CREWS Report Series 97–12

A Contextual Approach for Process-Integrated Tools

Klaus Pohl and Klaus Weidenhaupt

RWTH Aachen, Informatik V, Germany

Ahornstraße 55, 52056 Aachen

e-mail: {pohl,weidenhaupt}@informatik.rwth-aachen.de

appeared in Proceedings of the 6th European Software Engineering Conference / 5th ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 97), Zurich, September 22-25, 1997.

A Contextual Approach for Process-Integrated Tools

Klaus Pohl and Klaus Weidenhaupt

RWTH Aachen, Informatik V, Ahornstraße 55, D-52074 Aachen, Germany
{pohl,weidenh}@informatik.rwth-aachen.de

Abstract: *Research in process-centered environments (PCEs) has focused on project management support and has been dominated by the search for suitable process modelling languages and enactment mechanisms. The consequences of the process orientation on the tools used during process performance, and for offering fine-grained, method-based support to the engineers performing the process have been studied much less.*
In this paper, we discuss the requirements for a tighter integration of interactive engineering tools and present a contextual approach for the process-integration of those tools. To achieve process integration we argue that tools, like processes, should be explicitly defined. The integration of the tool models with the process definitions forms an environment model which is interpreted during tool execution. Based on this interpretation tool behavior is adjusted according to the process definition; i.e. the interpretation empowers the tools to provide fine-grained method-conform process support.
Our approach has been implemented as a reusable object-oriented framework and validated by specializing this framework to develop two prototypical process-integrated environments (PIEs).

1 Introduction

During the last decade a tendency of moving from product-oriented computer supported development environments to process-oriented environments, so-called process-centered environments (PCEs), could be observed. The explicit definition of processes is a prerequisite for easy adaptation to project specific needs and the integration of process changes. In contrast, the process support offered by product-oriented environments is hard-coded, i.e. there exists no explicit process definition. Due to the required re-programming process changes are hard to accomplish.

PCEs can be divided into three conceptually distinguishable domains [1,2]: the modelling, the performance, and the enactment domains. The *modelling domain* comprises all activities for defining and maintaining software process models using a formal language with an underlying operational semantics which enables mechanical interpretation of the models. The *enactment domain* encompasses what takes place in a PCE to support (guide, enforce, control) process performance; this is essentially a mechanical interpretation of the process models by a so-called process engine. The *performance domain* is defined as the set of actual activities conducted by human agents and non-human agents (computers).

Process support provided by PCEs can be characterized by the typical interactions between the three domains: (1) a process model is instantiated, i.e. process parameters like resources and time scheduling are bound to project specific values and passed to the enactment domain; (2) based on the interpretation of the instantiated model, the enactment domain supports, controls, and monitors the activities of the performance domain; (3) the performance domain provides feedback information on current process performance to the enactment domain. This is a prerequisite for adjusting process model enactment to the actual process performance and enabling branches, backtracks, and loops in process model enactment.

Research in the PCE area has focused on *process (project) management support* and has been dominated by the search for suitable process modelling languages and enactment

mechanism, i.e. *has focused on the modelling and enactment domains* [3]. It has resulted in a set of mature process modelling languages and enactment mechanism (e.g. [4,5,6,7,8,9]). Although the need for process integration of the tools used to perform the process has been recognized [10,3,11,12,2] the integration of the performance domain, respectively the tools, with the modelling and enactment domains has almost been neglected [3,2,11]. Such an integration is essential if fine-grained, methodical support should be offered to the engineers, e.g. during the creation of a requirements specification or a high/low level design.

The process integration of the tools presented in this paper is based on the assumption that the humans executing the process should play an active role during process execution, i.e. that process execution should not only be dictated by the process enactment mechanism. We argue that *process-integrated tools* are a prerequisite for enabling the user to understand and control process execution and to play a more active role by initiating the execution of predefined method fragments depending on the actual process situation.

In section 2 we elaborate the main requirements to be considered for process-integration of interactive engineering tools. The key ideas for achieving these requirements are sketched in section 3. The consideration of those requirements has led to a significant improvement of our approach for process-integration of tools described in [2]. We now advocate that for achieving process integration the capabilities of the interactive tools should be explicitly defined. The resulting tool models should, in addition, be integrated with the process definitions. Our new contextual approach for defining and integrating tool and process models is outlined in section 4. Our overall approach has been validated by implementing a generic architecture which interprets the integrated model and thereby assures that the engineering environment "behaves" according to the method definitions (section 5).

Finally, we compare our contextual approach for process-integrated tools with other research and existing PCEs (section 6) and provide an outlook to future work (section 7).

2 Requirements for Process-Integrated Tools

In the enactment domain, process definitions are enacted to drive process performance. In the performance domain, tools are used by humans to execute the various process steps and method fragments. Integrating these two domains for providing fine-grained methodical support for the engineers has different facets and requirements which are identified in this section.

A widely adopted view on the tool integration problem has been proposed by Wasserman who distinguishes platform, user interface, data, control, and process integration [10] but, like other authors, does not discuss process integration in depth. The integration between the enactment and performance domain mainly has to cope with data, control, and the process integration aspects. While we share Wasserman's view on data integration and control integration, process integration requires certain features which are not discussed in the literature so far. These requirements are also related to, but significantly more comprehensive than those discussed by [1,13,14,15,16,17] whose analysis is mainly based on the weak integration of the enactment and performance domains in existing PCEs.

2.1 Service Integration

A *tool service* is a functionality provided by a tool which can be accessed (called) from outside, e.g. the creation of a certain artefact, the compilation of the source code, printing or loading a document, etc. Tool services can vary in their complexity. To ensure that the tools of the performance domain can execute the services requested by the enactment mechanism, the tools used must be considered when defining process models.

Current process modelling formalisms lack comprehensive modelling concepts for representing tool resources at the same conceptual level as processes. They offer only limited, low-level constructs for representing service invocation (e.g. black transitions in SPADE [4], the `call`-statement in Marvel [5], or the wrapping techniques for blackbox integration employed in the OZ environment [18]). For exploiting existing tool support in the process definitions the method engineer has to collect information about the available tools, their services and the service invocation (e.g. parameters required) from various sources (e.g. manuals, program documentations, personal knowledge or experience). Considering the heterogeneous environments and work settings which exist today in industry, guaranteeing that the services defined in the process model are correctly mapped to the tool environment used for performing the process is not trivial, especially if a process model is enacted in different environments.

Thus, mechanisms are required which systematically support the method engineer in finding and assigning adequate tool support to certain process steps. If the available tools (e.g. their services, in and out parameters of the services) are defined at a conceptual level the method engineer can be supported in relating the tool services to the process definitions. For example, the tool and process models can be compared and discrepancies, such as lack of sufficient tool functionality or wrong assignments, can be detected.

