and vice versa. The OM implements a structurally object-oriented data model for the product and process data, providing generalization/specialization of object types. Four types of attributes are supported: object state, product data files, part-of-relationships, and general links between objects.

Transaction management functionality (for nested transactions) is provided by a transaction manager, a lock manager and a coordination manager. The top-level transaction corresponds to the execution of exactly one user command. A lock protocol based on strict two-phased locking is used. The execution of some concurrent schedules is based on analyzing the consistency constraints imposed by a specific development process. The analysis is based on the types of transactions involved (built-in, top-level, consistency forward chaining, automation forward chaining, and backward chaining), the state of the interfering transactions (their progress so far), and if they are interactive or not.

3.5.4 Evaluation of DBMS Support

The data model of OMS used in Marvel is rather specialized and somewhat limited in its functionality (e.g. only predefined attribute types are allowed) so that it cannot be used to model all aspects of the development process. Its advantages include the possibility to construct a virtual file system through the mapping of files to OMS objects and the provision of nested transactions.

3.5.5 Provence

Provence [74] is a project at AT&T Bell Laboratories. Its emphasis is on defining an open PCDE architecture allowing the incremental introduction of features and providing the means to integrate existing software tools. The logical architecture of the Provence kernel consists of five components which according to the opinion of the architects belong into every PCDE [74]: a *process server,* a *data management subsystem,* a *smart file system,* an *event engine,* and a *visualizer.* Marvel is used to provide the functionality of the process server and the data management system. However only process data and a description of product data is stored in Marvel, while all actual product data is stored in the file system. The event engine, Yeast [75], is responsible for the detection of specified event patterns and the triggering of corresponding actions. Simple events that can be recognized are absolute and relative time events (periodic and in monitoring intervals), changes of object attribute values, and explicitly notified *announced* events. Complex events can be constructed through conjunction, disjunction and sequence operators. The action is usually the execution of a command script. A special component (Enactor) mediates between Marvel and Yeast by generating Yeast specifications based on Marvel's process and data models, and by translating matching events in Yeast to Marvel actions. The file system notifies Yeast of changes made by tools to its contents (file creation, deletion, modification etc.).

3.6 Merlin

3.6.1 General Characteristics and Architecture

Merlin [93, 94] was developed in a cooperation between the University of Dortmund and STZ - Gesellschaft für Software-Technologie mbH. It is a prototype SDE using a rule based software process description and emphasizing process enactment in a multi-user environment.

3.6.2 Process Representation and Enactment

Process definitions in Merlin consist of roles (e.g. project manager, programmer), activities (e.g. specify, compile), software objects (e.g. modules, test plans) and resources (people and tools). Roles are groups of logically related activities. The development process is described in a special rule-based language similar to Prolog in which special kinds of facts are used describing roles (*work_on* facts), documents (*document* facts), responsibilities (*responsibilities* facts), activities (*task* rules) etc. in a declarative way. The Prolog backtracking mechanism is used to select one specific task to be executed. Thus backward chaining is used to select objects and activities a user may access and perform, depending on his role, and to query information about the status of a project. However, forward chaining is also used for automating the execution of machine tasks. Forward chaining rules consist of a rule name, a set of preconditions (queries on the rule base), a set of activities (CALL an external tool, INSERT or REMOVE a fact in the factbase), and a set of postconditions. Rules and facts (but not the process products themselves) are stored persistently by GRAS (see below) in an abstract syntax tree-like way. Rules are stored in a way that allows their efficient retrieval for interpretation. A process engine interprets the rules in the rule base.

The most important structuring concept in Merlin are *working contexts*. A working context provides the interface to the objects a user can operate on during a software process. The legal operations for a user at a particular point in time are included in his working context, and the cooperation of multiple developers is attained by coordinating the working contexts of various users. A three-level working context hierarchy is maintained. At the lowest level, a *project working context* (PWC) includes all objects or parts thereof which are necessary to coordinate the entire project. Members of teams are coordinated by *team working contexts* (TWC), and the remaining commonly accessible information is kept in *team member working contexts* (TMWC). The transactions which manage the construction of TWCs and are executed by the PWC are based on a checkout/checkin mechanism. Some temporary inconsistencies have to be tolerated. On the other hand, transactions managing the TMWCs enforce exclusive locking and immediately update data in the TWCs of team members. By building hierarchies of TWCs, some flexibility between these two extremes can be achieved.

3.6.3 DBMS Support

The underlying storage mechanism of Merlin is GRAS [69]. The data model provided by GRAS is based on attributed graphs. Abstract node classes are used to model sets of nodes with a common interface. Node classes form an inheritance hierarchy. Multiple inheritance is supported. Concrete node types are instances of node classes with different implementations (i.e. calculation of derived attributes) and nodes are in turn instances of node types. Node types may have attributes that are either *intrinsic* (values are assigned explicitly) or *derived* (values are calculated from values of other attributes in the same or neighboring nodes). Edge types represent intrinsic binary (bidirectional) relations between nodes. Paths are derived relations used to define complex graph queries. Binary directed relationships between objects are modelled as non-attributed edges.

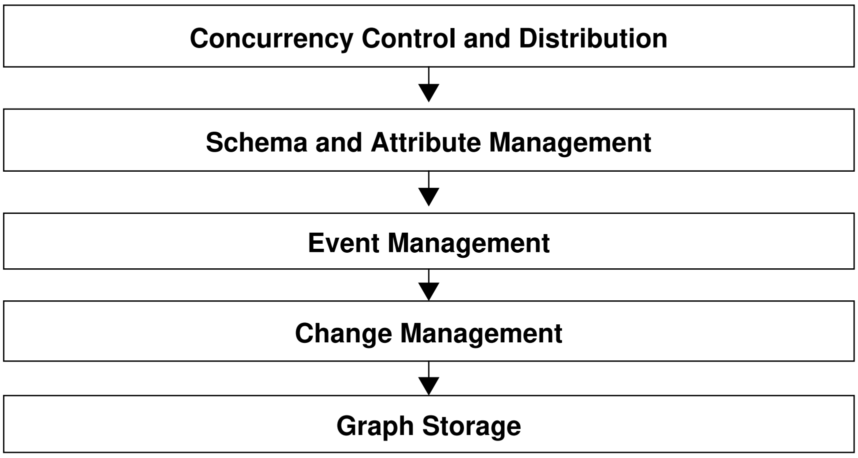

Figure 3 GRAS system architecture

GRAS has a layered architecture (see Figure 3).

