Change Management Needs
Integrated Process and Configuration Management

Gregor Joeris

Intelligent Systems Department, Center for Computing Technology
University of Bremen, P.O. Box 330 440, D-28334 Bremen
joeris@informatik.uni-bremen.de

Abstract. Change management is a core problem of software development. Management of changes means managing the process of change as well as managing all artifacts of an evolving software system. Both challenges have been focused extensively in the field of software process modeling and software configuration management, respectively. In this paper, we motivate that change management needs an integrated approach to process and configuration management. A fundamental prerequisite to provide such a comprehensive support is the integration of the underlying representation formalism of process and version models. To cope with this problem, we propose a conceptual framework to provide a common conceptual basis of process and version modeling concepts and identify the requirements for comprehensive process support for change management. Furthermore, we classify and evaluate different approaches of how to integrate process and version models from a conceptual point of view.

1. Introduction

Change management is one of the core problems of software development. Management of changes to any software document means (1) managing the process of change as well as (2) managing all artifacts of an evolving software system. Both aspects serve as motivation for an integrated modeling approach to support process and configuration management in a process-centered software engineering environment (PSEE):

(1) Focusing the *software development process* has been always a core challenge of software engineering. At the beginning, very coarse-grained and informal descriptions of the software process in terms of life-cycle models were considered and have been criticized ever since. In the last decade, the explicit and unambiguous specification and continuous improvement of processes were recognized as a key factor for successful completion of software projects ([CFFS92]).

The software process is characterized - like every engineering process - by the complexity of both, the product and the process, and the dynamically changing envi-

ronment in which the process takes place. Thus, *feedbacks*[1] and changes occur in all life cycle phases. Though many process modeling languages (PML) were developed and feedbacks in the software process have been recognized as an important factor of *software process technology* (SPT) (cf. [Dow87]), most PMLs provide only poor concepts for managing the process of change. Moreover, most existing PMLs lack integration of version and configuration modeling mechanisms into the overall representation formalism.

(2) Software configuration management (SCM) has been identified as a major part of a well defined software development and maintenance process ([Hum89]). SCM deals with controlling the evolution of complex software systems and supports both, technical development as well as management of the evolution of software systems ([Fei91, CoWe97]):

The *technical view* of SCM provides mechanisms and functions to control versions of software objects, to build derived versions, and to construct consistent software configurations. On a technical level, it supports performing changes by different mechanisms like workspace control, long transactions, or sub-databases. Many version models and concepts have been proposed ([CoWe97, Cag95]). But all of them lack sophisticated process support, although some process-oriented features were sometimes added. Currently, process support is of growing interest in software configuration management (cf. [Est95, MiCl96, EDA97, Leb97]).

The *management view* of SCM focuses on the organizational and administrative aspects of SCM. As part of the overall management of a software project, SCM provides methods to handle change requests, and to perform changes in a controlled manner by introducing well-defined change processes. Furthermore, it supports monitoring the status of the software components.

It seems that SCM provides all concepts and techniques for a methodical approach to change management. Moreover, change management seems to be just another notion of SCM. But, a detailed and comprehensive process support for implementation and management of changes is not given. On the technical level, support for process automation is sometimes provided to build derived objects and to trigger procedures before or after a SCM activity is done. On the other hand, the management view of SCM considers only very coarse-grained and informal process descriptions with all the drawbacks of such models. Thus, there is a gap of process support between the technical and the management view of SCM.

We believe that an integrated approach to process and configuration management can bridge this gap and is the basis for comprehensive change management support provided by a PSEE. The prerequisite of such an approach is the integration of the underlying process and version modeling formalisms. This paper aims at providing a conceptual framework for the integration of process and version models. It identifies

---

[1] The term feedback is used as a generic term to denote deviations from a linear progression of a process. Similar terms are 'iteration', 'repetition', or 'rework' (cf. [Dow87]).

the core problems and requirements for integrated representation formalisms. In particular, it focuses on the capabilities needed for comprehensive process support for change management. Furthermore, we use the framework to classify approaches of how to integrate process and version models on a conceptual level.

Section 2 introduces the conceptual framework for integration of process and version models and relates it to other work on conceptual frameworks. Further, it describes the main integration aspects of an integrated representation formalism. Section 3 characterizes change processes and outlines the main requirements for integrated process and configuration management in order to support change management. In section 4 existing process and version models are classified and evaluated using the proposed framework. Finally, section 5 gives a short conclusion.

2. Conceptual Integration Framework

2.1 Objectives of the Conceptual Framework and Related Work

Several conceptual and terminological frameworks have been proposed in the field of software process technology for different purposes (e.g., [CFFS92], [FeHu93], [Lon93], [CFF93], [CoLi95]). Some aim at providing an unambiguous and generally accepted terminology, some characterize and clarify the basic challenges, requirements and concepts, and some are designed for comparison and evaluation of existing approaches. All of them provide valuable knowledge for an essential understanding about the software process and are fruitful for developing both, a sophisticated theory and technology concerning software processes. In relation to our conceptual framework, the following work is of particular interest:

In [CFF93], a framework for evolving software processes is introduced. Process change and the process of process evolution is a specific topic concerning changes which is not focused on in this paper. Some requirements of evolving processes may be adopted and adapted for handling general changes of software documents.

