A Filter–Mechanism for Method–Driven Trace Capture[†]

Ralf Dömges, Klaus Pohl, Klaus Schreck
RWTH Aachen, Lehrstuhl Informatik V, 52056 Aachen, Germany
email: {doemges|pohl|schreck}@i5.informatik.rwth–aachen.de

Abstract: Traceability is a prerequisite for developing high quality (software) systems. Recording and maintaining all available information is too labor intensive and thus by far too expensive. A project-specific definition of the trace information to be recorded and the method fragments (so called *trace fragments*) to be executed for recording the information provides a solution for this problem. But the amount of traces to be recorded does not only vary from project to project. It also varies between project phases and even within a project phase. As a consequence project-specific trace fragments need to be adapted according to the actual project phase.

In this paper we propose a model-based filter mechanism to significantly reduce the required effort to adapt trace fragments. By defining appropriate filters the project manager is able to (dynamically) adapt the project-specific trace fragments to the actual needs. We present an example to highlight the benefits of the approach and discuss possible extensions.

[†] CREWS Report Series 98–10. A version of this paper appears in the *Proceedings of the 10th International Conference on Advanced Information System Engineering*, Pisa, Italy, June 8–12, 1998.

1 Introduction

1.1 Traceability: Definition and Motivation

Traceability, more precisely requirements pre– and post–traceability, is recognized by many research contributions and experience reports as an important prerequisite for developing high quality systems (e.g., [Collery, 1988;IEE, 1991;Ramesh *et al.*, 1996]). Among others, traceability facilitates the development, validation and maintenance of a system, provides invaluable support for the integration of changes, reduces the amount of errors during system development, allows experience–based process improvement, and is important to prove the fulfillment of contracts [Jarke *et al.*, 1994;Pohl, 1996b]. Traceability is thus required by various system development standards (e.g., V–Modell [Bröhl and Dröschel, 1993], DoD–2167A [DoD-2167A, 1988]) as well as quality and process improvement frameworks, like ISO 9000–3 [ISO, 1991] and CMM [Paulk *et al.*, 1993].

Various contributions (e.g., [Kaindl, 1993;Gotel, 1996;Yu and Mylopoulos, 1994;Conklin and Begeman, 1988]) and traceability frameworks (e.g., [Pohl, 1996b;Ramesh *et al.*, 1996]) define a large set of information to be captured for enabling traceable system development. Obviously, an approach recording all this trace information in each project and for all system components is by far too time consuming, thus too expensive, and therefore likely to fail [Tilbury, 1989].

1.2 Project–specific Trace Capture: Method–Driven Approach

To reduce the effort for capturing and maintaining traceability information the *traces*, i.e., the information which is actually being stored, has to be recorded according to *project–specific needs* [Pohl and Dömges, 1997].

Existing prototypical trace environments (e.g., TOOR [Pinheiro and Goguen, 1996], PRO–ART 1.0 [Pohl, 1996b]) as well as existing commercial requirements and system engineering environments like DOORS [Quality Systems & Software, 1996], RDD–100 [Ascent Logic Corporation, 1994], RTM [Marconi Systems Technology, 1996], or SLATE [TD Technologies, Inc., 1996] support the persistent recording and management of trace information. They typically provide a set of generic information types and operations which can be specialized according to project–specific needs. Thus, they empower the project manager to define project–specific trace information, e.g., design decisions and their relations. The environments provide comprehensive consistency checking capabilities and support the retrieval and display of the recorded traces by advanced ad–hoc and pre–defined querying, browsing, and reporting mechanisms. But they do not provide systematic support *for recording the defined trace information* during the system development process, e.g., the project manager cannot define when (in which situation) a decision should be recorded.

