SEAMAN: Implementing Process-Centered Software Development Environments on Top of an Active Database Management System

Dimitrios Tombros, Andreas Geppert, Klaus R. Dittrich

{tombros,geppert,dittrich}@ifi.unizh.ch

Technical Report 95.03

Institut für Informatik, Universität Zürich
Winterthurerstr. 190, 8057 Zürich
Switzerland

Abstract. The goal of the SEAMAN[1] project is to provide a framework for the efficient construction of customized process-centered software development environments (PCDEs). SEAMAN consists of a meta-level where the architecture of both the development environments and their process models can be described in a uniform way, and an operational level in which instantiated PCDEs operate. The formalisms we use in the meta-level are simple yet powerful and can be easily mapped onto mechanisms provided by an active object-oriented database management system (aDBMS) serving as the underlying implementation platform. In this paper we present the architecture of SEAMAN and the concepts used for the definition of software development environments and their process models. Finally we describe how an aDBMS can be effectively used to implement this framework.

1. **S**oftware **E**ngineering with **A**ctive **M**ech**AN**isms

1 Introduction and Motivation

The development of software systems is a complex task which involves the coordination of various agents - human and computational - performing various more or less related activities and operating on a large variety of artifacts among which complex dependencies exist. *Process-centered development environments* (PCDEs) have been proposed to support this task [10]. An important consideration for PCDEs is that they must be flexible enough to be used in various projects without imposing limitations on issues such as the process used, guidelines to be followed, and constraints to be maintained among various deliverables.

In PCDEs software artifacts are created and manipulated by services requested by the environment users. It is still an open question how to integrate these services into a coherent environment in a flexible way, particularly if services are provided by tools developed independently from each other [5]. In other words, we face the problem of the *appropriate software architecture of integrated software development environments*.

In the effort to cover all possible usage scenarios of PCDEs, usually a base system is defined which provides a large set of potentially useful services to its users, as for example in PCTE [18, 19]. This creates large systems which are difficult to overview and maintain and causes substantial overhead when only a subset of the provided functionality (depending on factors such as the PM or the development context) is actually needed. Thus a further problem is *how to provide a system offering the exact set of services needed in a specific development context*.

A software development project has to proceed according to predefined guidelines. In PCDEs *process models* (PMs) describe these guidelines formally, allowing processes to be enacted. A process engine executes the process program which coordinates the developers involved and automates certain tasks such as the invocation of external tools. A repository is used for storing information about the development process and the artifacts generated (example projects that have followed this general approach are Adele [8], EPOS [14], Marvel [4], Merlin [20], and SPADE [2]). The problem with this approach is that the adaptability of the system is limited by the possibilities offered by the PM within a relatively fixed architectural framework. In practice the structural and behavioral aspects of an architecture are interrelated and should be coordinated. In this respect, a third problem is *how to specify PMs so that they integrate well with the architecture of the PCDE, including the underlying repository.*

A way to achieve the required flexibility and power is by simplifying the task of *developing customized PCDEs*, optimized for use in a specific project or a group of similar projects. Thus depending on the specific requirements imposed by an organizational or development context, PCDEs satisfying exactly these requirements can be constructed and used. This approach is advantageous both for large development organizations where a large variety of development projects must be supported, and for PCDE vendors which can provide different products according to customer needs.

Subsequently, we describe the SEAMAN approach, that as we shall see supports both the construction of customized PCDEs and the modelling of the enacted processes in a coherent manner. The SEAMAN *meta-level* has a service-oriented view of PCDE architectures which has also been used in standardization efforts [9, 17]. The meta-level provides the framework for the definition, initialization and operation of specific PCDEs. Its prime constituents are so called *brokers* used to model the architecture of PCDEs. These are components responsible for providing a set of *services* to their clients, thus achieving a *service-oriented integration* [5]. Brokers allow the expression of structural, functional, and behavioral aspects of PCDEs, component modularization and cooperation, and the integration of existing tools by using interface brokers. The *same* concepts are used for software process modelling as a development process is described by the behavior of the brokers participating in it.

In most PCDEs a database management system (DBMS) serves as an integrating component based on a common schema describing development artifacts. These artifacts can be stored and manipulated in the DBMS. We extend the integrating role of the DBMS by implementing brokers and services (i.e., a PCDE instance) on top of an *active object-oriented database management system* (aDBMS). This allows us to implement both PCDEs and their products in a uniform way. We also use the DBMS to provide further functionality such as concurrent access to data, automatic recovery, etc.

The contributions of SEAMAN are thus threefold. It introduces powerful formalisms to describe the architecture and behavior of PCDEs in a simple and coherent manner, thereby integrating system components and development processes. This is in contrast to other approaches where structure and behavior are described in conceptually different ways (e.g. objects are used for structural description and a rule based formalism is used for process description). In addition, it allows a flexible and efficient construction of customized PCDEs providing the exact functionality needed for a specific development context. Finally, the use of an aDBMS as a standard base component provides an integrating layer with a powerful set of base services.

The organization of the paper is as follows: In Section 2 we present the concepts we use to define the architecture of PCDEs. In Section 3 we discuss the architecture of the construction framework SEAMAN and give an example of the use of the presented concepts. In Section 4 we describe how an aDBMS is used to support various aspects of our work. We then survey related work in Section 5 and summarizing in Section 6 we outline our future work.