Conradi and Liu discuss in [CoLi95] whether a process modeling language should be designed by using one or many languages for the different sub-models of a process model. They define four classes of PML design approaches. Furthermore, they distinguish six core process elements (activities, artifacts, production tools, roles, humans, model evolution) and take versioning as a non-core element into account. The aim of the paper is to clarify the basic concepts, functionality and tool architecture of a PSEE with respect to the various PML design approaches. A deep consideration of integration aspects for process and version models is not provided.

Our conceptual framework takes only a subset of the software process modeling domain into account, namely the process space, the product and version space, and the work environment which are introduced below. Thus, it neither claims to be a general framework for software process technology, nor is it intended to provide a precise terminology. Rather, the aim of our conceptual framework is to integrate concepts from software process technology and SCM to provide the conceptual basis for

studying change processes and to identify the integration dimensions of process and version models. Thus, it claims to be adequate for clarifying and improving the understanding of the interdependencies between modeling concepts of these research areas. Moreover, it serves as the basis for the characterization of different integration aspects and is used for the evaluation and classification of existing approaches.

Interrelationships between process and configuration management have been discussed by the SCM community mainly with respect to cooperation support (e.g. [GCCM95, EDA97]) or low-level process control and automation rather than focusing on all integration aspects. A deep consideration of these interrelationships, particularly with high-level process concept, is currently of growing interest.

2.2 Fundamentals of the Conceptual Framework

With the intention of considering process and version modeling formalisms in mind, we have to deal with different terminologies from SPT and SCM. In particular, a *process model* usually combines descriptions of different perspectives of a process, namely activities, artifacts, resources, organizational units, and execution behavior (cf. [CFFS92, Lon93]), but the artifact (sub)model of existing PMLs takes only the product space into account. On the other hand, a *version model* consists of a *product space* which describes the software documents[2] and their relationships and a *version space* which identifies and organizes the object's versions.

In order to define a common terminological basis and to avoid this overlapping meaning of process and version model, we adopt and integrate the basic notions from SPT and SCM: First, we adopt the notions repository, product, version, and workspace from SCM as defined below. Next, we use the notion process space to denote the active parts of a process and hence use it in a restricted sense rather than covering all aspects of a process.

Our framework is based on the notions of process space, product space and version space. Figure 1 illustrates these perspectives and their interdependencies and additionally shows the resource and organization perspectives which will be disregarded in the following. All these spaces together define the modeling scope of an integrated representation formalism. Moreover, Figure 1 illustrates the work environment which defines the virtual world in which processes are performed by agents. The building blocks of the framework are defined as follows and their integration aspects are discussed in the following sub-section.

The *process space* defines what tasks have to be done and how these tasks may be accomplished in terms of a partially ordered set of process steps. Furthermore, it specifies the dynamic execution behavior (execution states, control and data flow) of process instances. With respect to change management we emphasize two aspects, namely (a) different forms of feedbacks and change processes and (b) process support for SCM as part of a change process is needed on a low-level of abstraction for the technical side as well as on a high-level of abstraction for the management side.

---

[2] We use the notions document, artifact, and software object synonymously throughout this paper and use change as a general term for modification of any software document.
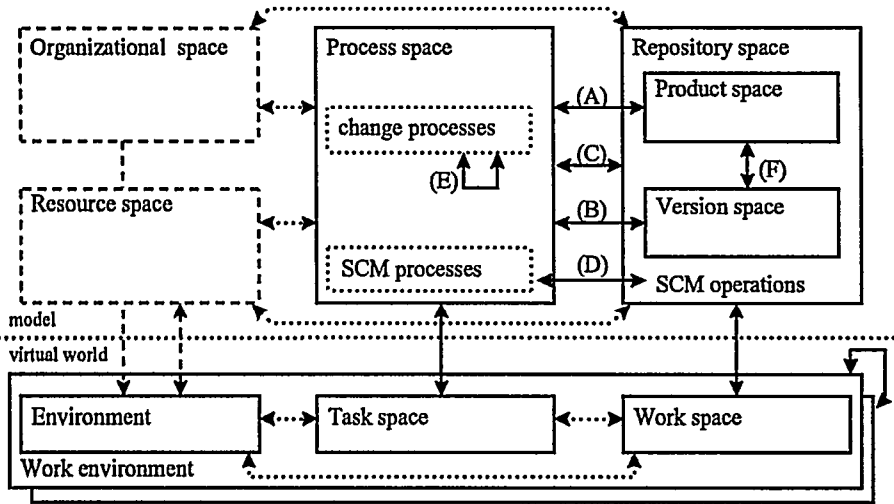
Fig. 1. Overview of the conceptual framework

The *repository space* consists of the *product space* and *version space* and provides the fundamental procedures for defining, identifying and accessing of versions, variants, and configurations ([CoWe97]). The *product space* describes the structure of a software product without taking versioning into account. It defines the composition of software documents/objects and their dependencies. Software objects are coarse-grained units which have an internal fine-grained structure. The *version space* defines how versions are represented and managed. In general, version representations may be classified into state-oriented versioning and change-oriented versioning. The internal representation is encapsulated by basic operations for retrieving and constructing of versions. Note, that we do not subsume capabilities for low-level process control under the repository space which are often associated with SCM (cf. [Leb94]). To provide an orthogonal and well-defined conceptual framework, these capabilities are part of the process space.