The GRAS kernel provides a graph-oriented interface allowing creation, deletion, and access to graph components. Additional functionality is provided by the following layers:

- *Change management layer*: recovery (undo/redo, checkpoints) of user and system-initiated actions, and version management. Nested transactions are also provided by this layer.
- *Event management layer*: event-action triggers, with priorities and parametrized actions. The possible events are limited to basic graph operations (insertion, manipulation, and deletion of nodes and edges).
- *Schema and attribute management layer*: preservation of graph consistency by controlling permitted graph operations and computing derived attribute values.
- *Concurrency control and distribution layer*: locking and distribution of entire graphs (not nodes). Exclusive access to a graph within one access group can be defined at the operational, transactional and session level. Each access group has a private copy of the graph and more than one client application may belong to an access group. Only one of the copies of an access group graph can be persistently modified.

3.6.4 Evaluation of DBMS Support

The process representation formalism used in Merlin is similar to Marvel. The efficient persistent storage of the process representation and the state of running processes has been a major issue. However, the concurrency control functionality provided by GRAS is a limiting factor with respect to developer cooperation and is only partially compensated by the working context model used.

3.7 Oikos

3.7.1 General Characteristics and Architecture

In the Oikos [1, 2, 4] project, a set of concepts and an architectural framework have been defined for the construction of PCDEs. An enactable PM consists of a hierarchy of reactive systems belonging to predefined classes and interacting with each other. Customizable predefined services provide basic functionality to support process enactment by accessing allocated resources. Run-time support is provided by the Expo [3] system which provides distributed execution of PM components, integration of external tools and human interaction. It consists of a set of components exchanging messages. A monitoring component controls distribution and resource allocation.

3.7.2 Process Representation and Enactment

A software development process is a set of concurrent activities associated with agents and objects. Agents represent automatic components and human beings. Objects are development products (deposited and working documents), tools, computational resources and process representations. In Oikos a process model describes activities and manages the objects; it is represented by a set of cooperative entities. A specific process is obtained by specializing and composing the process, environment, office, desk and service entity types. Resources needed for the process can be shared, can have access control properties, and can be arbitrated.

A service instance manages a specific concrete resource of the class for which the service has been defined. Each service instance is activated as a component of its parent entity. A service instance is available to its parent entity and to all entities in the hierarchy rooted in the parent. Services are entities and belong to the description of the PM. Tools have a *signature*, a *representation*, an *installation*, and an *application*. They are wrapped in envelopes as in Marvel.

The Patè language used in Oikos for software process enactment is a concurrent logic language derived from Extended Shared Prolog. Limbo is a super-set of Patè used for the specification of the PM entities. Patè is executed by an interpreter in the Oikos distributed run-time system Expo.

3.7.3 DBMS Support

Object management in Oikos is layered: the top layer offers a service interface to process activities; it is based on an object manager (OMK) which provides persistence and a logical global view. The OMK has a structurally object-oriented data model with links between objects. It is implemented as a logic database defined in LDL [29]. The lowest object management level consists of several components each responsible for the management of a different object class. It directly uses operating system functionality.

Through the service interface, the access to the functionality of different resources of a given class is managed. The services important in this context are the *product repository service* (PRS), *tool repository service*, and *workspace management service* (WS). They offer access protocols which guarantee consistent access with respect to the basic model of product and tool management. The PRS defines a concurrency control policy based on checkout/checkin of documents into WSs, and a version control model with releases and revisions based on the SCCS/RCS model [84].

3.7.4 Evaluation of DBMS Support

In Oikos, a considerable effort has been made to abstract from the concrete implementation of a PCDE and provide a PM formalism using the concepts of shared state and of services provided and used by cooperating entities. However, as the authors admit, the current (end of 1994) service access protocols are at a very low level, making structured access to workspaces a tedious task. Furthermore, at this point, a complete process model service which would maintain a persistent state of the enacted model is missing. Too little information on the OMS implementation is currently available to accurately assess its performance, which may suffer from its multi-layered architecture and the use of LDL.

3.8 SPADE

3.8.1 General Characteristics and Architecture

SPADE [7, 8, 9] is a joint project of the Politecnico di Milano and CEFRIEL. Its goal is to provide an environment for software process analysis, enactment, and evolution based on the PM language SLANG. The system architecture distinguishes between a process enactment environment where process artifacts are created and modified, and a user interaction environment which manages the interaction between users and tools in the environment. A central repository is used for the storage of process artifacts and process descriptions.

3.8.2 Process Representation and Enactment

The process model representation formalism in SPADE is based on ER nets, a kind of high-level Petri nets. Transitions represent events that may or may not occur in a given state. Conditions on the occurrence of an event are described at the transition input places. These places are persistent object repositories modeling collections of data, tools, and resources. The actions to be executed when an event occurs are expressed in a logic-based language. Timing information can be associated with tokens and actions.

3.8.3 DBMS Support

The fully object-oriented DBMS O_2 [92] provides repository services in SPADE-1, the first implementation of a SPADE environment. All SPADE data (software products and software process descriptions) are stored in the O_2 databases. A meta-schema allows the description and manipulation of schema elements. O_2 has a client-server architecture with a centralized page server and distributed clients. Each process execution engine is a database client. O_2 provides concurrency control mechanisms based on a hierarchical two-phase locking protocol on files and pages, and a set-oriented query language.

3.8.4 Evaluation of DBMS Support

SPADE is based on a commercial object-oriented DBMS. The functionality of O_2 has been extended by low-level active capabilities [33]. The Petri net formalism used is expressive and well suited for process modeling. A limitation of the DBMS which may seriously affect its usability in a SEE is the lack of support for nested transactions.

3.9 Selected Commercially Available Systems[4]

3.9.1 AD/Cycle

AD/Cycle[5] [30, 86, 98] is an IPSE developed by IBM. It provides support for the definition and execution of the development process. A process description consists of an ER-model of the production data (*work objects*), the definition of activities performed by tools on these work objects, and the definition of the execution sequence of these activities (*work-flow structure)*. The work-flow structure consists of a network of activities (branches and joins) onto which entry conditions and exit assertions (predicates on the work object attributes) are attached. A model administrator is responsible for the definition and maintenance of process models using the *work manager* tool. The work manager interprets the process model presenting a set of possible activities to software developers and invoking environment tools.