2 Service-Oriented Architecture

2.1 Architectural Reference Models

Services describe the functionality that has to be offered by an SDE. The advantage of a service-oriented approach is that it provides a conceptual view of the required functionality without being bound by any implementation. According to recently proposed reference models [9, 17], an SDE is composed of different service groupings:

- *Support* services, which focus on tasks common to all users independently from domain, activity or life-cycle phase (e.g. document processing, user communication).
- *Project management*, *technical management* and *technical engineering* services (e.g. design and coding, project scheduling and tracking).
- *Framework* services, which provide the infrastructure for the development process (e.g. object management, policy enforcement).

The first two kinds of services are services available to the environment end-users, while the third provide the infrastructure needed to realize the end-user services.

The services provided within the SDE are realized by various components. Usually a component may provide more than one service through its interface to human users or other components. Based on this description we can define four kinds of components:

- Software development components (i.e. the tools used for analysis, design, coding, testing, etc. of the produced artifacts).
- Environment specification components (i.e the meta-environment).
- Data and meta-data management components (i.e. DBMS facilities).
- Environment and process control components (e.g. a process engine).

Subsequently we introduce architectural concepts with which we describe services as well as the components that provide them.

2.2 The Broker/Services Model for Modeling PCDEs

In SEAMAN, we want to support the customization of PCDEs for concrete projects and the flexible definition of PMs in terms of the PCDE-architecture. In order to model PCDE-architectures we have to allow:

- the modelling of services to be offered by the PCDE,
- the definition of the reactive entities (managers) in the resulting PCDE,
- the assignment of components to services,
- the integration of existing tools into the PCDE,
- the modeling of software processes,
- the enactment of these processes.

SEAMAN uses an *extended broker/services architecture model* [12] in order to fulfill these requirements (see Fig. 1 for a simplified meta-schema of the model)

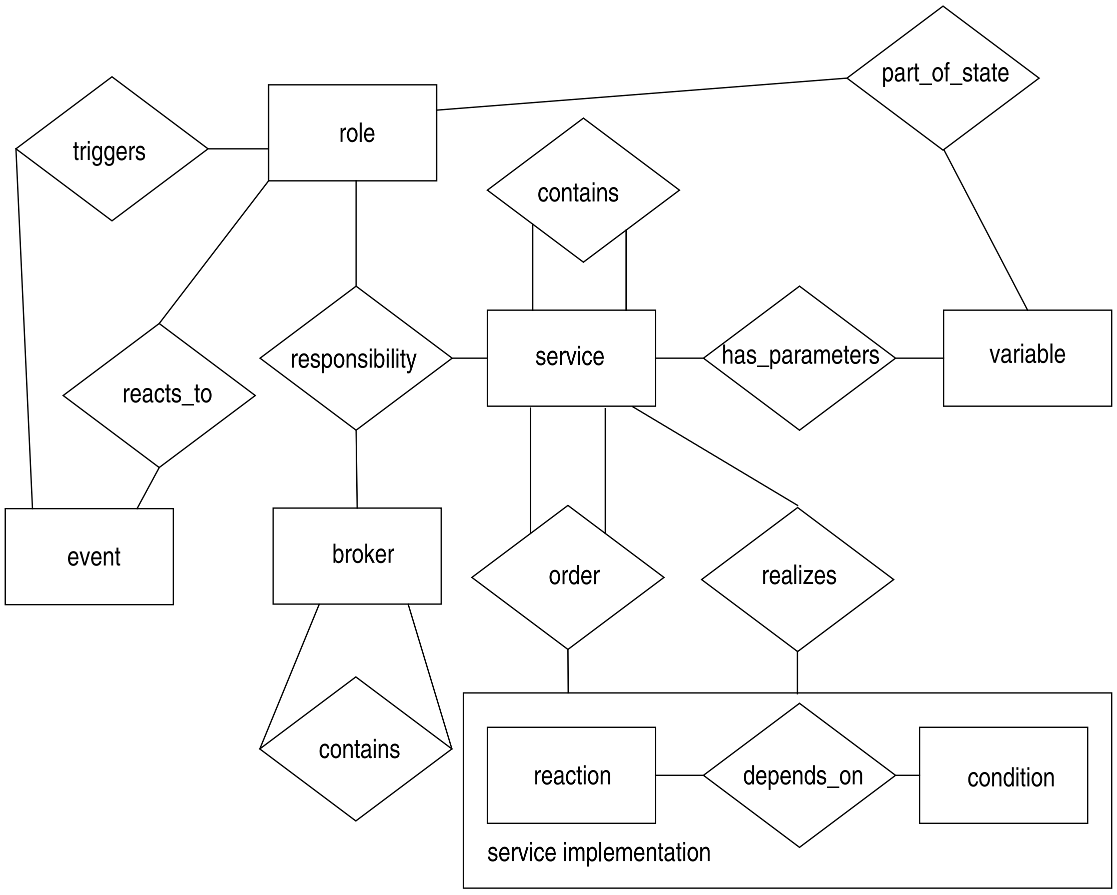

Figure 1: Meta-schema of the PCDE description constructs

This model provides the facilities to describe the structure of a system, its behavior, and architectural constraints. It uses an object-oriented approach to system construction, extended with the possibility to define reactive behavior of the objects[2]. Its main concepts are *brokers*, *services*, and *responsibilities* defined subsequently. A PCDE-architecture is defined as a collection of brokers operating in various roles, being responsible for providing services and being able to monitor complex events and react according to predefined ways. The services provided by brokers can refer to the manipulation of development data

2. The term “reactive behavior” describes the capability of objects to autonomously execute various actions in response to predefined simple or composite events (not just method calls).

or to the control and coordination of system components. These integrated services are thus viewed as a virtual machine encoding software process elements [5] while at the same time describing the system architecture.