The *work environment* provides the virtual environment in which an agent (human or machine) perform its activities. It is divided into the workspace, task space, and the technical and organizational environment: The *workspace* provides all software documents that the agent needs to do the work including temporary results. The *task space* contains the personal work list, provides enactment support (guidance, enforcement, automation) and handles process feedback information [DoFe94]. Thus, the task space is the activity-oriented equivalent to the data-oriented notion workspace. Finally, the technical and organizational *environment* is derived from the resource model and organization model and should be customized to the user-specific needs. E.g. it provides uniform invocation of tools.

The work environment controls and maintains the task and workspaces in conjunction with the process and configuration management services and allows for interop-

erability among different work environments. Different concepts exist for workspace management: E.g., a workspace may be either a partial or total copy of the repository space (e.g. implemented by check-in/out), or a specific view on it (e.g. implemented by virtual file systems, sub-databases, or long transactions) (cf. [EsCa94, Fei91]). Thus, it may be more or less under the control of the PSEE's repository.

2.3 Integration Aspects of Process and Version Models

After defining the building blocks of the conceptual integration framework, we now identify the interrelationships between process space and repository space. These interdependencies are denoted in figure 1 by (A) - (F). We outline the fundamental integration aspects which have to be considered when integrating process and version models on a conceptual level. Note, that every modeling space may be described on various levels of abstraction (type, template and instance), and hence, the integration of the modeling elements have to be considered on every level.

A. Interrelationships between process space and product space

a) *Interdependencies between process and product structure:* The process and product structure, i.e. the (de)composition of processes/documents and their (static) dependencies, have great interdependencies. E.g. the software architecture determines a lot of tasks to be accomplished and module dependencies define data and control flow dependencies. Managing these interdependencies is fundamental for an integrated modeling approach and in conjunction with the management of design decisions the basis for impact analysis.

b) *Formal input/output-relationship:* From a functional point of view, a process takes software objects as inputs and produces new or modified objects as output. From the product point of view, software documents are used, created and modified by operations or processes. In any case, there is a (static) relationship between the abstract process definition and product definition that determines which kind of input and output a process may consume or produce, respectively. This relationship further determines several properties of the actual dataflow (see below) and is more complex than in conventional programming languages: (1) kind of parameter (input, output, in/out), (2) kind of data passing (original, copy, reference), (3) kind of access (exclusive, shared, read-only, write), (4) cardinality of parameter (optional or mandatory, unique or multiple values), and (5) parameter type (restricted by document/object type and/or by a document specified in the product space which acts as a placeholder for the actual version).

B. Interrelationships between process space and version space

a) *Actual data flow:* Different kinds of actual data passing and access have to be taken into account. In particular, several versions may be passed between running activities. Thus, the actual data flow has to take versioning into account. In general, the logical data flow of the process space must be separated from the physical passing of versions using workspace management capabilities in order to realize different data flow scenarios (e.g. push vs. pull).

b) *Interdependencies between the version state and the process' execution behavior:* As the product structure influences the process structure, the actual version state may affect the execution behavior of a process. E.g. a process must not be started if the version is not approved. Furthermore, version state transition may be associated with specific activities.

c) *Concurrency control:* Processes which operate concurrently on the same version have to be coordinated using an adequate control mechanism which depends on the employed workspace management approach. This implicit coordination provided by a SCM system may affect the scheduling of processes under the control of a process engine which is in charge of explicit coordination of activities.

C. *Interrelationships between process space and repository space*

a) *Reflexivity:* Obviously, process specifications are documents; due to several reasons process models will evolve in time. Thus, process models are part of the repository space and will be under version control. Moreover, the process of process (model) changes (usually referred to as meta process) is defined in the process space. Therefore, process representation formalisms have to provide reflective features [BaFu93]. Note, that for the sake of readability this reflexivity relation was not accurately illustrated in figure 1.

D. *Interrelationships between change processes and SCM operations*

a) *Integration of basic SCM operations in the process space:* A version model defines not only the structure of the product and version space. It also provides basic operations to build up and modify these spaces (e.g. retrieving and constructing of versions, building of derived objects). These basic procedures have to be integrated into the general PML used for the process space.

b) *Support for general SCM processes:* For managing changes, process support is needed for SCM activities on different levels of abstraction (see Sec. 3.2).