The AD/Cycle environment provides repository and configuration management services. Application development meta-data (i.e. the production data model) and production data, are stored in a repository built on top of the *Repository Manager* (RM) which provides an object-oriented data model. Views of these data can be provided to different tools in the environment. RM objects are under version control.

3.9.2 ADELE

The commercially available ADELE IPSE [105] is marketed by VERILOG Inc. and is based on the research project described in section 3.1. The functionality provided by the product includes object management, version and configuration management, and support for developer coordination by means of working environments and integrity constraints on objects. Objects are typed, their types being defined in the ADELE data definition language and organized in a hierarchical fashion. Object attributes of predefined types can be added. Type constraints (logic expressions, ADELE commands) guarding object modification can be defined. Relationships can also be defined among object types. Objects stored in the ADELE OMS include source code, scripts, text, binaries, executables and virtual objects, not associated with files. The ADELE OMS provides atomic transactions, recovery, access control and distribution functionality.

Parallel distributed development in the ADELE environment is supported through the use of private workspaces. Manual and semi-automatic merging of object versions is possible.

3.9.3 Emeraude PCTE

Emeraude PCTE [104, 106] is a distributed SEE-framework implementing the ISO 13719 standard PCTE specification. Extensions to the standard are provided by vendors such as Transtar, a joint venture company of Group Bull and Thomson CSF. Emeraude PCTE provides the means to integrate various development tools through a common interface, tool interoperability (control integration), and sharing of data stored in a common repository. The repository provides an extended ER data model with links between objects. Schemata for the tools integrated in the environment are described as individual sub-schemata. Many subschemata can be combined to a working schema (equivalent to a view) through which a tool can access the OMS. Queries can be defined on the object base by using an extended shell language or a C library.

---

4. In order to provide an overview of the current state of the art in commercially available SEEs, we present here five SEEs which provide process support to some extent. Due to the difficulty of finding accurate literature about technical aspects of commercial systems, our description may not cover the whole spectrum of their functionality. We have attempted, however, by directly contacting their vendors to minimize that risk. The presented descriptions are thus correct to the best of our knowledge.

5. AD/Cycle is a registered trademark of IBM Corporation.

External tools can be encapsulated to interact with the OM and process management environment. Control integration is achieved by message queues assigned to tools and notification.

3.9.4 Maestro II

Maestro II IntegrateIT[6] [99] is an IPSE developed by Softlab (Germany). The system has a client/server architecture, where the server is a demon on top of a proprietary OMS and clients are various tools integrated in the environment. The OMS provides a data model closely related to the ER-model with the following modeling constructs: attributed objects and attributed relationships between objects. Attributes belong to predefined types. The OMS is used for the persistent storage of production data and development process descriptions and provides basic querying and access control functionality. Additional functionality such as the definition of integrity constraints, transaction management, configuration management, and process modeling can be defined through the use of appropriate schemata coupled with procedures in the proprietary procedural language, *Prolan*.

Processes in Maestro II are modelled by describing tasks and results in terms of the modeling constructs mentioned above. Additionally, transitions can be defined on the state of repository data, using *Prolan* procedures. These transitions may for example test conditions on object attributes or automatically call external tools. The environment provides a series of customizable editors and allows the definition of user-interface elements to directly manipulate OMS data. The integration of external tools (e.g. compilers) occurs at the file level. Files are checked out of the OMS, manipulated by the tools and then checked in again. A tighter integration is possible through the use of the OMS C-API.

3.9.5 HP SoftBench

HP SoftBench[7] [41, 38] is Hewlett-Packard's IPSE with an architecture closely following the ECMA reference model presented in section 2.2. It consists of a framework used to integrate various internal and external tools. This integration is achieved by a component called the *Encapsulator* [55] which provides the means to describe external tool functionality in a way usable to other environment components and to model the development process by defining tool interactions. The process is described in special specification language and is interpreted by the runtime component of the *HP Encapsulator.* A message forwarding component allows the communication of environment entities, and an *execution manager* starts up tools which have registered as providers of specific services when these services are requested by other tools. Triggers can be defined which couple the results of functions performed by an environment entity to the execution of functions by other entities (e.g. when a file is checked out, an editor displays it for editing). Sequential, alternative, parallel and iterative control structures can be defined in this way.

Production data are stored in SoftBench in the Unix file system under an RCS-compatible configuration management component. However, an alternative configuration manager, ClearCase[8] [5] developed by Atria Software, may be integrated in the environment. It provides version, workspace, and build management as well as process control. Software components

6. Maestro II IntegrateIT is a registered trademark of Softlab GmbH.

7. HP SoftBench and HP Encapsulator are registered trademarks of Hewlett-Packard Company.

8. ClearCase is a registered trademark of Atria Software Incorporated.

are stored in a *versioned object base* (VOB) implemented on top of a proprietary OMS and are under version control. Manual and semi-automatic merging of object versions is possible. Objects that can be stored in the VOB include source code, binaries, executables, documentation, test suites, compilers and libraries. They may have attributes of predefined types attached and can be connected by links. In order to manipulate these objects, users have to check them out in private workspaces defined by *views*.

ClearCase provides some process automation functionality in the form of pre-event and post-event triggers. Pre-event triggers can be defined to fire before the execution of VOB-operations (or operation classes) associating them with actions which can be programs, shell scripts, or ClearCase commands. The action execution can be guarded by a logical expression over object attributes. Post-event triggers can be defined to fire after a VOB-operation is completed and are used for the notification of users or to attach attributes to objects.