Subsequently, we describe in more detail the concepts we use for the modeling of PCDEs. An example of their use is presented in section 3.2.

2.2.1 Services

Services model the functionality of system components. They allow a service-oriented view of the environment abstracting from concrete implementations. A specific service is provided by one or more brokers (see below) and can be requested by various client brokers. It is specified by its signature and by the reactions of brokers responsible for its provision (i.e. which react to the service request). The service signature consists of the service name, its parameters, the possible replies and exceptions its request may cause. We decompose relatively high-level service descriptions (service groups) in the reference models to low-level definition of services in the broker/services model we are using. Thus, at the lowest level services in our model correspond to the operations of services described in [9, 17].

2.2.2 Brokers

Brokers represent “reactive” system components, responsible for the provision of end-user and framework services. We distinguish between three kinds of brokers: *internal*, *external*, and *interface* brokers. Internal and interface brokers are described by their state, the services they are responsible for providing, and their reaction to predefined events (see below). The state of a broker consists of typed instance variables which can be either *sub-brokers* or passive objects. Brokers which have sub-brokers are called *composite*. Sub-brokers of internal and interface brokers can only be internal or interface brokers themselves. Broker-specific methods can be defined which may take typed formal parameters and possibly return a typed result.

Internal brokers represent system components. An example is an object management system. Interface brokers implement the behavior of system components interacting with human beings and external tools. A typical example is a session manager representing the interaction of a user with the system. External brokers are blackboxes for which the internal state, methods and reactions do not have to (but can) be defined. They model the behavior of human users and external tools, and can request services from other external brokers as well as from interface brokers.

Brokers and their sub-brokers form a component hierarchy with predefined visibility of services. Brokers can directly request services from their siblings, but sub-brokers communicate with their parent's environment through their parent brokers. This means that we need different service request operations to forward requests to siblings within the same broker, to children of a broker or to siblings of the parent broker.

2.2.3 Roles

Roles specify the responsibilities of brokers in various situational and organizational contexts. To achieve this, a broker definition can contain a set of role specifications. Each role specification consists of state variables, methods, and production rules. There may however be state variables, methods, and production rules (see below) which are role-independent, i.e. common to all roles of a broker. These may be defined outside any particular role and may be used by all the roles of the broker. The concept of roles is needed to model for example the fact that a person, while being the same physical entity, may be both a designer and a reviewer within the same project. For an explanation of why roles are not adequately modelled by using multiple inheritance see [21].

2.2.4 Events

The description of reactive behavior, i.e. of the fact that objects may autonomously react to the occurrence of specific situations, is achieved by using *events*. Such events can occur for example due to a sequence of broker actions within a process, or when specific points in time are reached. In order to describe events occurring during the operation of a PCDE, we distinguish between a number of *event types*. In our architectural model we use *primitive* event types which are a modified subset of those defined in [11]:

- *Service request events* explicitly raised by clients through special request operations (see below). Each service request is accompanied by a list of actual parameters and a synchronization mode, evaluated by the service provider.

- *Value events* are related to the modification of an object value. This allows among others the monitoring of the state of the repository. Such events are defined for update operations on object attributes and take place before or after the operation that updates the value of the object is performed.
- *Method events* are bound to the execution point of a specific method. Their occurrence point is specified as being just before or immediately after (i.e. directly before the method returns to its caller) method execution.
- *Time events* occur when a particular point in time is reached. They are specified either absolutely (by giving a clock-time), as intervals, relatively to another event, or as periodic events.

Composite events are defined by combining primitive events by means of constructors: *conjunction*, *sequence*, *disjunction*, *closure*, and *negation*. For an exact definition of the semantics of these constructors, see [11].

2.2.5 Production Rules

Production rules define the reaction of brokers (eventually within the context of one of their roles) to specified external events of various types. They have a system-wide unique name and consist of an event clause, a condition clause which guards the execution of their action part, and an action part. It is possible that more than one rule reacts to the same event within one broker (role). A partial ordering of rule execution can be defined by using a precedence clause. It is important to note that a precedence order has to be defined in case the action part of a rule affects the condition part of another one, therefore influencing rule execution semantics.

2.2.6 Broker Operations

The production rules of a broker or its role define its behavior in response to an external event. In the action part of these rules, various broker-specific methods or predefined operations may be performed. The predefined operations can be used by all brokers to define service interactions and are described in the following paragraphs.

The `request`, `requestup` and `requestdown` operations generate a service request event. Their parameters are the service name, service parameters, and a request mode (synchronous or asynchronous). A synchronous request implies that the service provider must give a reply before he performs any further action. The difference between the three operations is the scope of their validity. The `request` operation is visible only to siblings of the broker requesting the service. The `requestdown` operation is used to propagate the service request to sub-brokers of a broker executing this operation. Finally the `requestup` operation has the opposite effect and serves for passing a service request to siblings of the parent broker. We note that the last two operations are not exactly symmetric. Suppose for example a broker A has as its siblings brokers B and C and as its children brokers AA and AB. AB is responsible for providing the service S1. If B requests S1 by performing `request(S1)`, then A must perform `requestdown(S1)` to inform its children of the request. Now suppose AB needs a service S2 provided by C. Then once AB performs `requestup(S2)`, A and its peers may catch the service request. Of course this assumes some knowledge on the part of AB about the responsibilities of its siblings.