E. *Intrarelationships in the process space*

a) *Correlation of change processes:* Many separately and independently initiated but correlated change processes which concern similar problems have to be synchronized during execution and may have to be joined together.

b) *Transactional and non-transactional* activities have to be integrated following different strategies of how to react on changes.

F. *Interrelationships between product space and version space:* The integration of product space and version space is essential for a versioned repository. In particular, construction of configurations and building of derived objects have to be investigated with respect to change management. In this paper we do not further address this topic. For a comprehensive overview see [CoWe97].

3. Process Support for Change Management

SCM deals with changes on both, technical and management level, but it lacks comprehensive process support for software engineers and managers, as mentioned in the introduction. In the following, we focus on changes from a process-oriented view in order to outline the main requirements of an integrated approach to process and configuration management. First, we focus on feedbacks in a software process from a technical point of view. Next, we describe the process support needed for SCM activities. Finally, we briefly sketch the cooperation support needed for actors who perform changes.

3.1 Feedbacks and Change Processes

The notion feedback was employed to subsume notions like backtracking, repetition, iteration, rework and so on (cf. [Dow87]). This variety of notions already shows that there will be no general concept for handling changes in a PSEE. The following list gives an overview of such situations (cf. [HKNW96]) and defines some of the above mentioned notions:

- *Loops*: predefined repetition with defined exit condition until an accurate result is accomplished; no analyzing or planning phase. Example: combination of manual and automatic process steps until an accurate result is accomplished
- *Rework*: predictable or unpredictable repetition of finished tasks or activities reusing the previous process and resources; additionally, a change request is documented, analyzed and authorized and the change will be validated and approved following general configuration management policies (cf. [Hum89]); but, no planning of activities and resources is needed. Example: iteration of coding, testing and reviewing a software module due to error correction
- *New process*: instead of repetition of the previous process, a new process is instantiated for implementing and managing the change which consists of analysis and authorization of change, planning the activities for change, implementation and approval; the process may be as complex as the overall development process. Example: modification of a system which does not fulfill the user's needs.
- *Reaction*: reaction to change requests and adaptation of previous results during enactment by the agent who currently performs the task (simultaneous engineering). Example: during module implementation new features may be added.
- *Backtracking/Rollback*: undoing previous actions and proceeding differently which is useful for automated process steps with transactional semantics.
- *Prototyping*: predefined iterations of processes on type level with corresponding iterated evolution of actual process instances and their results.

Furthermore, the situations of feedbacks in the software process sketched above are even more complicated since many possibly dependent or overlapping feedbacks may occur concurrently. Process support for feedbacks have to take this diversity of change situations into account. Thus, the core requirement with respect to the process modeling and enacting capabilities of a PML is the *adaptability* of the execution behavior specification which includes:

- *Modeling and initiating of feedbacks*: Flexible modeling mechanisms have to be provided in order to determine the target of a feedback and to define how to react to this situation. In particular, predefined control flow constructs like loops have to be combined with ad hoc decisions where to restart a process. Instead of reactivation of finished activities, in some situations new activities have to be dynamically initiated and integrated into the process for handling the changes.
- *Impact analysis and consequences*: After initiating a feedback, we have to consider who is potentially affected, what could be the consequences, and how should one react. E.g. transactional activities should be aborted whereas human activities may be suspended or even informed.
- *Managing the flow of change:* When actual changes are done, we have to decide when these changes should be propagated to the affected process steps. E.g., to support simultaneous engineering, this can be done before the activity is finished. Thus, advanced data and control flow mechanisms are needed and have to be integrated with workspace management services for propagation. Furthermore, changes may lead to on-the-fly modifications of the planned activities.

The need for a flexible and adaptable execution behavior induces further requirements. We emphasize the following three derived requirements: (1) On-the-fly modifications and local customization of enacting process instances to react to unforeseen situations (cf. [CFF93]), (2) coordination and joining of correlated change processes, and (3) support for transactional workflows ([GHS95]).

3.2 Process Support for Configuration Management

For change management, process support for SCM activities are needed on a low-level of abstraction for the technical side as well as on a high-level of abstraction for the management side. To bridge this gap we need capabilities for:
a) *Process control and process automation*: the process space should be able to control the basic SCM operations, i.e. who, when and where changes can be made (e.g. in conjunction with a role concept modeled in the resource space); moreover, mechanisms for automatically triggering actions should be provided and integrated in the overall process modeling capabilities, e.g., for building derived objects, for constructing or updating a workspace, etc.
b) *Integration of management processes and technical processes*: processes for implementation of changes have to be integrated with management processes of SCM like change request authorization or approval of changes. This requires special modeling features (reflexivity, see section 2.3) and monitoring capabilities. Furthermore, product-centered processes based on state-transition diagrams for controlling the product life-cycle have to be supported (cf. [EDA97]).
c) *Modeling of global SCM policies*: establishing and controlling general SCM policies (e.g. for quality assurance) and customization to the organization's existing policies require modeling capabilities for global process rules or constraints
Furthermore, rather to provide traceability only for the version history, an integrated approach has to be extended to include the process history as well.