Our method–driven approach to trace capture (see [Pohl *et al.*, 1997] for details) overcomes this shortcoming. It supports the *explicit definition* of project–specific *trace information* together with *trace fragments* which define strategies for capturing the information. Moreover the user is *guided* in recording the information according to the project–specific trace fragment definitions. This is technically achieved by interpreting the defined trace capture strategies in a process–integrated environment. Based on the interpretation the environment reminds the user about the traces to be captured, enforces (e.g., in project critical procedures), and even automates (whenever possible)

information type	trace fragment	recording mainly	
		interactive	automated
product information (e.g., ER–model)	process fragment	X	
supplementary product information (e.g., design decisions)	supplementary process fragment	X	
process observation information (e.g., method fragment)	process observation fragment		X
dependency information (e.g., interrelate ER-model and design decisions)	dependency fragment	X	X

Tab. 1. Trace information and corresponding trace fragments

the recording of traces[2].

We use the NATURE process meta model [Rolland and Grosz, 1994;Pohl, 1996b] to define the project specific trace fragments. The model distinguishes between:

- *Atomic fragments* for representing the part of a method definition which can be automated. Atomic fragments have no (externally) visible structure and are "hard–coded" in the tool or environment. By executing atomic fragments traces are (automatically) created and recorded in the *trace–repository*.
- *Strategy selection fragments* for representing the part of the method definition in which the user has to make a decision between at least two *alternatives*, i.e., alternative trace capture strategies. A trace strategy defines the way how traces are (interactively) created.
- *Composed method fragments* (resp., *composed trace fragments*) for defining complex trace strategies, i.e., for defining a certain order on a set of trace fragments.

The project manager uses the three types of fragments to define the project–specific trace strategies. Thereby s/he (implicitly) defines the trace information to be recorded. To support the definition of the trace strategies we distinguish between four information types. Depending on the information type the recording is performed manually and/or automatically. Table 1 depicts the information types and the way an information is typically being recorded.

We have integrated the project–specific trace capture strategy into our TECHMOD [Dömges *et al.*, 1996] and PRO–ART 2.0 [Pohl, 1996a] environments. We used both environments in small case studies. The studies showed that reminding the stakeholders about the recording of the project specific trace information resulted in traces of higher quality compared to traces produced by following "paper–based" trace capture guidelines. Moreover, recording of unnecessary trace information was avoided and the work load of the users was (significantly) reduced.

1.3 Project–specific Adaptation of Trace Fragments

However, it turned out that the trace fragments had to be adapted due to two main reasons:

1. Traces *vary between project phases*. For example, while in the specification phase recording of structured design decisions was demanded, the maintenance phase focused on recording and interrelating change requests and change approval forms.

[2] The detailed mechanism required to enable such a project–specific guidance of trace capture based on the interpretation of method fragments is described in [Pohl and Weidenhaupt, 1997].

2. Traces *depend on the components* developed, e.g., when safety critical and/or very complex components are developed design decisions, meeting notes, conceptual drawings, as well as scenarios must be captured and interrelated.

An obvious solution for the adaptation of trace fragments to project–phase–specific needs was to *specialize* the (existing) trace fragments accordingly. This solution had two significant shortcomings

1. *modeling and/or re–modeling* of the required trace fragments was a difficult and time consuming task;

2. *programming and/or re–programming* of atomic fragments was often required.

As a consequence the amount of trace fragments maintained in our method base *rapidly increased*. The trace fragments were *almost identical*; they differed only slightly in the trace–capture dependent parts. Thus *managing* the trace fragments got very complicated and *maintaining* them consistently was almost impossible.

1.4 Approach and Structure of Paper

In this paper we propose a *filter mechanism* to overcome the above shortcomings. In section 2 we elaborate the main requirements for a filter mechanism to significantly reduce the required effort to adapt trace fragments. Providing a filter mechanism is essential to empower an easy and flexible adaptation of fragments to continuously evolving information needs. The filter mechanism described in sections 3 and 4 allows a *dynamical adaptation* of trace fragments. Filters can be defined for a specific project phase as well as for particular trace fragments. A prototypical implementation of the filter mechanism was integrated into our process integrated environments and applied to small examples (section 5). Finally, we provide a conclusion of the achieved results and give an outlook on future work (section 6).

2 Requirements for a Model-based Filter Mechanism

2.1 Prevent Storage of Trace Information

The filter mechanism has to ensure that certain trace information is not recorded in the trace repository. To avoid the re–programming of atomic fragments the filter mechanism should be able to partially block the output of an atomic fragment from being stored in the trace repository.