2.2 Invocation of Method Fragments

On the one hand, methodical support can not be fully predefined due to many criteria not known a priori which influence the actual process performance and/or due to the weak understanding of the steps themselves. The actual process performance is thus often driven by humans which, depending on the process situation, decide what to do next, i.e. process performance depends on intelligent and creative individuals which make the right decisions. It is thus important that the computer based environment does not restrict the humans in their creativity.

On the other hand, there exist method fragments which are well understood, do not depend on unknown criteria and can thus can be predefined [19,2]. To increase the productivity and the quality of the product under development, such methodical knowledge should, whenever possible, be used to guide the engineer. As a consequence, the computer-based environment has to assure that the well understood method fragments are used to guide the user whenever possible.

Process-integrated tools must thus provide means for initiating the execution of predefined method fragments, e.g. by comparing the current process situation with the method fragment definitions.

2.3 Process Sensitive Tools (Informing the User about the Enactment State)

A tight integration between the enactment and performance domains can only be achieved if both domains consider the process status of each other. The enactment domain has to consider current process performance for deducing the process steps to be performed next (section 2.4), whereas the current enactment state has to be reflected in the performance domain. To assure that the user is aware of the current enactment state, the tools must be process sensitive. A process sensitive tool

- adapts its behavior (the user interactions allowed and the services provided) according to the current enactment state and the process definitions. For example, the selectability of product parts may be restricted to the ones allowed in the current state, or the product parts on which a service can be performed are highlighted to draw user attention to them;
- empowers the user to activate predefined method fragments and services provided by other tools. Since process definitions are subject to frequent change, the activation

of predefined method fragments and services should not be hard-coded in the tool. Instead, the activation should be based on the actual process definitions. The tools must be process-aware; i.e. must have knowledge about the actual process definitions.

Supplying the performance domain with knowledge about the enactment state is straightforward in situations where a particular service has to be performed on a certain product part. In this case, the relevant product parts are passed as parameters of the service request. If there are alternatives among which the user has to choose, the enactment domain must inform the tool about the set of services allowed, the selectable product parts for each service, respectively product part combinations.

2.4 Feedback Information (Informing the Process Engine about Performance State)

For adjusting the enactment state according to the current process performance, the performance domain must provide feedback information about the execution of a particular service by a tool. The data to be exchanged depend on the service executed. Consequently, the feedback data have to be defined as out-parameters for each service type (see service integration above). In addition, information about the current process performance state, e.g. unforeseeable events like a process deviation, has to be provided. This information can either be created by observing (monitoring) activities, or directly provided by the user.

Technically, the distribution of the feedback information has to be enabled by a control integration mechanism.

The problem of gathering feedback information from the performance domain has also been recognized in current approaches. In SPADE [4], for example, a specific Petri-Net construct, the user input place, has been introduced. Message events generated by the tools have to be mapped into tokens of such places. In Provence [20], the enactments mechanism captures events from the performance domain via a monitoring system for operating system traps, e.g. file system accesses. However, mapping performance domain events to feedback information understood by the enactment mechanism is by far not trivial, e.g. to deduce that saving a file in an text editor means that a bug fix in the source file has been completed.

2.5 Synchronization of Enactment and Performance Domain

The definition of a communication protocol and its application within each domain is a prerequisite for synchronizing the process states of both domains. In current PCE approaches, the interaction between the enactment domain and the tools is typically established by an implicit client-server relationship, i.e. the enactment domain acts as a client which requests the execution of a tool service. Conversely, the tool plays the role of a server which executes the service and returns the results (feedback information) to the enactment mechanism. This simple cooperation pattern is sufficient as long as we consider traditional tools which are not process-integrated.

Due to the more active role of process-integrated tools (section 2.2 - 2.4), a more elaborate interaction protocol between the two domains than client-server has to be provided. Such a protocol should, for example, distinguish between different process states like normal process performance, process deviations, the performance of automated services, or user choices.

2.6 Process-Aware Control Integration Mechanism

In contrast to service integration which defines the service interfaces, a control integration mechanism is required for transmitting particular service requests and feedback information between the components of a process-integrated environment. A control integration mechanism is responsible for passing the requested service to a tool which is able to execute the service. To enable correct physical distribution of the service requests and the

feedback information provided after service execution, the control integration mechanism has to be aware of the services provided by a particular tool. In addition, service and feedback distribution has to consider relevant knowledge defined in the process model, e.g. if the process model restricts possible allocation of resources needed for performing a service, or if the model explicitly defines the service provider, this has to be considered when distributing a service. Thus, either the control mechanism must be process-aware, i.e. must know the relevant parts of the actual process definition, or the enactment domain must instruct service distribution according to the process definition. In most existing PCEs (e.g. SPADE [4], MELMAC [6], Merlin [21], ProcessWEAVER [9]), neither a process-awareness of the control mechanism nor the ability of the enactment domain to control service distribution is offered.

Existing control integration mechanism like FIELD [22], BMS of HP's Softbench [23], Tooltalk [24], or CORBA [25] provide an excellent foundation for implementing a process-aware control integration mechanism.

3 Key Solution Ideas

Our key solution ideas for fulfilling the requirements discussed in section 2 (see figure 1) are:

- The explicit definition of *tool models* and their integration with process models (section 4). The solution described in this paper extends our previous solution [2] in two main aspects. First, tool models are defined separately from the process models and represent the capabilities of the tools. Secondly, the integration of the tool and process models forms an integrated *environment model* that takes both process and tool support into account. The environment model offers five main advantages: (a) it achieves service integration by uniformly defining the services provided by the tools and by the enactment mechanism; (b) it enables the tool to adapt the user guidance according to the actual method definitions; (c) it defines how objects should be displayed by the tool; (d) it defines the required feedback information for each type of service request; (e) it empowers the tool to support the user in initiating the invocation of predefined method fragments.
- The definition of a comprehensive *communication protocol* as a basis for synchronising