3.3 Work Environment Support for Performing Changes

Support for performing changes has to deal with the trade-off of "sharing vs. isolation" ([Cag95]). On the one hand, software engineers want to perform changes isolated from influences of other changes as much as possible. On the other hand, they want to be notified about all changes which affect their work and want to obtain these new results as early as possible. Thus, the right degree of cooperation between various work environments defined as inter work environment data exchange is essential. We need support for both, controlled cooperation and free cooperation:

Controlled cooperation support deals with notification of changes and propagation of intermediate and final results. It may be explicitly controlled on process level by passing of intermediate results or may be implicitly controlled on data level by extended transaction protocols with relaxed isolation properties depending on the applied workspace management concept. Furthermore, we have to distinguish between simultaneous changes of versions of dependent objects and concurrent changes to the same version of an object. The latter further requires reconciliation of concurrently changed versions.

Free cooperation support deals with direct interaction among engineers which is not fully controlled by the process engine or repository (e.g. shared applications, video conferencing, etc.). The concept of shared workspaces seems to be a good approach in this direction: On process level, we can control which activities may share parts of their workspaces, whereas the actual data interchange is managed by the actors (cf. [EDA97]). Finally, we need synchronization mechanisms to integrate synchronous cooperating activities into a predefined workflow on process level.

Another important service of work environment management is the configuration of the workspace which has to support some form of *'recovery'* for change management: To implement a change, the old work environment must be re-established and adapted, i.e. at least old and new input versions, the previously produced results, and the modified task description or the change request have to be provided.

4. Classification of Process and Version Model Integration

Representation formalisms of both, process and version models vary between several paradigms (cf. [FNK94], [CoWe97]). Thus, existing approaches of integrating process and version models can only be compared and classified on an abstract and conceptual level. To achieve this goal, we apply meta modeling techniques in conjunction with our conceptual framework, i.e. we neglect all details and use an ER-like notation to sketch a model of the underlying concepts which will be embedded in the conceptual framework. In addition, we give examples and briefly evaluate some of the few existing integrated approaches to process and configuration management and point out their strengths and weaknesses. Most of them are descended from configuration management systems extended by process-oriented features.
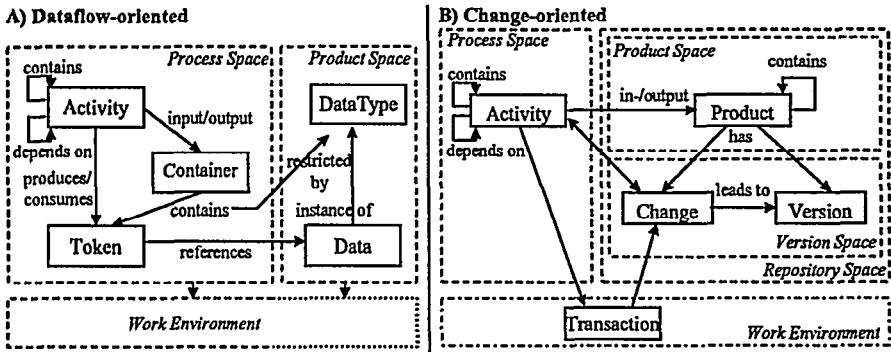
Fig. 2. Activity-centered integration approaches

First of all, we distinguish between activity- and product-centered integration approaches. Furthermore, we divide the activity-centered approaches into dataflow-oriented and change-oriented, illustrated in Figure 2. The product-centered approaches are classified into product-state-based and object-based approaches, illustrated in Figures 3.

4.1 Activity-centered Integration

A) Dataflow-oriented: The dataflow-oriented approach is based on activities and information flows between them. Furthermore, it is characterized by a clear separation of processes and products. In the process model, the dataflow is represented by tokens which refer to any product or data item and hence abstract from their representation, as illustrated in Figure 2. Thus, no assumption is made about the underlying object model so that every object model (versioned or not) may be integrated. Another advantage is that feedbacks and change processes can be supported on a high-level of abstraction. On the other hand, only a very loose-coupled integration can be achieved. E.g., since versioning cannot be presumed, dataflow and execution behavior do not take versioning into account. Moreover, most often only poor work environment and cooperation support is provided by this approach.

Examples of this approach are Petri-net based or activity-centered process management approaches like *Spade* [BFGL94] or DYNAMITE [HJKW96]. The latter is particularly designed to cope with the dynamic nature of software processes, and provides some remarkable capabilities to support change processes. It provides specific mechanisms for feedback handling and supports cooperation on process level by passing of intermediate results between running activities. The involved tasks of a feedback and the produced results are versioned within the process model so that traceability and reestablishing of the work environment is guaranteed. The integration with a version and resource model is currently investigated (cf. [HKNW96]).