4 Conclusion

4.1 Summary of Observations

Based on the survey, the following general observations can be made regarding database support in current process-centered software development environments (see Table 2 at the end of the paper):

- Many of the surveyed systems use a DBMS or OMS for the storage and manipulation of product data. In all systems, either the basic DBMS/OMS functionality was inadequate and had to be extended or a proprietary OMS had to be constructed. From the systems using a commercial DBMS, only SPADE (with NAOS) uses a fully object-oriented data model.
- All the surveyed systems use a DBMS or OMS for the storage/administration of (usually a part of) the process descriptions. Often, however, the provided data model is not adequate for the complete description of process models and/or software production data. Thus queries and administration operations on this information are cumbersome to implement and have to be implemented using database-external mechanisms.
- Most but not all the DBMSs or OMSs used provide configuration and version management facilities. These have been realized mostly due to the demand pull from the application domain.
- Although all the process modeling languages surveyed use at least in part some kind of event-based formalism, only some of the DBMS and OMS used in the surveyed systems provide support for the modeling of simple active behaviour in the form of event-action (EA) triggers. With the exception of NAOS (used in SPADE), the only event types supported are database operations/object methods and/or user-defined abstract events. Also, most systems lack formal event definition language and rule execution semantics.
- Only a small subset of the surveyed systems provide support for dynamic schema evolution, namely the two systems which are based on a fully object-oriented data model (Arcadia and SPADE).
- Only some of the systems provide extended transaction management functionality. Those that do provide long transactions with checkin/checkout mechanisms and private workspaces. Two of the systems provide a nested transaction model. In the other systems, transaction management functionality has to be explicitly provided by the PM language.
- Only two of the systems provide multi-language interoperability, these being again the two

systems which are based on a fully object-oriented data model.
- Only few systems provide some support for distribution of data and operations, e.g. by being based on a client/server model. However, none of the DBMSs used so far has a distributed architecture. This fact leads to possible performance bottlenecks and/or inefficiency.

4.2 Open Issues

Based on the survey of existing systems and the requirements stated in section 2.5, a series of unsolved problems concerning DBMS support for PCDEs can be identified. We think that these problems originate from three different causes:
- Adequate or optimal theoretical and conceptual solutions still have not been found for a number of issues.
- Due to the relative immaturity of the research domain, many projects have concentrated on finding optimized solutions for specific problems considered, such as process modeling or concurrency control. Thus, integrated complete solutions have not yet been implemented in the context of concrete projects.
- Significant work on database support for SEEs has been done in both the database and the software engineering community. However, our impression is that some approaches have been developed in parallel, sometimes with the effect of "re-inventing the wheel" in one of the communities (of course, with notable exceptions). Examples are active mechanisms in databases vs. triggers in PCDEs, or software processes in PCDEs vs. workflows. It is our hope that in the future SEEs in general and PCDEs in particular might benefit from combined efforts and "interdisciplinary" discussions.

We review here the open issues identified from the survey and present some aspects of possible solutions.

Process and data modeling. In current systems, there is no adequate integration of data and process modeling. Most combine representation formalisms from various paradigms (e.g. a structurally object-oriented model for production data and rules for the process description). However, the DBMS/OMS used do not provide adequate functionality to model and manage both kinds of representations uniformly.

Furthermore, although advanced transaction models have been developed, there still seems to be no consensus on their use in PCDEs. More specifically, there seems to exist a semantic gap between the (extended) transaction models developed by database researchers and adequate transaction concepts for software processes and workflows in general. This observation is sustained by the plethora of papers on process and workflow description languages which propose proprietary transaction concepts. A further issue refers to the need for additional flexibility with respect to the choice of a transaction model depending on the concrete (sub)process executed. Ideally, a framework in which the TM can be defined and customized should be provided.

Tool integration. A different but related problem refers to the integration of heterogeneous external tools within a PCDE. The requirements are conflicting as the integration must be loose enough to allow binding of various tools in a specific process without excessive setup effort. However, it should also be tight enough not to allow back doors outside process control through the use of built-in tool functionality (e.g. an editor allows the execution of shell commands). This kind of integration can be achieved more efficiently and securely when software artifacts are stored in a DBMS and the access and workspace control mechanisms of the DBMS are used, instead of the rudimentary ones provided by normal file systems. However, as we already mentioned, there are still modeling and performance inadequacies if repositories are used for storage of the software products themselves instead of only for their descriptions.

Process evolution. The disciplined modification and evolution of process models is either only supported in part or not at all. Although some modern object-oriented DBMS allow the dynamic (runtime) modification of their schema [10, 109], the use of this functionality on process representations still poses unsolved problems. More specifically schema evolution in PCDE DBMSs has to be performed in the following general cases:

- Static schema evolution has to be performed when artifact and process descriptions of stored (but currently inactive) processes are modified. This is a general problem of static schema evolution for which a solid body of research has been made. The problem can be reduced in general to the definition of correct instance conversion operations and the application of these operations to the appropriate data instances. Such conversion operations can be defined naturally, as methods of a PCDE meta-schema.
- Dynamic schema evolution has to be performed when the process and data schema of currently active processes have to be modified. Such modifications should not be considered as exceptional situations but rather as the rule in long-running processes. The problems concerning such schema evolutions are much more complicated and involve the identification of permissible modifications which as a rule will depend on the state of the active process and on the constraints defined for it. The abstraction level at which such modifications can be adequately described is also a matter of dispute.

Process enactment. Finally, most PCDEs rely, at least partially, on a process execution engine for the enactment of their processes. The most common approach in the surveyed systems is to persistently store part of the process description in the DBMS/OMS. The execution of the process is undertaken by the process enactment engine which is either a DBMS application or some special purpose language interpreter. The interaction between these two components depends on the concrete system. If a DBMS is used and the process model is completely described by the DDL, the enactment engine can be a DBMS application (client) using the DBMS DML and programming interface. In this case, the cooperation between the components can make use of the full DBMS functionality for transaction management, security, etc. If an OMS just provides persistent storage of the process description, then usually the process execution engine is an interpreter external to the DBMS. In this case, the process description has to be retrieved into main memory where it is then interpreted. This approach is insecure and may potentially create a lot of communication overhead between the OMS and the interpreter.

During process enactment, the occurrence of relevant events (such as an update operation to process data) must be communicated to the process enactment engine. There are three possible integration levels of the DBMS and the process enactment engine:

- If the DBMS has no active capabilities, event signalling has to be done by applications periodically polling the database state or by the event triggering instance. The polling approach tends to be inefficient, imposing significant processing overhead. Furthermore some kind of reliable communication channel has to be set up between the triggering instance and the process enactment engine.
- If simple event-triggers or ECA-rules with simple events can be defined in the database, such events can be signalled by the DBMS itself. The process enactment engine, however,

still has to somehow keep track of composite event occurrences in order to recognize more complex situations. This approach has the disadvantage of limiting the transaction functionality that can be made available from the DBMS.