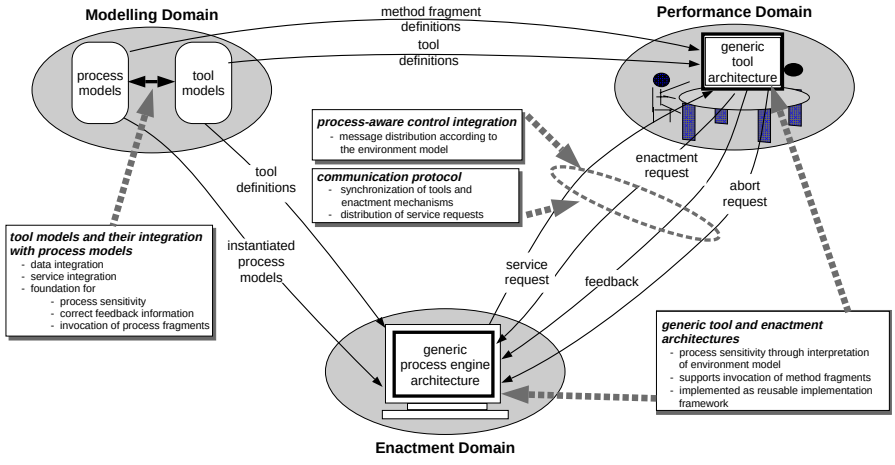

Fig. 1. The Key Solution Ideas.

both domains. The communication protocol extends the conventional client-server relationship between enactment mechanism and tools offered by most PCEs to a more flexible pattern where both domains can play the role of a server and a client, i.e. the enactment domain can request the execution of tool services and the performance domain (tools) can request the enactment of predefined method fragments. The detailed description of the communication protocol can be found in [2];

- The implementation of a *process-aware control integration mechanism* on top of ToolTalk [24], which ensures that service requests are distributed between both domains in accordance with the process definitions.
- The development of *generic architectures* for process-integrated tools and enactment mechanisms (section 5). Both architectures ensure that process performance is in accordance with the environment model and guarantee the synchronization of the enactment and performance domains based on the communication protocol. The generic tool architecture facilitates the invocation of predefined method fragments and assures that the guidance provided to the user corresponds to the process definition and the actual process situation. The generic enactment architecture handles enactment requests of the performance domain by enacting the requested method fragments and provides means for an easy integration of existing enactment mechanisms.

4 Modelling Tools and Processes

According to the requirements discussed in section 2 there are *three types of services* in process-integrated environments (PIE): automated, guidance and enactment services (section 4.1). Whereas the process model defines when a service should be performed, the tool model defines which services are provided by a particular tool.

In section 4.2 we sketch a *contextual process meta model*, i.e. a process modelling language, which defines an ontology for representing the three service types as well as their situated invocation. We further illustrate the definition of the three service types using the defined ontology.

For representing the services offered by a tool we propose a *tool meta model*, i.e. tool modelling language, which provides additional concepts for defining the capabilities of the tool (section 4.3). The definition of the concepts of the tool meta model was driven by the need for an easy integration with the process meta model.

An integration of the proposed tool and process meta models can be achieved by defining associations between both meta models (section 4.4). Thereby the so called *environment meta model* is formed.

4.1 Requirements for Modelling Concepts

4.1.1 Three Service Types

According to the requirements discussed in section 2 there exist three types of services in a PIE:

- *automated services* which require no user interactions and are executed by the tool according to the service request obtained by the enactment domain;
- *guidance services* which guide the user in making a selection among a set of alternative services and product parts. If the execution of a *guidance service* is requested, the tool must adapt its behavior (the services offered and the product parts displayed at its user interface) according to the process definition and the information obtained with the service request;
- *enactment services* which enable the tools to request the enactment of a predefined method fragment from the enactment domain.

To illustrate the three service types assume a requirements engineering environment consisting of a set of interactive tools.

In this environment the creation of an entity type is an atomic action (*automated service*) provided the Entity Relationship (ER) editor. The refinement of an entity type is defined as *guidance service*. Since the guidance service defines two possible ways for achieving a refinement of an entity type (discrimination of attributes and the specialization (subtyping) of the entity type), the ER editor has to offer these alternatives to the requirements engineer, i.e. the ER editor has to adapt its user interface according to the guidance service definition.

The refinement technique "subtyping" is defined as a method fragment which specifies a set of process steps (services) to be performed in a certain order. If the requirements engineer selects this alternatve, the defined method fragment has to be enacted by the process engine, i.e. the ER editor has to request the execution of the predefined method fragment by the process engine (*enactment service*).

4.1.2 Service Invocation is Situated

The applicability of a particular service depends on the current process situation. A situation is normally regarded as an abstraction of current reality based on observed object states (see [26,2] for details). A simple example for the situated nature of actions is a delete action; if no object (e.g. entity) exists the delete action (e.g. delete entity) can not be applied.

In most situations many services (actions) can be applied. For example, when modelling an ER-diagram you can create an arbitrary new entity, delete one of the entities, define a new attribute and so forth. Consequently, a choice among the allowed services has to be made. As stated in many contributions, such a choice (decision) is always driven by the goal (or set of goals) which the person (or group) tries to achieve (e.g. [27,28,29,30,31]). The decision which service to apply in a given situation can either be predefined in the process model (i.e. the process modeler does not allow any choice) or the decision has to be made by the engineers performing the process (i.e. the process modeler has defined more than one service for a given situation).

To enable the definition of a set of services which can be applied in a given situation, we have to define the situations themselves as objects and provide means to represent the goal the user wants to achieve in a given situation.

4.2 A Contextual Process Meta Model

The contextual process meta model (see upper part of figure 2) described in this section was developed within the NATURE project (see [32,33,19,2] for a detailed description).

A *situation* is built from product parts of the *product* undergoing the development process. An *intention* reflects the goal to be achieved in a given situation. A *context* represents a meaningful relation between a situation and an intention. Thus, the meta model provides concepts for the explicit representation of process situations and the goals to be achieved in such situations. The refinement of the notion of context into executable, choice and plan contexts enables the representation of the three service types which appear in a PIE:

- *Executable contexts* represent the part of the process definition which can be strictly enforced, or even automated, i.e. the user does not have any choice what to do next. An executable context is operationalized by performing the *action* related to this context. Performing the action changes the product and may thus generate new situations. Thus, executable contexts are used to define atomic tool services (automated services);
- *Choice contexts* represent the part of the process definition, in which the user has to make a decision. For each choice context at least two *alternative*s must be defined. An alternative can be another choice, executable, or plan context. For each alternative, *argument*s (pros and/or cons) can be provided to guide the application engineer in

choosing one of the alternatives. Thus, choice contexts are used to define the guidance services.

- *Plan contexts* define a strategy to fulfill a particular intention (goal); i.e. they define a certain order on a subset of arbitrary contexts (plan, choice, and/or executable contexts). A plan context can be supported by forcing the application engineer to deal with the contexts in the order defined by the plan. Thus, plan contexts are used to define enactment services.[1]

4.3 The Tool Meta Model

Representing processes and tools at a conceptual level is a prerequisite for comparing and mapping the services defined in the process definition to the services offered by the tools of the environment. For achieving process-sensitive tools we propose to model tools not only in terms of the services provided (as in most other PCE approaches), but also in terms of their graphical user interface and interaction capabilities. The tool meta model was designed to facilitate an easy integration with the contextual process meta model proposed in section 4.2.