The `reject` operation can be performed as a reply to a service request when the broker is not able to offer the service, due to its internal state at the moment it caught the request. Its parameters are the rejected service name and an optional status message.

The `reply` operation is used to inform clients about the results of a service request. Its parameters are the name of the requested service and variables which contain the results of the provided service.

The operations `block` and `resume` are used to temporarily suspend and resume the execution of a reaction to an event. Their parameters are the name of the suspended and resumed service, respectively.

The `exception` operation is used to signal the occurrence of a situation with which the system cannot cope through the normal execution flow. Its parameter is the name of an exception. We assume that exceptions are forwarded to successively higher levels of the architecture until an exception handler defined at some level catches this particular exception.

3 The SEAMAN Architecture

Based on the architecture model described in the previous section, we now turn to the architecture of the SEAMAN environment (Fig. 2) which provides the facilities for the construction of PCDEs and their subsequent initialization and operation.

SEAMAN consists of a meta-level and an operational level. At the meta-level, PCDEs and their PMs are defined (in terms of brokers/services), subsequently initialized and managed at run-time. The PCDE-*manager* is responsible for writing and modifying broker definitions and tailoring the PCDE to the needs of specific projects. At this level PCDE architectures and PMs are explicitly represented and can be manipulated, thus introducing a PCDE life-cycle and enabling the administration of various PCDE instances. At the operational level, we encounter PCDE instances used for “real” software production. These environments contain a database and a rulebase implementing the various functional and behavioral aspects of the PCDE which is also used for storage of the created software artifacts. Various external tools can be attached to the environment through interface brokers. In each PCDE a special broker called the *globe* is responsible for coordinating the top-lev-

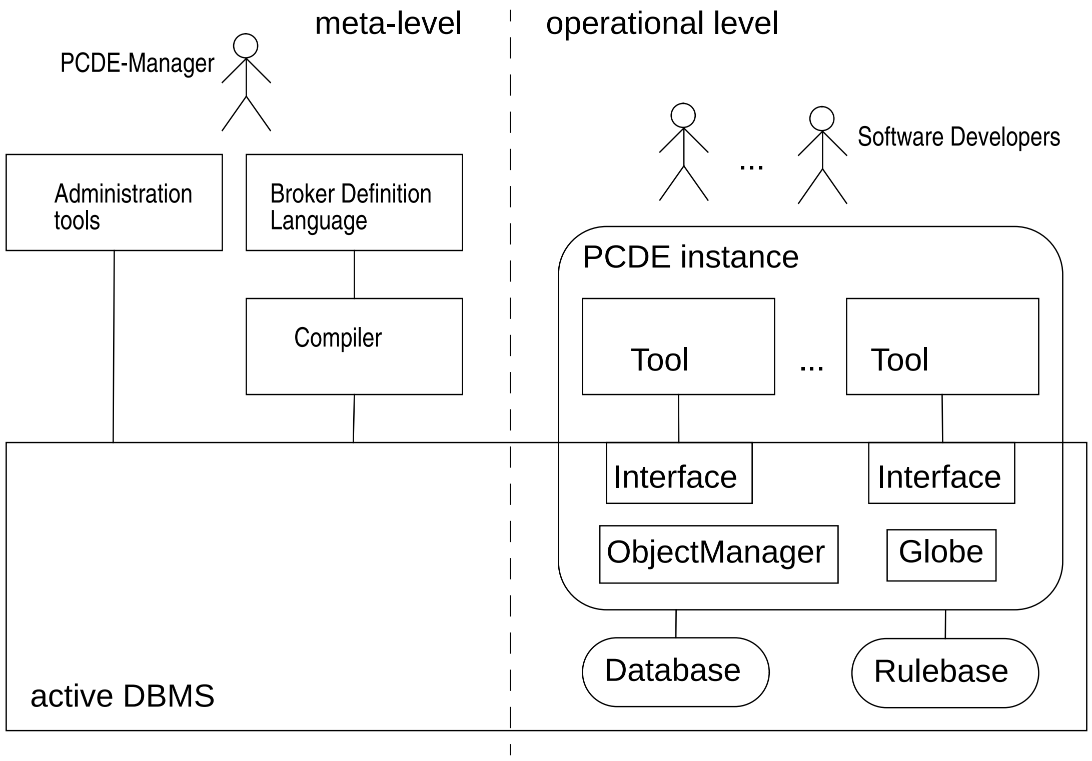

Figure 2: The architecture of SEAMAN

el brokers of the various subsystems (e.g. ObjectManager) and the interface brokers. This consists primarily in routing requests to responsible brokers by sending them an appropriate message. As noted before tools can be external entities and as such cannot contain sub-brokers.

3.1 Specification of PCDEs and Process Models in SEAMAN

The construction of single PCDEs, including process modeling, is the cornerstone of the SEAMAN environment. The entire construction is quite complex and is therefore described here on a rather abstract level.

The PCDE (including its behavioral aspect - the PM -) is specified in terms of brokers and services in an iterative process. The analysis of specific requirements yields service definitions realizing the functionality of the PCDE. These services are then assigned to responsible brokers. During this step, new brokers are added to the architecture whenever needed. The result of this first construction phase is an initial PCDE-architecture, which can then be completed iteratively. Extensions of this architecture may be required in two cases:

- when realizing other services results in the need for additional new services, and/or
- when decisions to realize a service in a particular way result in additional service requirements or architectural constraints.