B) Change-oriented: In the change-oriented approach, activities, transactions, and changes are closely interrelated and are the basis for the integration of process and version models. An activity is performed in a (mostly human-centered) transaction within the work environment. The transaction leads to some change of a product which is related to the activity. The actual flow of information between the work environments is not represented on the process level. Rather, this approach is based on support for controlled cooperative work between activities which determine the content and topology of the corresponding work environments. Whereas the activities define the context, the underlying SCM system is in charge of providing version and workspace control capabilities and of supporting cooperative work on the technical level. Its strength lies in the good cooperation support and its weakness is the little support for coordination on the process level. Obviously, this approach fits very well to change-oriented version models and hence allows for tightly integrated version and process modeling formalism as well. However, the systems which follow this approach most often support integration only on a technical level so that only a loose integration on a conceptual level is achieved (e.g., *EPOS* [Con+94], *COO* [God+96], *ClearGuide* [Leb97]) or have very limited process management capabilities (e.g., *Asgard* [MiCl96]).

4.2 Product-centered Integration

C) Product-state-centered: The product-state-centered approach to integrate process capabilities into configuration management focuses mainly on controlling the product life-cycle and on a low-level process functionality based on state transitions and event/trigger mechanisms, as illustrated in Figure 3. Usually, there is no (high-level) notion of an activity. Thus, the work environment consists only of the workspace which is managed by the SCM system. Event/trigger mechanisms may well support low-level process functionality. But they are not suitable as a general process modeling formalism, because they are difficult to understand for humans, the execution is hard to control, and high-level concepts for process specifications are missing [BEM94].

Most SCM-systems fall into this class. E.g., *ClearCase* [Leb94] provides a process *control* toolset which supports access control, event recording and dynamic policy enforcement based on triggers. However, ClearCase does not offer any comprehensive process modeling and enacting capabilities.

The problems of the low-level process formalisms may be reduced by a process formalism which hides the low-level event/trigger definitions: *Adele-Tempo* [EsCa94, BEM94] is a SCM-system which provides process functionality based on a trigger mechanism. On top of this, the process formalism Tempo was proposed which is based on the role concept. A process type is defined as a set of objects playing a role. Thus, this formalism is again product-centered. It supports actor assignment to activities but lacks modeling information and control flows of processes. Comprehensive process support is therefore missing.

Another approach of integrating high-level process support for SCM systems and in particular for the Adele system is *APEL* [EDA97]. This approach is mainly
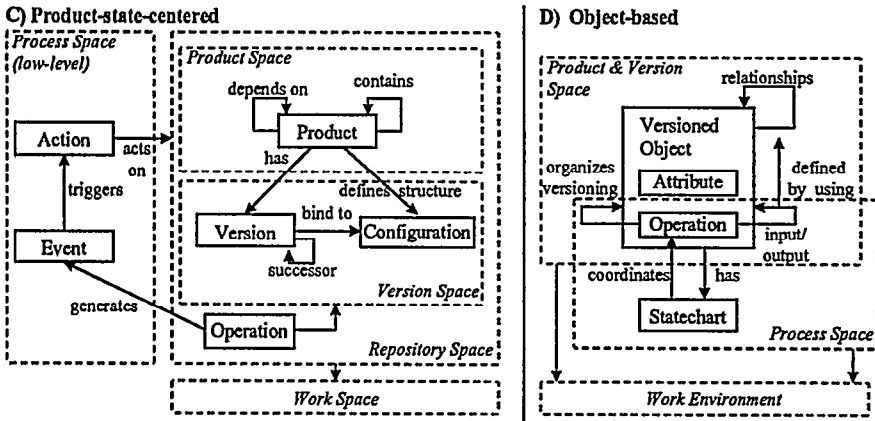
Fig. 3. Product-centered integration approaches

based on activities (following the change-oriented approach), but also integrates
product-state-based aspects. It provides comprehensive process and cooperation sup-
port. Unfortunately, enactment is not provided on this high-level of abstraction but
needs compilation to the Adele language, i.e. to event/trigger rules.

D) Object-based: The object-based approach is adapted from object-oriented mod-
eling techniques which are applied for process and configuration management. In
this approach, software objects are the building blocks of the model. They define the
structure of the product space, organize their own versioning, and specify operations
which may be performed on an object. Further, the execution behavior is specified by
a statechart variant which defines the coordination of the object's operations. Opera-
tions and their coordination specifications form the process space which overlaps the
product and version space. In this approach, product-state- and structure-based as-
pects are integrated best. Moreover, SPM and SCM concepts are integrated on a high
level of abstraction. However, we believe that an approach that is exclusively based
on objects has its limitations for a general PML. The intrinsic subordination of ac-
tivities to software objects or documents and the implicit representation of informa-
tion flows are strong weaknesses. But, an enhancement of such an approach which
defines and integrates different object classes for processes, artifacts and their ver-
sions etc. whose behavior are coordinated by message/event passing seems to be a
promising approach (cf. [Joe97]).

    ESCAPE+ [NSS96] and *SOCCA* [EnGr94] follow this approach. SOCCA does
not explicitly take versioning into account, whereas ESCAPE+ extends its predeces-
sor ESCAPE [Jun95] of version concepts.