The cornerstone of the tool meta model is the concept *tool category* (see lower part of figure 2). An atomic service (*action*) provided by a tool category is related to the tool category by a *provides_action* association, e.g. the action CreateEntity is related to the ER-editor. For each action the *input* and *output* parameters (*products*) are defined. The modelling of the atomic actions is important for assigning executable contexts to tool categories (section 4.4).

The graphical presentation of the product parts is defined as instantiation of the association *displays* between a graphical shape provided by the tool (modelled as instances of the concept *graphical shape*) and a product part.

Besides the capabilities of a tool for displaying product parts, also the interaction capabilities have to be defined. We assume that each tool enables the selection and de-selection of product parts by default and thus do not model these interactions. In contrast, the *CommandElements* provided by a tool have to be explicitly defined. In our implementation we currently distinguish between three types of such capabilities, namely *PullDownMenu*, *ControlIcons* and *ControlKeys*.

4.4 The Environment Meta Model: Integrating Process and Tool Meta Models

While the process meta model supports the definition of method fragments in terms of executable, choice and plan contexts, the tool meta model is used to define the capabilities of the tools available in the environment. By interrelating the tool and process meta models an integrated meta model, the so called *environment meta model*, is formed which defines how and by which tool a context has to be executed (see dashed lines in figure 2).

To define the tool category responsible for executing a choice or executable context (remember plan contexts are enacted by process engines) the executable and choice contexts defined in the process model must be related to the tool definition. Since the tool meta model was designed with this interrelation in mind, the integration of the tool and process meta models is fairly easy (see figure 2).

Besides the easy integration, the tool and process meta models facilitate consistency checks through which a consistent interrelation of both models can be supported.

---

[1] For defining the *control flow* of plan contexts we propose to represent the concepts of the process meta model using an existing process modelling language. For our prototypical environments (section 5) we have chosen the petri-net language SLANG [4] and the imperative language C++ (see [34] for details).

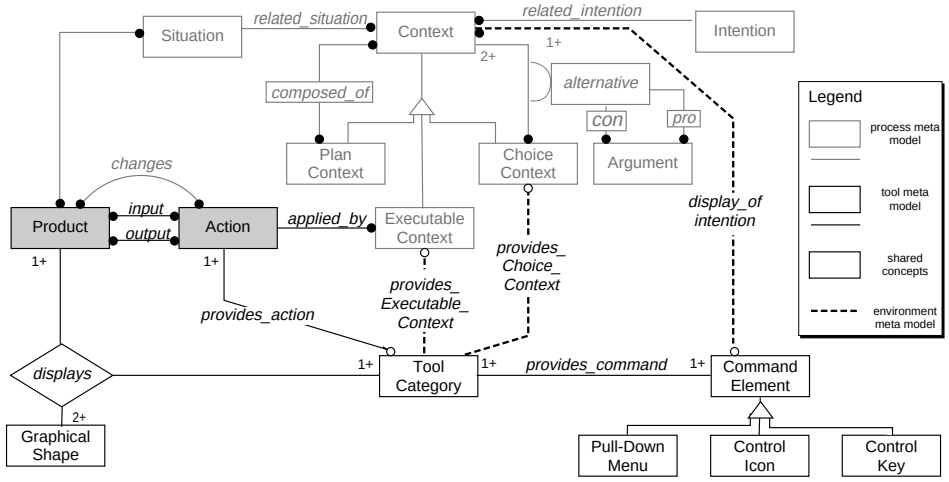

Fig. 2. The Environment Meta Model.

4.4.1 Relating Tool Categories and Executable Contexts

Each executable context defined in the process model has to be associated with the tool category responsible for executing the context. This responsibility is represented as an instance of the association *provides_executable_context*. For example, if the executable context *CreateEntity* is related to a tool category *ER-editor* the ER-editor has to perform the action *A* associated in the process model with this context.

Given an executable context *E*, the associated action *A* and a set of tool categories T_1 – T_n, we distinguish between three types of assignment:

- *automated assignment*: If there exists exactly one tool category T_i which offers the required action *A* this tool category is automatically associated to the executable context *E*;
- *choice of tool category*: If there exist two or more tool categories T_1 - T_n which offer the required action *A* the method engineer must relate exactly one tool category with the executable context *E*;
- *lack of tool support:* If no tool category provides the required action *A* a new tool action has to be implemented in a tool and defined in the corresponding tool model, or the process model has to be changed.

For each assignment between a tool category and an executable context two consistency checks can be performed to ensure that the input and output defined for the tool action in the tool model correspond with the process model definitions.

C1 *assure that the* output *associations defined in the tool model between the action and the product parts are subsumed by the* change *associations defined for the action in the process model:* Given an action *A*. Let P_o be the set of product parts related to *A* in the tool model using *output* associations and P_c the set of product parts related to *A* in the process model via *change* associations. Then, P_o must be a subset of P_c;

C2 *assure that all product parts defined as* input *for the action are subsumed by the situation of the executable context related to the action:* Given an action *A*. Let P_i be the set of product parts related to *A* in the tool model via *input* associations. Let *E* be the executable context associated to the action *A* in the process model, *S* its situation and P_s the set of product parts defined for the situation. Then P_i must be a subset of P_s.

Only if both checks are successful, i.e. if the input and output parameters defined in the tool model correspond with the process definitions, the tool category is assigned to the executable context. Otherwise, the method engineer is informed about the mismatch.

4.4.2 Relating Tool Categories and Choice Contexts

Each choice context has to be related to exactly one tool category by an *provides_choice_context* association. Thereby the tool category assigned to the choice context is made responsible for performing the choice context, i.e. a new guidance service is defined for the tool category.

In addition, for each context C_j defined as alternative of the choice context *CC* the presentation of the intention *I* related to the context C_j has to be defined. Since an intention (e.g. the intention *delete*) can be associated with more than one context (e.g. the context *deleteEntity* and *deleteAttribute*) a context dependent presentation of the intention is required. This is achieved by relating the context C_j to at least one command element using the *display_of_intention* association (figure 2).