- In the most powerful approach, enough active capabilities are provided by the DBMS to allow a full modeling and management of complex events and rules representing the software process. In that case, the DBMS can be used as a process engine, where process execution is tightly integrated with the provided transaction mechanisms. Additionally, the disciplined evolution of the process model can be supported and enforced by the DBMS [e.g. 57] if it provides adequate rulebase evolution capabilities. At this moment, none of the systems surveyed has adequate functionality for such a configuration to be used. Research is done in this direction, for example in the SEAMAN project [103].

Triggers are used in most approaches surveyed above. However, triggers are either implemented *on top of* the DBMS (e.g. in EPOS) or are in a rather simple form provided by the DBMS (e.g., Adele, NAOS in SPADE). What is needed is a more powerful abstraction than triggers e.g., an event management and dispatching system as proposed in [62]. Since powerful active, object-oriented DBMS-prototypes are now available, it is an interesting yet open question whether they provide such an abstraction, and whether PCDEs can benefit from this enhanced DBMS-functionality. PCDEs could use this functionality for tasks such as consistency maintenance, process model enactment, etc.; the potential benefits include reduced communication overhead among environment components, since complex reactive behavior is implemented within the DBMS, increased robustness of process control, and a leaner add-on layer on top of the DBMS. It still has to be proven, however, that this kind of reactive behavior (rule definition language and ECA-rule execution semantics) is appropriate for PCDEs.

Integration and Mediation Architectures for PCDEs. We conclude the list of open issues with some questions concerning the software architecture of PCDEs and DBMSs.

PCDEs typically come as a framework that is customized for the concrete project or development environment needed, requires DBMS-functionality, and includes a collection of external, heterogeneous tools. In order to combine the heterogeneous components into a coherent whole, a systematic approach towards mediation and integration is crucial. We thus need an *integration and mediation architecture* for PCDEs (e.g., comparable to OMG's CORBA [19]). Especially, it is still an open issue which role DBMSs may play in such an architecture. For instance, some parts of the DBMS-functionality might in turn need to be customizable (e.g., transaction management), but still must be integratable into the PCDE-framework.

Moreover, given that we have customizable PCDEs and DBMSs together with an integration architecture, a crucial question is whether the current heavyweight DBMSs are still useful. Alternatively, the integration architecture would define a set of standard services (as done in CORBA) for persistence, transactions, relationships etc. which would then be used for a concrete PCDE. The "classic" DBMS as a large and central component would vanish, specialized components would implement the actually required services, and unused functionality would not burden the overall system.

5 Summary and Outlook

This paper surveyed existing process-centered software development environments (PCDEs), concentrating on the integration aspects with their components that provide data and process management functionality. The definition of the domain of interest was followed by the formulation of a set of functional and architectural requirements for PCDE repositories. From the survey of a series of research prototypes and commercial systems it is clear that the current state of the art (especially with respect to integration and architectural issues) is still not satisfactory. Finally a set of important issues was identified which we feel belong among the most promising directions for future work.

6 Acknowledgments

We gratefully acknowledge the comments and ideas contributed by Klaus R. Dittrich. We also would like to thank Rolf Amort from Softlab AG, Alfred Bürger from VERILOG Deutschland GmbH, Roland Dietiker from Hewlett-Packard (Schweiz) AG, and Peter Pausinger from ATRIA Software GmbH for the information kindly provided about the products of their respective companies.

7 References

1. V. Ambriola, P. Ciancarini, C. Montagero, Software Process Enactment in Oikos. In *Proc. of the 4th ACM SIGSOFT Symposium on Software Development Environments*, Irvine, CA, 1990.
2. V. Ambriola, G.A. Cignoni, C. Montagero, The Oikos Services for Object Management in the Software Process. In B.C. Warboys (ed.), *Software Process Technology, Proc. of the 3rd European Workshop*, Springer 1994.
3. V. Ambriola, G.A. Cignoni, Expo 2.0: User's Guide and Reference Manual. Technical Report 6/100, Dipartimento di Informatica, Università di Pisa, January 1994.
4. V. Ambriola, C. Montagero, Modeling the Software Development Process. In V. Ambriola, C. Tortora (eds.), *Advances in Software Engineering and Knowledge Engineering*, World Scientific Publishing Company, Singapore, 1993.
5. Atria Software Inc., *ClearCase Product Summary*, 1995.
6. F. Bancilhon, W. Kim, H.F. Korth, A Model of CAD-Transactions. *Proc. 11th Intl. Conf. on Very Large Data Bases*, Stockholm, Sweden, August 1985.
7. S. Bandinelli, A. Fuggetta, C. Ghezzi, Software Process Model Evolution in the SPADE Enviornment. *IEEE Transactions on Software Engineering*, 19:12, December 1993.
8. S. Bandinelli, M. Braga, A. Fuggetta, L. Lavazza, The Architecture of the SPADE-1 Process-Centered SEE. In [107].
9. S. Bandinelli, A. Fuggetta, C. Ghezzi, L. Lavazza, SPADE: An Environment for Software Process Analysis, Design, and Enactment. In [54].
10. J. Banerjee, W. Kim, Semantics and Implementation of Schema Evolution in Object-Oriented Database Systems. In U. Dayal, I. Traiger (eds.) *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, San Francisco, CA, May 1987. SIGMOD Record 16:3, 1987.
11. N.S. Barghouti, G.E. Kaiser, Concurrency Control in Advanced Database Applications. *ACM Comp. Surveys,* 23:3, 1991.
12. N. S. Barghouti, Supporting Cooperation in the MARVEL Process-Centered SDE. In H.

Weber (ed.) *Proc. of the 5th ACM SIGSOFT Symposium on Software Development Environments*, Virginia 1992.