4.3 Comparison

From the process management point of view, most PMLs disregard versions and configurations. They are designed on top of SCM systems and hence are integrated only on a technical level. From the configuration management point of view, support for process control and automation may be found in SCM systems for a few years based on "low-level" mechanisms like rules and triggers. Integration of more high-level process modeling and enacting concepts is currently investigated in the SCM community. The capabilities of the most promising approaches are summarized in Table 1. Note, that these approaches are not dataflow-oriented and provide only poor support for feedback handling on the process level.

Requirement	EPOS	Adele/Tempo	APEL	ESCAPE
Integration Aspects				
structural integration	by planning	no	no	inherent
input/output relationship	alternate task and data net	implicit,	in/out parameter + dataflow dependencies	derived from object relation
behavioral integration	based on attributes	ECA-rules on attributes	local STDs	inherent (statecharts)
scheduling & concurrency control	not integrated	implicit	(yes)	implicit
SCM operations	(no)	yes	(no)	yes
general SCM processes	yes	(no)	yes	(no)
joining of correlated processes	no	no	no	no
Int. WE support				
controlled cooperation	cooperating transactions	role collaboration	various cooperation policies	cooperation patterns
free cooperation	WE overlaps	public areas	public areas	no

Table 1: Integrated SPM and SCM support provided by different systems

5. Conclusion

Integration of process and configuration management support is of growing interest. We have addressed this topic by focusing changes and change processes. A fundamental prerequisite to provide such a comprehensive support in a PSEE is the integration of the underlying representation formalism of processes and versions to obtain explicit descriptions of the process and repository space. Due to the diversity of the existing modeling languages for processes and versions, integration of these formalisms is a great challenge. To cope with this problem, a common conceptual basis of process and version modeling concepts and elements is needed. In this paper, we have proposed a first approach to such a conceptual framework which adopts and integrates generally used notions of SPT and SCM, and is based on the notions of

process space, product space, version space and work environment. We have tried to outline the interdependencies between process and version models on a high-level of abstraction and to identify the basic requirements for an integrated approach.

An important issue discussed in this paper is the classification of how to integrate process and version models. Based on the fundamental distinction of activity-centered and product-centered integration, four classes have been identified, namely the dataflow-oriented, the change-oriented, the product-state-based, and the object-based approach. Every approach has its specific strengths and weaknesses.

We hope that the material presented in this paper will be fruitful to support future work on this challenge. Still much work has to be done in this direction. First, it is an open issue whether the development of a comprehensive PML leads to one large language or to several separated but well integrated sub-PMLs (cf. [CoLi95]). Second, the interdependencies between process and version models must be discussed in more detail. E.g., the consequences of specific representation paradigms like state-oriented vs. change-oriented version models on the integration with process concepts have to be considered. Furthermore, we have to focus on the impact of different concepts for workspace management on an integrated approach. Third, SCM functionality for building derived version and constructing configurations which may be provided by version rules have to be considered in a more comprehensive version of the framework. Finally, the framework has to be extended to explicitly describe the interrelationships between process and version models on the different level of abstraction (type, template, instance level).

Acknowledgments

References

[BaFu93]   Bandinelli, S.; Fugetta, A.: "Computational Reflection in Software Process Modeling: the SLANG Approach", in *Proc. of the 15th Int. Conf. on Software Engineering*, IEEE Computer Society Press, 1993; pages 144-154.

[BEM94]   Belkhatir, N.; Estublier, J.; Melo, W.L.: "ADELE-TEMPO: An Environment to Support Process Modelling and Enaction", in [FKN94], pages 187-222.

[BFGL94]   Bandinelli, S.; Fuggetta, A.; Ghezzi, C.; Lavazza, L.: "SPADE: An Environment for Software Process Analysis, Design, and Enactment", in [FKN94]; pages 223-248.

[Cag95]   Cagan, M.: "Untangling Configuration Management", in [Est95]; pages 35-52.

[CFFS92]   Conradi, R.; Fernström, C.; Fuggetta, A.; Snowdon, R.: "Towards a Reference Framework for Process Concepts", in *Software Process Technology - Second European Workshop EWSPT'92*, LNCS 635, Springer, Berlin, 1992; pages 3-17.

[CFF93]   Conradi, R.; Fernström, C.; Fugetta, A.: "A Conceptual Framework for Evolving Software Processes", in *ACM SIGSOFT Software Engineering Notes*, 18(4), Oct. 1993; pages 26-34.

[CoLi95] Conradi, R; Liu, Ch.: "Process Modelling Languages: One or Many?", in *Software Process Technology - Fourth European Workshop EWSPT'95*, LNCS 913, Springer, Berlin, 1995; pages 98-118.

[Con+94] Conradi, R. et al.: "EPOS: Object-Oriented Cooperative Process Modelling" In [FKN94]; pages 33-70.