Like the relation of an executable context to a tool category, also the relation of a choice context to a tool category can be supported by automated consistency checks:

C3 *assure that the tool category can display all intentions associated with the alternatives of the choice contexts:* Given a choice context *CC* which is related in the process model to a set of alternative contexts *CA*. For each context $C_x \in CA$, the tool category *T* associated (using the provides_choice_context association) with the choice context *CC* must be assigned to at least one command element (via a provides_command association) which is related (using a display_of_intention association) to the context C_x;

C4 *assure that the tool category can display all product parts associated with the situations of all alternative contexts of the choice context:* Given a choice context *CC* for which a set *CA* of alternative contexts is defined in the process model and a set of product parts P_{CA} which subsumes all products related to any situation *S* which is related to a context $C_x \in CA$. If a tool category *T* is associated to the choice context *CC* then for all product parts $P_i \in P_{CA}$ a displays relation between P_i, a graphical shape *G*, and *T* must exist.

4.5 Environment Meta Model: Summary

In summary, representing both processes and tools at a conceptual level supports the method engineer in assigning the required tool functionality to the method definitions. Moreover, the above mentioned consistency constraints assure correct assignments in the environment model.

Service integration is achieved by the environment model through the relation of executable and choice contexts defined in the process model to the tool definitions.

The *feedback required* after context execution is inherently defined by the context types (in the case of executable contexts as output product types; in the case of choice contexts the contexts as alternatives).

The foundation for the *invocation of method fragments* is established by the fact that a plan context can be related as an alternative to a choice contexts. This makes the tools aware of plan context definitions, i.e. the tools get to know the plan contexts which can be activated in a given process situation (illustrated in section 5.2)

Vice versa, the environment model empowers the enactment domain to invoke the tool responsible for executing a choice or executable context whenever such a context becomes active during the enactment of a plan context.

Last but not least, the definition of the graphical and interaction capabilities for the contexts lays the foundation for the *adaptation of tool behavior* (illustrated in section 5.1).

The interpretation of the environment model at run-time by the three main components of our implementation framework, namely the *tools, the enactment mechanism and the control integration mechanism*, builds the foundation for achieving a process-integrated environment.

5 Implementation and Validation

Based on the contextual process and tool modelling approach outlined in section 4 and the communication protocol defined in [2] we have defined a generic architecture for process-integrated environments which consists of three main components:

- An *enactment architecture* for process engines which interprets the process relevant parts of the environment models. It drives process enactment based on the process knowledge defined in the environment model, deduces the tool category for performing a particular service (context) based on the environment models, handles enactment requests of the performance domain by enacting the requested method fragments, and provides means for an easy integration of existing enactment mechanisms for the interpretation of plan contexts;
- A process-aware control integration mechanism which has been implemented on top of the ToolTalk. It controls message distribution (service requests) based on the interpretation of the environment model.
- A *generic tool architecture* which consists of two major generic subsystems, the *StateManager* and the *ContextManager*. The task of the *StateManager* is to ensure that message exchange with the enactment mechanism is carried out according to the communication protocol. In addition, the StateManager controls and maintains the current state of the tool. State transitions are triggered by user interaction events or the receipt of a message from the enactment domain. The task of the *ContextManager* is assigned to two major subcomponents. The *ContextExecutor* is responsible for adjusting the tool behavior and for providing user guidance according to the environment model (section 5.1). The task of the *ContextMatcher* is to identify predefined method fragments (section 5.2).

All architectural components have been realized as a set of collaborating object-oriented components. The design of the architecture and its implementation is described in detail in [34]. In this paper we focus on the process-sensitivity of the tools and thus sketch the two components of the ContextManager: the ContextExecutor and the ContextMatcher.

5.1 The ContextExecutor

The ContextExecutor controls the execution of choice and executable contexts. Context execution is either initiated by the ContextMatcher or by the StateManager due to the receipt of a context execution message from the enactment domain.

If the execution of an automated service (executable context) is requested, the Context-Executor invokes the tool action associated with the executable context in the environment model, with the situation data as input parameters.

If the execution of a guidance service (choice context) is requested, the ContextExecutor adapts the user interface of the tool according to the definition of the choice context and the current situation data. More precisely, in the command region of the user interface only those menu items and icons are displayed which are associated to an alternative of the choice context. In the product region, all products corresponding to the situation data of the choice context are *highlighted* to draw user attention on them (the entity publication in figure 3). Furthermore, all products which may contribute to a situation of an alternative context are displayed as *selectable* (bright color), whereas all other products part become

unselectable (e.g. the relationship loaned_by in figure 3).

Thus, all contexts (situations and intensions) defined as alternatives of the choice context are displayed to the engineer. On user request, the ContextExecutor displays the pros and cons for each alternative defined in the model in a special guidance window (not shown in figure 3).

As an example for the adaptation of tool behavior, the right part of figure 3 shows an entity relationship (ER) editor which currently executes the choice context `CC_RefineEntity` with the entity type `publication` as current situation data. There are three alternatives defined for this choice context (see left part of figure 3): (1) the executable context *EC_Create_isA_link* by which an IsA-Link between two selected entity types is created; (2) the executable context *EC_Discriminate_Attribute* for creating a discriminating attribute for the selected entity type; and (3) the plan context *PC_SubtypeEntity* for creating subtypes of the selected entity type.

According to the associations specified between the choice context and the command elements provided by ER editor, the intention of the alternative context `EC_CreateIsALink` appears as menu item in the `edit` pull-down menu and as icon in the icon bar using the bitmap `CreateIsALink.xpm` (see figure 3). Similar, the command elements for the other two alternative contexts of the choice context are retrieved from the environment model and displayed to the user (not illustrated in figure 3).

In the product area, the situation data (the entity type `proceedings` of the choice context `CC_RefineEntity`) is highlighted. According to the environment model, all situations of the three alternatives are only based on entity types (not depicted in figure 3). Consequently, the ContextExecutor has marked all other objects as unselectable (displayed in gray), i.e. only entity types (`book, copy_of_publication, user`) are selectable (displayed in white).

5.2 The ContextMatcher

During the execution of a choice context the user selects and deselects product parts and command elements. The task of the ContextMatcher is to compare the current state of the user interface (product parts and command elements selected) with the definitions of the contexts defined as alternatives of the currently active choice context. It compares the commands with the intentions of the alternative contexts and the product parts with the situation of the alternative contexts. The matcher currently applies a best fit approach, i.e. it associates a situation slot with the most specific selected product (part).