The PCDE-manager defines the component architecture as a collection of top-level brokers. Internal brokers are defined which are responsible for the framework services (e.g., object management, integrity enforcement). These services are less likely to change among different PCDEs and thus provide a high reuse potential. Internal top-level brokers can contain sub-brokers or passive objects. For end-user services usually interface and external brokers are defined. Especially, human participants and generally all components for which the behavior cannot be exactly predefined are represented by external brokers. What is important is their interaction with interface brokers. External brokers are used primarily to ensure the completeness of the definition of the services provided by the system itself.

The PCDE-architecture is then extended to realize the desired PM. Particularly, note that the PM defines the behavioral aspect of the PCDE, and is defined as part and in terms of the PCDE-architecture. The PM is specified by adding role definitions to the existing brokers. Each such role defines a context-dependent behavior according to the state of the development process and the broker itself. This behavior is implemented in terms of production rules.

The result of the construction is an operational PCDE. Enactment of a concrete software process is implemented by the collection of brokers. The defined PM can be enacted after the initialization of the PCDE.

3.2 The ISWP6 Example

We are going to present an example of the use of the modeling concepts previously presented. It is inspired by the process modeling problem presented in [15]. Extensions were made to this problem in order to present how automatic system components are described. Because of the limited space we will just present a small part of the model describing the structure of some of the components involved and part of the behavior of selected brokers.

The problem consists of the design, coding, testing and management of a local change to a software system due to a change in the requirements. The whole process is simplified by the assumptions that only one module is affected and that there are no delays, constraints or conflicts due to resource unavailability. The process is divided into eight component tasks: scheduling and assignment, design modification, design review, code modification, test plan modification, test package modification, unit testing, and progress monitoring. Information that is available to all tasks are the change of requirements, the notification of the task assignments and their scheduled dates, the notification of the revised task assignments and their scheduled dates, and the notification of the completion of each step. Tasks are assigned to various human participants which work on them either alone or in groups. As already mentioned, a small part of the structure and behavior of participating brokers is shown here. This consists of the definition of the composite `Project-Team` broker, the `Designer` broker in its role within the project team and the interaction

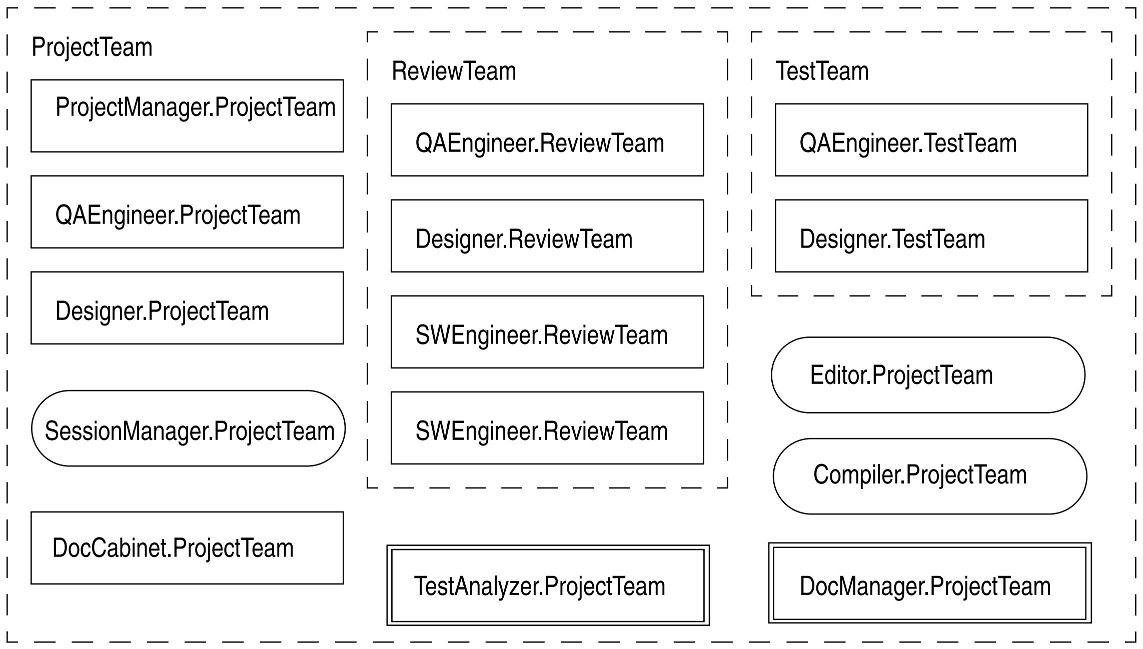

Figure 3: Structure of the brokers used to model the ISWP6 software process example. Interface brokers are represented as round boxes, external brokers as plain boxes, and internal brokers as a double box. Composite brokers are represented as dashed boxes. The role of brokers is given in postfix notation.

with the `SessionManager` representing the user interface of the designer during a coding session. Fig. 3 presents a schematic illustration of the broker structure of the example.