[CoWe97] Conradi, R.; Westfechtel, B.: "Version Models for Software Configuration Management", to appear in *Computing Surveys*, 1997

[DoFe94] Dowson, M.; Fernström, Ch.: "Towards Requirements for Enactment Mechanisms", in *Software Process Technology - Third European Workshop EWSPT'94*, LNCS 772, Springer, Berlin, 1994; pages 90-106.

[Dow87]   Dowson, M.: "Iteration in the Software Process", in *Proc. of the 9th Int. Conf. on Software Engineering*, IEEE Computer Society Press, 1987; pages 36-39.

[EnGr94] Engels, G.; Groenewegen, L.: "SOCCA: Specification of Coordinated and Cooperative Activities", in [FKN94], pages 71-102.

[EDA97]   Estublier, J.; Dami, S.; Amiour, M.: "High Level Process Modeling for SCM Systems", in *Software Configuration Management - ICSE'97 SCM-7 Workshop*, LNCS 1235, Springer, Berlin, 1997; pages 81-97.

[EsCa94] Estublier, J.; Casallas, R.: "The Adele Configuration Manager", in [Tic94], pages 99-130.

[Est95]   Estublisher, J. (ed.): "Software Configuration Management - ICSE SCM-4 and SCM-5 Workshops, Selected Papers". LNCS 1005, Springer, Berlin, 1995.

[FeHu93] Feiler, P.H.; Humphrey, W.S.: "Software Process Development and Enactment: Concepts and Definitions", in *Proc. of the 2nd Int. Conf. on the Software Process*, IEEE Computer Society Press, 1993; pages 28-40.

[Fei91]   Feiler, P.H.: "Configuration Management Models in Commercial environments". Technical Report CMU/SEI-91-TR-7, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 1991.

[FKN94]   Finkelstein, A.; Kramer, J.; Nuseibeh, B.: "Software Process Modelling and Technology". Research Studies Press, Chichester, 1994.

[GCCM95] Godart, C.; Canals, G.; Charoy, F.; Molli, P.: "About some relationships between configuration management, software process and cooperative work: the *COO* Environment", in [Est95]; pages 173-178.

[God+96] Godart, C.; Canals, G.; Charoy, F.; Molli, P.; Skaf, H.: "Designing and Implementing *COO*: Design Process, Architectural Style, Lessons Learned", in *Proc. of the 18th Int. Conf. on Software Engineering*, IEEE Computer Society Press, 1996, pages 342-352.

[GHS95]   Georgakopoulos, D.; Hornick, M.; Shet, A.: "An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure". *Distributed and Parallel Databases*, 3(2), April 1995; pages 119-153.

[HJKW96]   Heimann, P.; Joeris, G.; Krapp, C.-A.; Westfechtel, B.: "DYNAMITE: Dynamic Task Nets for Software Process Management", in *Proc. of the 18th Int. Conf. on Software Engineering,* IEEE Computer Society Press, 1996, pages 331-341.

[HKNW96]   Heimann, P.; Krapp, C.-A.; Nagl, M.; Westfechtel, B.: "An Adaptable and Reactive Project Management Environment", in Nagl, M. (ed.) *Building Tightly Integrated Software Development Environments: The IPSEN Approach*, LNCS 1170, Springer, Berlin, 1996; pages 504-534.

[Hum89]   Humphrey, W.: "Managing the Software Process". Addison-Wesley, Readings, Massachusetts, 1989.

[Joe97]   Joeris, G.: "Cooperative and Integrated Workflow and Document Management for Engineering Applications", to appear *in Proc. of the DEXA'97 Workshop on Scientific Workflows*, IEEE Computer Society Press, 1997.

[Jun95]   Junkermann, G.: "A Grapical Language for Specification of Software Processes", Dissertation at the University Dortmund, Germany, 1995 (in german).

[Leb94]   Leblang, D.B.: "The CMChallenge: Configuration Management that Works", in [Tic94], pages 1-37.

[Leb97]   Leblang, D.B.: "Managing the Software Development Process with ClearGuide", in *Software Configuration Management - ICSE'97 SCM-7 Workshop*, LNCS 1235, Springer, Berlin, 1997; pages 67-80

[Lon93]   Lonchamp, J: "A Structured Conceptual and Terminological Framework for Software Process Engineering", in *Proc. of the 2nd Int. Conf. on the Software Process*, IEEE Computer Society Press, 1993; pages 41-53.

[MiCl96]   Micallef, J.; Clemm, G.: "The Asgard System: Activity-Based Configuration Management", in *Software Configuration Management - ICSE'96 SCM-6 Workshop*, LNCS 1167, Springer, Berlin, 1996; pages 175-186.

[NSS96]   Neumann, O.; Sachweh, S.; Schäfer, W.: "A High-Level Object-Oriented Specification Language for Configuration Management and Tool Integration", in *Software Process Technology - Fifth European Workshop EWSPT'96*, LNCS 1149, Springer, Berlin, 1996; pages 137-143.

[Tic94]   Tichy, W. (ed.): "Configuration Management". John Wiley & Sons, New York, 1994.