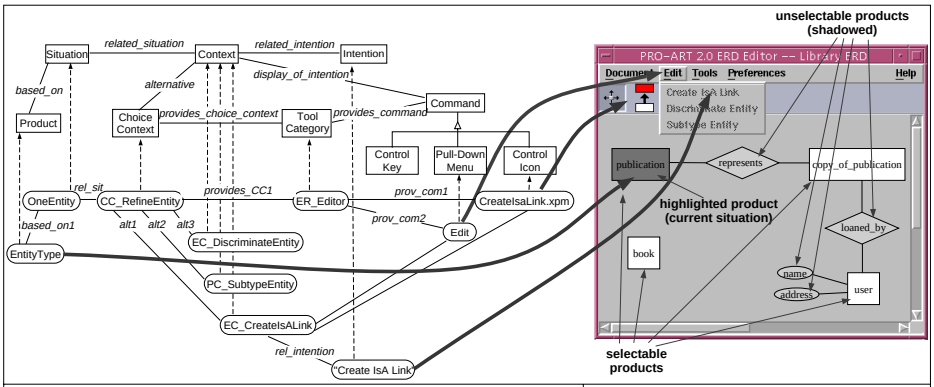

Fig. 3. Influence of the Environment Model on the Tool Behavior.

Whenever the selected product parts and the intentions match with a context definition, the ContextMatcher requests the activation of the context from the StateManager. If the activated context is a plan context the StateManager sends an enactment request (together with the situation data) to the enactment mechanism.

Note that the detection and activation of a new defined plan, executable or choice context can be achieved by just relating the new context as alternative to a choice context or (only in the case of choice contexts) directly to the tool category. The definition of a new method fragment does not require any reprogramming or adaptation of the tools, since the responsible tool automatically displays the intentions and highlights the products. Moreover, it automatically compares selected intentions and product parts with the new context definition and activates the detected context. For example, if the selected product parts and the selected command element match with a new defined plan context, the enactment of the plan context is initiated by the tool.

In the following we illustrate the invocation of a method fragment (figure 4). The ER editor is in the choice context `CC_RefineEntity`. The user has selected the two entities `publication` and `journal` and the menu command associated with the intention `Create IsA Link`. The ContextMatcher compares the selected product parts and the intention associated with the selected command element with the context definition subsumed in the environment model. For efficiency reasons the matching is performed whenever a command element (intention) has been selected by the user.

The ContextMatcher detects that the selected items match with the definition of the executable context `EC_CreateIsALink` defined as alternative of the active choice context (see left part of figure 4).

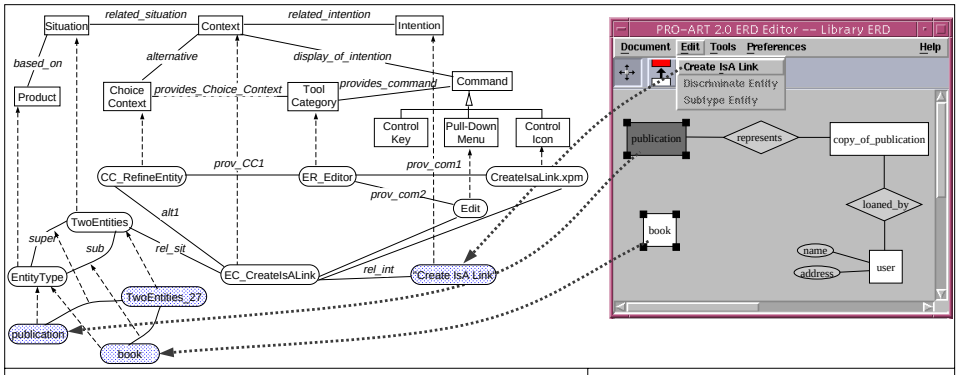

Fig. 4. Matching a Context.

5.3 Validation

We have validated the integrated process and tool modelling approach and the resulting implementation framework by building two prototypical process-integrated environments: PRO-ART 2.0, a requirements engineering environment, and TECHMOD, an environment for supporting the construction of simulation models for chemical plants. Both environments have, in addition, be applied in small case studies performed by students and in the case of TECHMOD by chemical engineers. Conclusions could be drawn from three different perspectives:

Application engineer's view: Most users reported that the fine grained method support provides very helpful guidance. The reflection of the methodical guidance in the behavior of

the tools was regarded as a major advantage, especially the adaptation of the user interface according to the method definitions and the support provided for invoking predefined method fragments even if the user is not aware them.

Method engineer's view: It turned out that defining method fragments using the three context types offers significant advantages. First the three context types provide a guideline how to structure method models. Secondly, the method engineer is forced to make decision points explicit (by defining choice contexts), Thirdly, the three context types imply that the tool definitions are considered adequately during process definition. Fourthly, adaptation of method guidance can be achieved by just changing the model definitions and it can be assured that the application engineer is always aware of the actual method definition. This is essential especially in settings, where new methodical knowledge about good process performance is constantly elicited and learned.

Tool builder's view: From this viewpoint, the main result was that the building blocks of the integrated environment meta model were sufficient for defining tools. Moreover, the developers were forced to define process knowledge explicitly in plan contexts instead of embedding it in the tools, i.e. the "process in the tool syndrome" [11] was avoided. The implementation of twelve process-sensitive tools for the two PIEs was significantly facilitated by the generic tool architecture and the reuse of the generic implementation framework.

6 Related Work

Whereas the main focus of our approach is to establish a process integration of the tools used in a PIE, most research contributions do not consider the integration of tools in PCEs although the problems of a-posteriori integration of existing CASE tools have been widely recognized (e.g. [35,11,36,15]). Recent publications therefore argue that "a posteriori tool integration (e.g. by means of wrappers) could be less effective since a tool is still seen as a monolithic 'operator' " [4]. Consequently, existing PCEs do not offer process-integrated tools.

An exception, where partial process-integration of the tools is offered, is the GTSL approach [12] developed within the GOODSTEP project [37] which aims at the *generation* of specific tool services, schemata, and consistency checks from tool specifications which are coupled with process models. This approach mainly provides solutions to the service and data integration problem, but does not provide means for the invocation of method fragments nor for the dynamic adaptation of the tool behavior according to the process definition and the enactment state.

Meta-CASE environments or CASE shells like MetaEdit [38] or Phedias [39] are based on the generation of tools according to a (meta model) specification. They focus mainly on notational and ontological aspects, but lack process-orientation (see also [40]).

Existing process modelling languages almost neglect the definition of tools, although some provide low-level constructs for the invocation of foreign programs like black transitions in SLANG [4], the `call`–Statement in Marvel [5], or the binding of abstract process operators to tools during process instantiation in ALF [41]. On the other hand, control-oriented tool integration approaches like FIELD [22] and its commercial derivatives (such as HP's BMS [23] and Sun's ToolTalk [24]) as well as object-oriented distribution infrastructures like CORBA [25] or OLE [42] store tool (service) descriptions in their message servers/object brokers, but do not consider the process models.

As a consequence, if at all, tool and process models coexist in the message server repository and in the process repository without a systematic approach for assuring consistency.