13. N.S. Barghouti, G.E. Kaiser, Scaling Up Rule-Based Software Development Environments. In A. van Lamsweerde, A. Fugetta (eds.) *Proc. of the 3rd European Software Engineering Conference*, Milan 1991.
14. N. Belkhatir, J. Estublier, W.L. Melo, Adele 2: A Support to Large Software Development Process. *Proc. 1st Intl. Conf. on the Software Process*, Redondo Beach, California, 1991.
15. N. Belkhatir, J. Estublier, W.L. Melo, Software Process Model and Work Space Control in the Adele System. *Proc. 2nd Intl. Conference on the Software Process*, Berlin, February 1993.
16. N. Belkhatir, J. Estublier, W.L. Melo, ADELE-TEMPO: An Environment to Support Process Modeling and Enaction. In [54].
17. N. Belkhatir, W.L. Melo, Evolving Software Processes by Tailoring the Behavior of Software Objects. *Proc. Intl. Conf. on Software Maintenance,* Victoria, Canada, September 1994.
18. N. Belkhatir, W.L. Melo, The Need to a Cooperative Model: The Adele/Tempo Experience. *Proc. 9th Intl. Software Process Workshop*, Airlie, VA, October 1994.
19. R. Ben-Natan, *CORBA. A Guide to Common Object Request Broker Architecture*, Computing McGraw-Hill, 1995.
20. I.Z. Ben-Shaul, G.E. Kaiser, G.T. Heineman, An Architecture for Multi-User Software Development Environments. In H. Weber (ed.) *Proc. of the 5th ACM SIGSOFT Symposium on Software Development Environments*, Virginia 1992.
21. I.Z. Ben-Shaul, G.E. Kaiser, A paradigm for decentralized process modeling and its realization in the Oz environment. *Proc. 16th Intl. Conference on Software Engineering*, IEEE Computer Society Press, 1994.
22. I.Z. Ben-Shaul, G.E. Kaiser, Process Support for Synchronous Groupware Activities. Columbia University, CUCS-002-95, January 1995.
23. P.A. Bernstein, Database System Support for Software Engineering -- An Extended Abstract. *Proc. 9th Intl. Conf. on Software Engineering*, Monterey, CA, 1987.
24. P.A. Bernstein, V. Hadzilacos, N. Goodman, *Concurrency Control and Recovery in Database Systems*. Addison Wesley, 1987.
25. A.W. Brown, P. H. Feiler, K. C. Wallnau, Past and Future Models of CASE Integration. *Proc. Intl. Conf. Computer Aided Software Engineering*, Montreal, 1992.
26. M. J. Carey, D. J. DeWitt, G. Graefe, D. M. Haight, J. E. Richardson, D. T. Schuh, E. J. Shekita, S. Vandenberg, The EXODUS Extensible DBMS Project: An Overview. In D. Maier, S. Zdonik (eds.), *Readings on Object-Oriented Database Systems*, Morgan Kaufmann, San Mateo, CA, 1990.
27. G. Canals, N. Boudjlida, J.-C. Derniame, C. Godart, J. Lonchamp, ALF: A Framework for Building Process-Centred Software Engineering Environments. In [54].
28. P.P. Chen, The Entity-Relationship Model - Towards a Unified View of Data. *ACM Transactions on Database Systems* 1:1, 1976.
29. D. Chimenti, R. Gamboa, R. Krishnamurthy, S. Naqvi, S. Tsur, C. Zaniolo, The LDL

System Prototype. In *IEEE Transactions on Knowledge and Data Engineering*, 2:1, March 1990.

30. G. Chroust, H. Goldmann, O. Gschwandter, The Role of Work Management in Application Development. *IBM Systems Journal*, 29:2, 1990.
31. G. Clow, E. Ploedereder, Issues in Designing Object Management Systems. *Software Engineering Environments, Proc. of the Intl. Workshop on Environments*, Springer 1989.
32. E. Codd, A Relational Model for Large Shared Data Banks. *Communications of the ACM* 13:6, 1970.
33. C. Collet, T. Coupaye, T. Svensen, NAOS Efficient and modular reactive capabilities in an Object-Oriented Database System. *Proc. 20th Intl. Conf. on Very Large Data Bases*, Santiago-Chile, September 1994.
34. R. Conradi, T.M. Didriksen, B. Gulla, H. Eidnes, et.al., Design of the kernel EPOS Software Engineering Environment. In M. Madhavji, W. Schaefer, H. Weber (eds.), *Proceedings 1st Intl. Conf. on System Deevelopment Environments and Factories*, Berlin 1989.
35. R. Conradi, M. Hagaseth, J.-O. Larsen, M.N. Nguyên, B. Munch, P.H. Westby, W. Zhu, M.L. Jaccheri, C. Lu, EPOS: Object-Oriented Cooperative Process modeling. In [54].
36. R. Conradi, M.L. Jaccheri, C. Mazzi, M.N. Nguyên, A. Aarsten, Design, Use and Implementation of SPELL, a language for Software Process modeling and Evolution. In [40].
37. P. Constantopoulos, M. Jarke, J. Mylopoulos, Y. Vassiliou, The Software Information Base: A Server for Reuse. *VLDB Journal,* 4:1, 1995.
38. J. Courant, SoftBench Message Connector: Customizing Software Development Tool Interactions. *Hewlett-Packard Journal*, June 1994.
39. U. Dayal, M. Hsu, R. Ladin: Organizing Long-Running Activities with Triggers and Transactions. *Proc. ACM-SIGMOD Intl. Conf. on Management of Data*, Atlantic City, May 1990.
40. J.C. Derniame (ed.), *Proc. 2nd European Workshop on Software Process Technology (EWSPT'92)*, Trondheim, Norway, September 1992.
41. J. Diamant, H. Davidson, G. Thunquest, Process modeling in HP SoftBench. *Proc. 6th International Software Process Workshop*, Hakodate, Japan, October 1990.
42. T.M. Didriksen, A. Lie, R. Conradi, E.-A. Karlsson, S.O. Hallsteinsen, P. Holager, Change Oriented Versioning. *Proc. 2nd Intl. Conf. on Software Engineering*, Warwick, September 1989.
43. K.R. Dittrich, W. Gotthard, P.C. Lockemann: DAMOKLES - A Database System for Software Engineering Environments. In R. Conradi, T.M. Didriksen, D.H. Wanvik (eds): *Advanced Programming Environments*. LNCS 244, Springer 1986.
44. K.R. Dittrich, Preface. In [45].
45. K.R. Dittrich (ed.), *Advances in Object-Oriented Database Systems, Proc. of 2nd Intl. Workshop*, Bad Munster am Stein-Ebenburg, LNCS 334, Springer 1988.
46. K.R. Dittrich, S. Gatziu, A. Geppert (eds), The Active Database Management System Manifesto: A Rulebase for ADBMS Features. In T. Sellis (ed.) *Proc. 2nd Workshop on Rules in Databases*, Athens, Greece, September 1995. Lecture Notes in Computer

Science, Springer 1995.