The `ProjectTeam` broker is defined below through its state variables and two rules describing its reaction to various events. When it receives the request for change and the change authorization event occurs, these are passed down to its components. It will also forward an eventual cancellation of the change request. This broker serves as the enclosing broker for the whole process and thus has as its state variables all other sub-brokers.

```
--------------------------------------------------------------------------------
EXTERNAL BROKER ProjectTeam

   rt:                 ReviewTeam
   tt:                 TestTeam
   de:                 DesignEngineer
   qe:                 QAEngineer
   pm:                 ProjectManager
   cc:                 Compiler
   ed:                 Editor
   dc:                 DocCabinet
   dm:                 DocManager
   ta:                 TestAnalyzer
   sm:                 SessionManager

   // react to initial request for change by requesting a scheduling service
   // and propagating the request to sub-brokers
   DEFINE Develop-Change
   ON   change-and-test(reqchange:Document) AND
        change-authorized(authorization:Verbal)
   DO   requestdown(change-and-test,reqchange,ASYNC)
        requestdown(change-authorized,authorization,ASYNC)

   // cancel change effort
   DEFINE RULE Cancel-Change
   ON   cancel-change
   DO   requestdown(cancel-change,ASYNC)

END
--------------------------------------------------------------------------------
```

The `Designer` broker models the behavior of a designer responsible for modifying the code, compiling it, and releasing it for test as soon as it compiles without errors. We just present here three production rules defining its reaction to the code modification service request in its role as member of the project team. A complete definition would include, in addition to other activities for this role, the definitions of its role in the review and test teams.

```
--------------------------------------------------------------------------------
EXTERNAL BROKER Designer

ROLE IN ProjectTeam

   // react to code modification service request, coding may begin sometime af-
ter
   // the task has been assigned or at the latest after design approval
   DEFINE RULE Modify-Code-1
   ON   modify-code(codetask:EMail) AND
        change-development-and-test(reqchange:Document)
   DO   DEFINE EVENT start-coding(date)

   // designer may start coding whenever he has decided...
   DEFINE RULE Modify-Code-2
   ON   start-coding
   DO   request(start-edit-session,”source code”,ASYNC)
        ... // edit source
        request(end-edit-session,”source code”,ASYNC)
        request(do-compile,”source code”,ASYNC)

   // but at the latest when design has been approved
   DEFINE RULE Modify-Code-3
   ON   NOT(start-coding) BEFORE
        design-approved(outcome:EMail,defcts:EMail,effrt:EMail,design:Document)
   DO   request(start-edit-session,”source code”,ASYNC)
        ... // edit source
        request(end-edit-session,”source code”,ASYNC)
        request(do-compile,”source code”,ASYNC)
        DELETE EVENT start-coding
   ...
END
```

Finally, the `SessionManager` interacts with the `Designer` by starting editor and compiler sessions. Note that the two brokers `Editor` and `Compiler` are not the actual tool components but only interface brokers. These call the appropriate external tools with the needed parameters. The tools are not called directly by the `SessionManager`, a fact which provides the possibility of customizing them for a specific role (e.g. pass different parameters to a compiler according to the context of its use).

```
--------------------------------------------------------------------------------
INTERFACE BROKER SessionManager

ROLE IN ProjectTeam

   scr:      Screen
   session:  Session

   // start interactive editor session
   DEFINE RULE Start-Edit-Session
   ON   start-edit-session(fn:Name)
   DO   request(edit-file,fn,SYNC)
        session=scr->newSession(fn)
        reply(start-edit-session)

   // end interactive editor session
   DEFINE RULE End-Edit-Session
   ON   end-edit-session(fn:Name)
   DO   scr->closeSession(fn)

   // start compilation
   DEFINE RULE Start-Compilation
   ON   do-compile(fn:Name)
        request(compile-file,fn,SYNC)
        request(compile-end,result,ASYNC)

END
--------------------------------------------------------------------------------
```

4 Using an Active Object-Oriented DBMS in SEAMAN

A powerful realization platform is desired for the construction and operation of SEAMAN PCDEs. In this section we identify specific requirements, present the chosen platform, and show how it is used for PCDE construction.

4.1 Requirements for the Realization Platform

The realization platform used for SEAMAN has to fulfill a number of requirements, stemming from its use both as construction framework and operational environment. The requirements for the construction framework include management and persistent storage of the PCDE definitions and their PMs, and support for the definition of reactive behavior. The requirements for its operational use include persistent storage of produced artifacts, transaction management and concurrency control, automatic recovery, and the realization of reactive behavior.

In order to meet these requirements, we use an active object-oriented DBMS as the realization platform. ADBMSs extend passive object-oriented DBMSs by supporting the reactive behavior of stored objects through rules defining reactions to predefined situations. Situation monitoring can be done in the aDBMS instead of in polling applications. “Active objects” are able to automatically react to changes in their internal state and to external events.

For the implementation of SEAMAN we need an aDBMS which provides the mechanisms to implement both the static and the dynamic aspects of a PCDE. The static part refers to the implementation of PCDE components and the storage of the software artifacts developed. Both are implemented as database objects. The dynamic behavior refers to the enactment of PMs (by brokers). The aDBMS rule mechanism is used to implement the reactive part of brokers. In order to achieve fine-grained process enactment, reactive behavior has to be defined at an arbitrarily fine level [5] (i.e. at the level of simple objects). Rules are also used to express consistency constraints [13]. Finally, the aDBMS is used for the implementation of application programs operating on the stored data. These programs are used among other things, for the realization of parts of the meta-level of SEAMAN.

In SEAMAN we use the aDBMS SAMOS [11] which is based on the object-oriented DBMS ObjectStore and permits the specification of reactive behavior by Event-Condition-Action (ECA) rules consisting of an event, a condition predicate and an action part. Rules can be attached to classes (*class-internal*) or can be *class-external*. Events in SAMOS can be *primitive* or *composite*. The condition part of SAMOS ECA-rules is a function expressed in the DML of the underlying ooDBMS and returns a boolean value.