Many PCE approaches like SPADE [4], Process WEAVER [9], EPOS [43], and Merlin

[21] employ such mechanisms for invoking tool services, although the tool model used by the message server and the process model used by the enactment domain are not systematically integrated. In addition, tool invocation in existing PCEs is restricted to atomic actions while user guidance by adapting the accessible objects and operations through guidance services is not systematically supported; i.e. the tools of these environments are not process-sensitive.

In contrast, our approach takes a different stance on process and tool modelling. Among others, it offers a uniform way to express enactment and tool services by which service and data integration between the enactment and performance domains can be achieved, and the foundation for process-sensitive tools is established.

The FIELD-based Forest environment [44] is an attempt to establish a central description of processes and tools. Forest extends the tool-related message distribution patterns stored in the message server by so-called policy descriptions which can be regarded as primitive process definitions. Although this approach improves the integration of tool and process models it provides no means for establishing process-sensitive tools and for supporting the invocation of method fragments.

In summary, the problem of tool integration in PCEs was recognized and some partial solutions to the problem exist. So far, no comprehensive approach was proposed which establishes process integration of tools and, on the other hand, enables the humans performing the process to play a more active role by initiating the execution of method fragments.

7 Conclusions

Our contextual approach for fine grained and adaptable method support in process centered environments presented in this paper is based on the requirements for the process-integration of interactive engineering tools elaborated in section 2. The consideration of those requirements have led to a significant improvement of our previous solutions for process-integrated tools described in [2].

To meet the requirements we argued that from a modelling perspective *tools should no longer be treated as second class citizens*. Instead tools, as the processes, have to be explicitly defined. We presented a contextual meta model for defining the process (method) support and a *tool meta model* for defining the basic capabilities of the interactive engineering tools used to perform the processes. We then proposed to relate the available tool support with the method definition. The interrelation of the tool and method definitions can be supported by automated consistency checks, e.g. it can be checked if a tool is able to perform the associated services (executable and choice contexts) defined in a method fragment.

The integration of the tool and the method definitions forms an *environment model*. Based on an interpretation of the environment model *process-sensitive tools* are established. The tools are able to *adapt themselves automatically* according to the method definitions. Moreover, the engineering *tools can initiate the enactment of predefined method fragments* by matching user interactions and method definitions.

Our overall approach has been validated by implementing a *generic architecture for process-integrated engineering environments* which consists of three main components (enactment architecture, tool architecture, and process-aware control integration mechanism). Each of these components interprets the environment model and thereby assures that the environment “behaves” according to the method definitions.

These applications have shown that our approach significantly facilitates the adaptation of fine-grained method support to organizational and project specific needs as well as the

integration of new method fragments. For example, the approach has proven very useful for defining and enforcing project specific trace services, i.e. to guide the engineers in capturing trace information in accordance to a contract [45].

Future research is concerned with the detection and the execution of choice contexts across tool boundaries. Moreover we plan to validate the TECHMOD environment in an industrial setting and investigate in the generation of basic executable services based on model definitions, like the ones applied in the MetaEdit+ CASE shell [38] or in GTSL [12].

Acknowledgments: The authors like to thank their colleagues R. Klamma, R. Dömges, P. Haumer and M. Jarke for many fruitful discussions and comments on early versions of the paper. Without the enthusiasm of our students S. Brandt, S. Ewald, M. Hoofe, T. Rötschke, K. Schreck, W. Thyen the implementation of the generic architecture and the PRO-ART 2.0 and TECHMOD environments would not have been possible.
This work was in part founded by the European Community under ESPRIT Reactive Long Term Research 21.903 CREWS and by the DFG Project 445/5–1 "Prozeßintegration von Modellierungsarbeitsplätzen".

References

[1] M. Dowson. Consistency Maintenance in Process Sensitive Environments. In *Proc. of the Process Sensitive Software Engineering Environments Architectures Workshop*, Boulder, Colorado, USA, Sept. 1992.
[2] K. Pohl. *Process Centered Requirements Engineering*. RSP marketed by J. Wiley & Sons Ltd., England, 1996.
[3] J. Lonchamp. An Assessment Exercise. In A. Finkelstein, J. Kramer, and B. Nuseibeh, editors, *Software Process Modelling and Technology*, pages 335–356. RSP, London, 1994.
[4] S. Bandinelli, E. Di Nitto, and A. Fuggetta. Supporting Cooperation in the SPADE-1 Environment. *IEEE Transactions on Software Engineering*, 12(12):841–865, 1996.
[5] N. Barghouti. Supporting Cooperation in the MARVEL Process–Centered Software Development Environment. In *Proc. of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, pages 21–31, New York, New York, USA, 1992.
[6] W. Deiters and V. Gruhn. The FUNSOFT Net Approach to Software Process Management. *Intl. Journal of Software Engineering and Knowledge Engineering*, 4(2), 1994.
[7] T. Mochel, A. Oberweis, and V. Sänger. Income/star: The petri net simulation concepts. *Journal of Mathematical Modelling and Simulation in Systems Analysis*, 13:21–36, 1993.
[8] P. Heimann, G. Joeris, C.-A. Krapp, and B. Westfechtel. DYNAMITE: Dynamic Task Nets for Software Process Management. In *Proc. of the 18th Int. Conf. on Software Engineering*, pages 331–341, 1996.
[9] C. Fernström. Process WEAVER: Adding Process Support to Unix. In *Proc. of the 2nd International Conference on Software Processes*, pages 12–26, Los Alamitos, CA, USA, 1993.
[10] A. I. Wasserman. Tool Integration in Software Engineering Environments. In F. Long, editor, *Proc. of the Intl. Workshop on Software Engineering Environments*, pages 137–149, Berlin, Germany, 1990. Springer-Verlag.
[11] C. Montangero. The Process in the Tool Syndrome: is it becoming worse? In *Proc. of the 9th Intl. Software Process Workshop*, pages 53–56, Arlie, Virginia, USA, Oct. 1994. IEEE Computer Society Press.
[12] W. Emmerich. *Tool Construction for Process–Centred Software Development Environments based on Object Databases*. PhD thesis, University of Paderborn, Paderborn, Germany, 1995.
[13] M. Dowson and C. Fernström. Towards Requirements for Enactment Mechanisms. In B. Warboys, editor, *Proc. of the 3rd Europ. Workshop on Software Process Technology*, number 772 in LNCS, pages 90–106, Villard de Lans, Frankreich, Feb. 1994. Springer-Verlag.
[14] C. Fernström. State Models and Protocols in Process Centered Environments. In W. Schäfer, editor, *Proc. of the 8th Intl. Software Process Workshop*, pages 72–77, Wadern, Germany, Mar. 1993. IEEE Computer Society Press.
[15] C. Fernström and L. Ohlsson. Integration Needs in Process-Enacted Environments. In *Proc. of the 1st Intl. Conf. on the Software Process*, pages 142–158, 1991.
[16] I. Thomas and B. A. Nejmeh. Definitions of Tool Integration for Environments. *IEEE Software*, 8(2):29–35, 1992.
[17] ECMA-NIST. *A Reference Model for Frameworks of Software Engineering Environments*. Number TR/55 Version 3. ECMA & NIST, 1993.
[18] G. Valetto and G. E. Kaiser. Valetto, g. and e. kaiser, g. In *Valetto, G. and E. Kaiser, G.*, pages 40–48, July Valetto, G. and E. Kaiser, G.
[19] K. Pohl, R. Dömges, and M. Jarke. Decision Oriented Process Modelling. In *Proc. of the 9th Intl. Software Process Workshop*, pages 124–128, Arlie, Virginia, USA, Oct. 1994. IEEE Computer Society Press.
[20] N. S. Barghouti and B. Krishnamurthy. Using event contexts and matching constraints to monitor software processes. In *Procs 17th Intl. Conf. on Software Engineering, Seattle, Washington, USA*,