47. A. Earl, Principles of a Reference Model for Frameworks of Computer Aided Software Engineering Environments. In F. Long (ed.), *Software Engineering Environments*, Springer LNCS 467, 1989.
48. A.K. Elmagarmid, *Database Transaction Models For Advanced Applications*. Morgan Kaufmann Publishers, 1992.
49. W. Emmerich, W. Schaefer, J. Welsh, Databases for Software Engineering Environments - The Goal has not yet been attained. In [100].
50. J. Estublier, What Process Technology needs from Databases. In [107].
51. J. Estublier, N. Belkhatir, M. Ahmed-Nacer, W.L. Melo, Process centered SEE and Adele. In G. Forte, N.H. Madhavji, H.A. Muller (eds.), *Proc. 5th Intl. Workshop on Computer Aided Software Engineering*, Montreal, CDN, July 1992.
52. European Computer Manufacturers Association, National Institute of Standards and technology, Reference Model for Frameworks of Software Engineering Environments. Technical Report, ECMA TR/93/55, August 1993.
53. P. Feiler, W.S. Humphrey, Software Process Development and Enactment: Concepts and Definitions.
54. A. Finkelstein, J. Kramer, B. Nuseibeh, *Software Process Modeling and Technology.* Research Studies Press, Taunton, UK, 1994.
55. B.D. Fromme, HP Encapsulator. *Hewlett-Packard Journal*, June 1990.
56. D. Georgakopoulos, M. Hornick, A. Sheth, An Overview of Workflow Management: From Process modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, Kluwer Academic Publishers, 1994.
57. A. Geppert, S. Gatziu, K.R. Dittrich, Rulebase Evolution in Active Object-Oriented Database Systems: Adapting the Past to Future Needs. Technical Report 95.13, Department of Computer Science, University of Zurich, February 1995.
58. A. Geppert, M. Kradolfer, D. Tombros, Realization of Cooperative Agents Using an Active Object-oriented Database Management System. In T. Sellis (ed.), *Proc. 2nd Intl. Workshop on Rules in Database Systems*, Athens, Greece, September 1995, LNCS Springer.
59. M. Glinz, J. Ludewig, SEED - A DBMS for Software Engineering Applications Based on the Entity-Relationship Approach. *Proc. 2nd IEEE Intl. Conf. on Data Engineering*, Los Angeles, 1986.
60. J.N. Gray: Notes on Database Operating Systems. In R. Bayer, R.M. Graham, G. Seegmueller (eds): Operating Systems. An Advanced Course.. LNCS 60, Springer 1978.
61. R.L. Haskin, R.A. Lorie, On Extending the Functions of a Relational Database System. *Proc. ACM-SIGMOD Intl. Conf. on Management of Data*, Orlando, Florida, June 1982.
62. D. Heimbinger, Experiences with an Object Manager for a Process-Centered Environment. In *Proc. of the 18th VLDB Conference*, Vancouver, CDN, 1992.
63. S.E. Hudson, R. King, The Efficient Support of Functionally-Defined Data in Cactis. In K.R. Dittrich, U. Dayal, A.P. Buchmann (eds), *On Object-Oriented Database Systems*. Topics in Information Systems, , Springer 1991.

64. M.L. Jaccheri, R. Conradi, Techniques for Process Model Evolution in EPOS. In *IEEE Transactions on Software Engineering*, 19:12, December 1993.
65. L. Jaccheri, J.-O. Larsen, R. Conradi, Software Process modeling and Evolution in EPOS. In *Proc. 4th Int. Conf. on Software Engineering and Knowledge Engineering*, Capri, 1992.
66. G.E. Kaiser, N.S. Barghouti, P.H. Feiler, R.W. Schwanke, Database Support for Knowledge Based Engineering Environment, *IEEE Expert*, 3:3, 1988.
67. G.E. Kaiser, C. Pu, Dynamic Restructuring of Transactions. In [48].
68. R.H. Katz, Towards a Unified Framework for Version modeling in Engineering Databases. *ACM Computing Survey,* 22:4, 1990.
69. N. Kiesel, A. Schürr, B. Westfechtel, Design and Evaluation of GRAS, a Graph-Oriented Database System for Engineering Applications. In H.-Y. Lee, T.F. Reid, S. Jarzabek (eds.), *Proc. 6th Intl. Workshop on Computer-Aided Software Engineering*, Singapore, IEEE Computer Society Press, 1993.
70. W. Kim, Object-Oriented Database Systems: Promises, Reality, and Future. In [71].
71. W. Kim (ed.), *Modern Database Systems*. Addison Welsey, New York, 1995.
72. H.F. Korth, G.D. Speegle, Formal Model of Correctness Without Serializability. In *Proc. ACM-SIGMOD Intl. Conf. on Management of Data*, Chicago, Illinois, June 1988.
73. A.M. Kotz, K.R. Dittrich, J.A. Mülle, Supporting Semantic Rules by a Generalized Event/Trigger Mechanism. *Proc. Intl. Conf. on Extending Database Technology (EDBT)*, Venice, Italy, March 1988. LNCS 303, Springer 1988.
74. B. Krishnamurthy, N.S. Barghouti, Provence: A Process Visualisation and Enactment Environment. In [100].
75. B. Krishnamurthy, D.S. Rosenblum, Yeast: A General Purpose Event-Action System, *IEEE Transactions on Software Engineering*, 21:10, 1995.
76. F. Leymann, W. Altenhuber, Managing Business Processes as an Information Resource. *IBM Systems Journal,* 33:2, 1994.
77. A. Lie, R. Conradi, T.M. Didriksen, E.-A. Karlsson, Change Oriented Versioning in a Software Engineering Database. In W.F. Tichy (ed.), *Proc. 2nd Intl. Workshop on Software Configuration Management*, Princeton, USA, October 1989, ACM SIGSOFT Software Engineering Notes, 14:7, November 1989.
78. C. Liu, R. Conradi, Automatic Replanning of Task Networks for PM Evolution in EPOS. In [100].
79. P.C. Lockemann, K.R. Dittrich, M. Adams, M. Bever, B. Ferkinghoff, W. Gotthard, A.M. Kotz, R.-P. Liedtke, B. Lüke, J.A. Mülle: Database Requirements of Engineering Applications. An Analysis. University of Karlsruhe, Internal Report 12/85.
80. J. Lonchamp, A Structured Conceptual and Terminological Framework for Software Process Engineering.
81. P. Loucopoulos, R. Zicari, (eds.), *Conceptual modeling, Databases, and CASE*. John Wiley, New York, 1992.
82. D.R. McCarthy, U. Dayal, The Architecture of an Active Data base Management System. *Proc. 19th ACM SIGMOD Conf. on the Management of Data*, Portland, Oregon, June 1989.