SAMOS uses a nested transaction model. It also provides nested rule execution. During the execution of a rule, new events may occur as a result of actions performed and trigger the execution of further rules. The execution order of multiple rules defined for the same event can be specified through precedence relations among rules.

4.2 Mapping of Brokers to SAMOS Objects and Rules

In this section we are going to describe how SAMOS is used to realize instances of PCDEs. This step comprises the realization of a concrete PCDE consisting of a set of broker definitions. It is done by compiling the broker definitions thus mapping them to elements of the SAMOS model and initializing the run-time environment. The initialization includes the creation of a new database and rulebase as well as the execution of broker initialization methods.

The principles of the mapping of the PCDE definition on to the model provided by SAMOS are the following:

- Each instance of a PCDE has its own SAMOS database. This is needed in order to achieve an isolation of possible side effects between the process models used in the different PCDEs. Thus after the definition of the PCDE a new database is created containing the production rules and the data pertaining to it. It exists for the whole duration of the process.
- Internal brokers and interface brokers are mapped to classes. The production rules for a role are class internal ECA-rules. The different roles of a broker are expressed by adding a role-dependent state variable in each class. Role transitions are thus easily implemented.
- External brokers are also mapped to classes which however do not have any class internal ECA-rules. Their rules are just defined for checking the completeness of the specification of automatic and interface brokers.
- Consistency constraints are specified using the programming-by-contract [16] paradigm and are also transformed to ECA-rules through the mapping process described in [13]. Object-level and extension-level invariants can be defined.

5 Related Work

Two aspects of SEAMAN are considered in this comparison to related work: the underlying object model and the provided event type support. These two aspects are important when considering the degree of control and data integration provided in PCDE architectures.

ADELE -TEMPO [3, 8] is a PCDE consisting of two layers: a custom-made database based on an extended entity-relationship data model (providing objects and relationships between them) and a software process description layer where processes are modelled as an aggregate of object roles (customizing object behavior) to which constraints can be attached. External tools can only be attached by using the Unix shell interface. The expressiveness of the data model used is limited compared to SEAMAN and only simple method events can be defined in an external to the database layer.

ALF [19] is similarly to our approach a meta-environment for the creation of process-centered CASE environments supporting different life-cycle models and design and development methods. It is based however on a structurally object-oriented [7] object management system [18] which conceptually separates objects from their functionality. Production rules defining automatic reactions to specific situations arising during the software process have to be executed by a separate interpreter serving as the communication layer between the object manager and the user interface. Only a limited set of events related to database operations can be defined (read, create, update, lock, etc.). Operators, to which pre- and post-conditions can be attached, are used to describe various activities performed by external tools.

EPOS [14] is a multi-user kernel software development environment providing process modeling and configuration management facilities. It has a layered architecture consisting of a common user interface to various programming tools, to an activity manager and a planner. Process artifacts are represented by entities (objects) with mutual relationships. The basic concepts used for PM are activities and products, which are described as types in an extension of the basic data model. EPOS is based on a structurally object-oriented DBMS. In contrast to SEAMAN, reactive object behavior can only be provided by using a special interpreter and is not an inherent part of the database objects.

Marvel [4] is a rule-based PCDE. Production rules are used for process modeling. Marvel has a fixed client-server based architecture and uses a proprietary transaction manager and object manager. The data model provided by the object manager is structurally object-oriented providing only aggregation, generalization/specialization and links between objects. The implicit representation of enactable process models and executing processes makes it difficult to change them or inspect them.

In SPADE [2] PMs are described in a Petri net based language called SLANG. The SPADE environment is divided into the user interaction environment, the process enactment environment in which process engines concurrently execute activities of the process model, and a filter responsible for the communication of the two environments. Special transitions for which only the input and output has to be known are used to model user interaction with tools. The type system of SLANG permits the modelling of data produced, used, and manipulated in a software process. Meta-types allow the manipulation of PMs themselves. SPADE resides over the aDBMS NAOS [6] which compared to SAMOS supports a smaller set of event types (user-defined, method and program execution events) and no composite events. Rules for user-notification, application access-logging, tool communication etc. can be modified, it is not yet clear however how they are exactly used. The architecture of SPADE is predetermined, only the used PM can be modified.

Summarizing, some of the concepts we use exist in previous systems. However in SEAMAN a unifying concept of development architectures, processes and their products is provided. SEAMAN uses as its realization platform a fully object-oriented active DBMS simplifying the development of customized PCDEs significantly. Most existing systems use structurally object-oriented DBMSs. By using a fully object-oriented DBMS we have the additional advantage of being able to easily integrate passive with reactive components representing their behavior and structure in a uniform way.

6 Conclusions and Future Work

We have described a concept for the definition and construction of flexible PCDEs. An important aspect of our work is the use of the extended broker/services model for the definition of PCDE architectures. The concepts we use allow a structural and behavioral decomposition of the system by using composite brokers, they support information hiding, modularity and extensibility of the architecture. The generation of an instance of a concrete PCDE is largely automated and adaptation of the process model used is supported. Data, tool, and control integration [5] is supported by the use of a powerful aDBMS as the realization platform. The aDBMS offers the facilities for process modeling and enactment, process and artifact management, and consistency enforcement.