pages 83–92, May 1995.
[21] G. Junkermann, B. Peuschel, W. Schäfer, and S. Wolf. MERLIN: Supporting Cooperation in Software Development Through a Knowledge–Based Environment. In A. Finkelstein, J. Kramer, and B. Nuseibeh, editors, *Software Process Modelling and Technology*, pages 103–130. RSP, London, 1994.
[22] S. P. Reiss. Connecting Tools Using Message Passing in the FIELD Environment. *IEEE Software*, 4(7):57–67, July 1990.
[23] M. Cagan. The HP SoftBench Environment: An Architecture for a New Generations of Software Tools. *Hewlett–Packard Journal*, 41(3):36–47, June 1990.
[24] SunSoft. The ToolTalk Service (White Paper). Technical report, SunSoft Inc., June 1991.
[25] OMG. *CORBA: Architecture and Specification*. Object Management Group, Inc., 1995.
[26] L. A. Suchmann. *Plans and Situated Actions: The problem of human machine communication*. Press Syndicate of the University of Cambridge, 1987.
[27] R. Stallman and G. Sussman. Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis. *Artificial Intelligence*, 9(2):135–196, 1977.
[28] V. Dhar and M. Jarke. On modeling processes. *Decision Support Systems*, (9):39–49, 1993.
[29] C. Potts. A Generic Model for Representing Design Methods. In *Proc. of the Eleventh Intl. Conf. on Software Engineering*, Pittsburgh, PA, May 1989.
[30] G. Fischer. Integrating Construction and Argumentation in Domain-Oriented Design Environments. In *Proc. of the First Intl. Symp. of Requirements Engineering*, page 284, San Diego, CA, Jan. 1993. IEEE Computer Society Press.
[31] M. Jarke, K. Pohl, C. Rolland, and J.-R. Schmitt. Experience-Based Method Evaluation and Improvement: A Process Modeling Approach. In *IFIP WG 8.1 Conference CRIS '94*, Maastricht, Netherlands, 1994.
[32] C. Rolland and N. Prakash. Reusable Process Chunks. In *Proc. of the Intl. Conf. Database and Expert Systems Applications*, Prague, Slovakia, Sept. 1993.
[33] C. Rolland and G. Grosz. A General Framework for Describing the Requirements Engineering Process. In *Proc. of the Intl. Conf. on Systems, Man, and Cybernetics*, San Antonio, Texas, USA, Oct. 1994. IEEE Computer Society Press.
[34] K. Pohl, R. Klamma, K. Weidenhaupt, R. Dömges, P. Haumer, and M. Jarke. A Framework for Process-Integrated Tools. Technical report, RWTH Aachen, 1996.
[35] A. Fuggetta and C. Ghezzi. State of the Art and Open Issues in Process-Centered Software Engineering Environments. *Journal of Systems and Software*, 26:53–60, 1994.
[36] M. Anderson and P. Griffiths. The Nature of the Software Process Modelling Problem is Evolving. In *Proc. of the 3rd European Workshop on Software Process Technology, EWSPT '94*, LNCS 772, pages 31–34, 1994.
[37] GOODSTEP-Team. The GOODSTEP Project: General Object-Oriented Database for Software Engineering Processes. In *Proc. of the Asia-Pacific Software Engineering Conference*, pages 410–420, Tokyo, Japan, 1994.
[38] S. Kelly, K. Lyytinen, and M. Rossi. MetaEdit+ — A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment. In *Proc. of the 8th Intl. Conference on Advanced Information Systems Engineering*, LNCS 1080, pages 1–21, Heraklion, Crete, Greece, 1996.
[39] X. Wang and P. Loucopoulos. The Development of Phedias: a CASE Shell. In *Proc. 7th. Int. Workshop on CASE, Toronto, Canada*, pages 122 – 131. IEEE Computer Society Press, 1995.
[40] P. Marttiin, K. Lyytinen, M. Rossi, V. Tahvanainen, and J.-P. Tolvanen. Modeling requirements for future CASE: Issues and Implementation Considerations. *Information Resources Management Journal*, 8(1):15–25, 1995.
[41] G. Canals, N. Boudjlida, J.-C. Derniame, C. Godart, and J. Lonchamp. ALF: A Framework for Building Process-Centred Software Engineering Environments. In A. Finkelstein, J. Kramer, and B. Nuseibeh, editors, *Software Process Modelling and Technology*, pages 153–186. RSP, London, 1994.
[42] K. Brockschmidt. *Inside OLE, Second Edition*. Microsoft Press, Redmond WA, 1995.
[43] R. Conradi, M. Hagaseth, J.-O. Larsen, M. Nguyen, B. Munch, P. Westby, W. Zhu, M. Jaccheri, and C. Liu. EPOS: Object–Oriented Cooperative Process Modelling. In A. Finkelstein, J. Kramer, and B. Nuseibeh, editors, *Software Process Modelling and Technology*, pages 33–70. RSP, London, 1994.
[44] D. Garlan and E. Ilias. Low-cost, Adaptable Tool Integration Policies for Integrated Environments. In *Proc. of the 4th ACM SIGSOFT Symposium on Software Development Environments*, volume 15, 1990.
[45] K. Pohl, R. Dömges, and M. Jarke. Towards Method-Driven Trace Capture. In *Proc. of the 9th Intl. Conf. on Advanced Information Systems Engineering*, Barcelona, Spain, June 1997.