83. N.H. Madhavji, Environment Evolution: The Prism Model of Changes. *IEEE Transactions on Software Engineering*, 18:5, 1992.
84. J. McDermid (ed.), *Software Engineer's Reference Book*, Butterworth-Heinemann, Oxford 1991.
85. W.L. Melo, N. Belkhatir, TEMPO: A Support for the Modeling of Objects with Dynamic Behavior. To appear in A. Verbraeck, H.G. SOl, P.W.G. Bots (eds.), *Dynamic Modelling and Information Systems*. North-Holland, Elsevier Science Publishers, 1994.
86. V.J. Mercurio, B.F. Meyers, A.M. Nisbet, G. Radin, AD/Cycle Strategy and Architecture. *IBM Systems Journal*, 29:2, 1990.
87. C. Montangero, V. Ambriola, Oikos: Constructing Process-centred SDEs. In [54].
88. J. Mylopoulos, A. Borgida, M. Jarke, M. Koubarakis, Telos: Representing Knowledge About Information Systems. *ACM Transactions on Information Systems* 8:4, 1990.
89. M.H. Nodine, S.B. Zdonik, Cooperative Transaction Hierarchies: A Transaction Model to Support Design Applications. In *Proc. 16th Intl. Conf. on Very Large Data Bases*, Brisbane, Australia, August 1990.
90. E. Onuegbe, Database Management System Requirements for Software Engineering Environments. *Proc. 3rd Intl. IEEE Conf. on Data Engineering*, 1987.
91. F. Oquendo, J.-D. Zucker, P. Griffiths, A Meta-CASE Environment for Software Process-centred CASE Environments. In P. Loucopoulos (ed.), *Proc. Advanced Information Systems Engineering, 4th Intl. Conf. CAiSE 92*, Manchester, UK, May 1992.
92. The O_2 Team, A Technical Review of the O_2 System. In [81].
93. B. Peuschel, W. Schäfer, S. Wolf, A Consistency Model for Team Cooperation. In *Proc. of the 7th Intl. Software Process Workshop*, Yountville, CA. 1991.
94. B. Peuschel, W. Schäfer, Concepts and Implementation of a Rule-based Process Engine. *Proc. of 14th International Conference on Software Engineering*, Melbourne, 1992.
95. S. Rehm, T. Raupp, M. Ranft, R. Längle, M. Härtig, W. Gotthard, K.R. Dittrich, K. Abramowicz: Support for Design Processes in a Structurally Object-Oriented Database System. In [45].
96. J.E. Richardson, M.J. Carey, Programming Constructs for Database System Implementation in EXODUS. *Proc. ACM SIGMOD Intl. Conf. on Management of Data,* 1987.
97. J. Rumbaugh. Relations as Semantic Constructs in an Object-Oriented Language. *Proc. ACM SIGPLAN Conf. on Object-Oriented Programming Systems, Languages and Applications*, Kissimmee, Florida, October 1987.
98. J.M. Segawa, Repository Manager Technology. *IBM Systems Journal*, 29:2, 1990.
99. Softlab, Maestro II IntegrateIT Software Engineering Platform Produkt Profile, 1995.
100. I. Sommerville, M. Paul (eds.), *Proc. 4th European Software Engineering Conference*, Garmisch-Partenkirchen, Springer 1993.
101. S.M. Sutton, D. Heimbinger, L.J. Osterweil, APPL/A: A Language for Software Process Programming. University of Colorado Technical Report CU-CS-727-94.
102. R.N. Taylor, et.al., Foundations for the Arcadia Environment Architecture. *Proc. of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, ACM SIGSOFT, 13:5, 1988.

103. D. Tombros, A. Geppert, K.R. Dittrich, Brokers and Services: Constructs for the Design and Implementation of Process-Oriented Environments. Technical Report 95.26, Computer Science Department, University of Zurich, 1995.

104. Transtar Software Inc., Emeraude PCTE, The Development Environment Builder.

105. VERILOG, ADELE Technical Presentation, July 1992.

106. L. Wakeman, J. Jowett, *PCTE The standard for open repositories*. Prentice Hall, 1993.

107. B. C. Warboys (ed.), *Software Process Technology, Proc. 3rd European Workshop*, Villard de Lans, France, February 1994, Springer, LNCS 772.

108. J.C. Wileden, A.L. Wolf, C.D. Fisher, P.L. Tarr, PGRAPHITE: An Experiment in Persistent Typed Object Management. *Proc. Symposium on Practical Software Development Environments*, ACM SIGSOFT, 13:5, 1988.

109. R. Zicari, A Framework for Schema Updates in an Object-Oriented Database System. In F. Bancilhon, C. Delobel and P. Kanellakis (eds.), *Building an Object-Oriented Database System: The story of O_2*, Morgan Kaufmann, 1992.

	DBMS/OMS	data model	active functionality	used for	dynamic schema evolution	extended transactions	CM/ versions	multi-language interoperability	distribution
Adele/ TEMPO	proprietary	extended ER	EA-triggers	artifacts/ PMs	-	-	yes	-	-
ALF	PCTE	extended ER	EA-triggers	artifacts/ PMs/ tools	-	-	yes	-	client/server
Arcadia	Triton	full OO	EA-triggers	artifacts/ PMs	yes	-	-	yes	-
EPOS	proprietary	object-relation	-	PMs	(yes)	long/nested	yes	-	-
Marvel/ Pro-vence	proprietary	structural OO	-	artifacts/ part of PM	-	nested	-	-	client/server
Merlin	GRAS	attribute graphs	-	PMs	-	checkin/ checkout	yes	-	-
OIKOS	proprietary	structural OO	-	PMs	-	checkin/ checkout	yes	-	-
SPADE	NAOS (O_2)	full OO	ECA-rules	artifacts/ PMs/ tools	yes	checkin/ checkout	yes	yes	client/server

Table 2: Overview of DBMS/OMS functionality in the surveyed non-commercial PCDEs