Although simpler active-mechanism like approaches have been used in other projects the use of *active object-oriented database technology* as a unifying factor has not yet been explored. By using object-oriented technology, we can associate passive functional behavior with objects providing a basis in the data model to access services through object references. This results in a clean integration concept. By using active mechanisms we specify the reactive behavior of software components and various consistency constraints. In addition to a large spectrum of primitive event types including various types of time events (intervals, periodic, etc.) we can also exploit the expressive power of composite events. Thus the aDBMS provides the functionality of a process engine.

We expect that specifications (mainly for process models) can *evolve* during the progress of a software engineering project. Once a process model has been defined it may be necessary to modify it even if the process is already running, in case requirements have changed or the initial specification has turned out to be inappropriate. Of course, the rule base implementing the process model also evolves in such a case. Hence, the questions are which modifications can be performed on a rule base without compromising its consistency, and how the rule base has to be modified. Rulebase evolution is also still an open question in database research and is a part of our future research.

7 References

1. M. Atkinson, F. Bancilhon, D.J. DeWitt, K.R. Dittrich, D. Maier, S.B. Zdonik: *The Object-Oriented Database System Manifesto (a Political Pamphlet).* In *Proceedings 1st Int'l. Conf. on Deductive and Object-Oriented Databases*, 1989.
2. S. Bandinelli, M. Braga, A. Fuggetta, L. Lavazza: *The Architecture of the SPADE-1 Process-Centered SEE.* In B.C. Warboys (Ed.), *Software Process Technology, Proc. 3rd European Workshop*, Villard de Lans, 1994.
3. N. Belkhatir, J. Estublier, W.L. Melo: *Software Process Model and Workspace Control in the Adele System.* In L. Osterweil (Ed.), *Proc. 2nd Int'l. Conf. on the Software Process*, Berlin, 1993.
4. 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.
5. A.W.Brown, P.H.Feiler, K.C. Wallnau: *Past and Future Models of CASE Integration*. In *Proc. Int. Conf. Computer Aided Software Engineering*, Montreal, 1992.
6. C. Collet, T. Coupaye, T. Svensen: *NAOS Efficient and modular reactive capabilities in an Object-Oriented Database System.* In J. Bocca, M. Jarke, C. Zaniolo (Eds.), *Proc. 20th Int'l. Conf. on Very Large Data Bases*, Santiago, September 1994.
7. K.R. Dittrich, *Preface*. In K.R. Dittrich (Ed.), *Advances in Object-Oriented Database Systems, Proc. of 2nd Int'l. Workshop*, Bad Munster am Stein-Ebenburg, LNCS 334, Springer 1988.
8. 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 Int'l. Workshop on Computer Aided Software Engineering*, Montreal, 1992.
9. European Computer Manufacturers Association, National Institute of Standards and Technology: *Reference Model for Frameworks of Software Engineering Environments*. Technical Report ECMA TR/55, NIST Special Publication 500-211, August 1993.
10. A. Finkelstein, J. Kramer, B. Nuseibeh: *Software process Modelling and Technology.* Research Studies Press, Taunton, 1994.
11. S. Gatziu, K.R. Dittrich: *Detecting Composite Events in Active Database Systems Using Petri Nets*. In *Proc. 4th Int'l. Workshop on Research Issues in Data Engineering: Active Database Systems*, Houston, February 1994.
12. A. Geppert: *Methodical Construction of Database Management Systems*. Doctoral Dissertation, University of Zurich, 1994.
13. A. Geppert, K.R. Dittrich: *Specification and Implementation of Consistency Constraints in Object-Oriented Database Systems: Applying Programming-by-Contract.* In *Proc. Conf. Datenbanken in Büro, Technik und Wissenschaft (BTW)*, Dresden, March 1995.

14. L. Jaccheri, J.-O. Larsen, R. Conradi: Software Process Modeling and Evolution in EPOS. In *Proc. 4th Int'l. Conf. on Software Engineering and Knowledge Engineering*, Capri, 1992.

15. M.I. Kellner, P.H. Feiler A. Finkelstein, T. Katayama, L.J. Osterweil, M.H. Penedo, D. Rombach: *ISWP-6 Software Process Example*. In *Proc. 6th Int'l. Software Process Workshop*, IEEE, 1991.

16. B. Meyer: *Object-Oriented Software Construction*. Prentice-Hall, New York, 1988.

17. NCGR Project Support Environment Standards Group, National Institute of Standards and Technology, Software Engineering Institute -CMU: *Reference Model for Project Support Environments (Version 2.0)*. Technical Report NIST SP 500-213, CMU-SEI-93-TR-23, November 1993.

18. F. Oquendo, G. Boudier, F. Gallo, R. Minot, I. Thomas: *The PCTE+ OMS: A Distributed Software Engineering Database System for supporting Large-Scale Software Development Environments*. In *Proc. 2nd Int'l. Symposium on Database Systems for Advanced Applications*, Tokyo, April 1991.

19. F. Oquendo, J.-D. Zucker, P. Griffiths: *A Meta-CASE Environment for Software Process-centred CASE Environments*. In *Proc. 4th Int'l. Conf. on Adavnced Information Systems Engineering*, Manchester, May 1992.

20. B. Peuschel, W. Schäfer: *Concepts and Implementation of a Rule-based Process Engine*. In *Proc. 14th Int'l. Conf. on Software Engineering*, Melbourne, 1992.

21. J. Richardson, P. Schwarz: *Aspects: Extending Objects to Support Multiple, Independent Roles*. ACM SIGMOD, May 1991.