For example, take the automated recording of process observation information by an atomic fragment. This fragment records all executed trace fragments together with the agent who executed them. Assume that due to contractual or organizational regulations the agents should not be recorded. A trace filter which could block the information about the agent of being stored in the trace repository avoids the re–programming of the atomic fragment.

2.2 Restrict Selection of Trace Strategies

The filter mechanism must provide means to restrict the alternatives provided by a strategy selection fragment. Whenever an alternative of a strategy selection fragment should not be used, it should not be offered to the user.

For example, a strategy selection fragment provides four alternatives to the user to justify the creation of a new product version: (1) recording a structured design decision, (2) stating the reasons for the change as informal text, (3) selecting an appropriate part

of the project–contract, or (4) indicating the person creating the new version. If during the early project–phases justifications should only be given by informal text or by naming the responsible person the two other alternatives must be removed.

2.3 Prevent Execution of Trace Fragments

The filter mechanism must allow to *prevent the execution* of a trace fragment. Whenever the entire output information of a trace fragment should not be recorded the execution of the fragment has to be prevented; especially if a trace fragments requires user interactions.

For example, if design decisions should not be recorded during early project phases the execution of the trace fragment for recording the decision has to be prevented.

2.4 Enable Filter Definitions for the Overall Projects and Project Phases

In addition to filter definitions for a single trace fragment the filter mechanism should provide means to define filters which affect all trace fragments executed in a project phase. For example, to capture no process observation information during the proposal phase whereas their recording is mandatory during the requirements engineering and design phases it should be possible to define a filter for an entire project phase.

2.5 Empower Nested Filter Definitions

Trace fragments can be nested forming composed trace or strategy selection fragments. This requires propagation rules for filters defined for nested trace fragments. For example, assume that capturing process observation information is explicitly prohibited for a composed trace fragment. Consequently, none of the trace fragments used for defining the composed trace fragment should record process observation information, even if defined differently by filters of these trace fragments.

2.6 Enable Enforcement of Information Storage and Fragment Execution

Within a nested trace fragment various information types can be blocked, alternative strategies can be restricted, and/or the execution of trace fragments can be prevented by defining the appropriate filters. For a composed trace or strategy selection fragment the filter mechanism must provide means to ensure the recording of a certain type of information, the offering of particular alternative trace strategies, and/or the execution of specific trace fragments regardless of the filters defined for the fragments which are (constituent) parts of the composed fragments.

For example, assume that the recording of automated dependency information is blocked by a filter *F* defined for a trace fragment *tf*. To assure the recording of this information in a composed trace fragment the project manager should be able to define a filter *F'* which "replaces" *F*; i.e., which assures the recording although the nested filter *F* prohibits the recording.

3 Information, Strategy, and Method Filters

To fulfill the requirements defined above three filter types are required: *information filters* which prevent already produced information from being stored in the repository (section 3.1), *strategy filters* which restrict the available trace capture strategies defined by a strategy selection fragment (section 3.2), and *method filters* which prevent trace fragments from being executed (section 3.3).

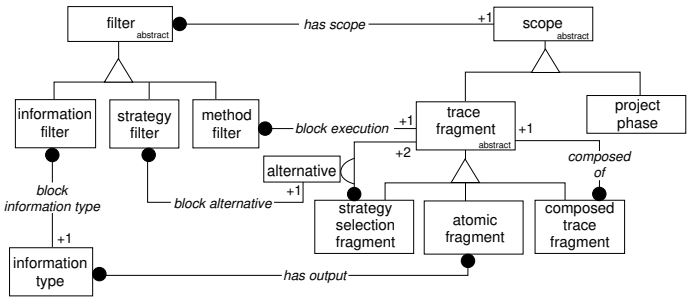

Fig. 1: Filters types

These three filter types are defined as specialization of the concept *filter* (see figure 1). The *scope* defines if a filter is valid within a *project phase* or a *trace fragment* (see requirement 2.4). A filter can be related to more than one scope. If a filter should be applied in particular project phase(s), the filter is associated to the phase(s) by using the *has scope* association. If a filter is associated to all project phases, the filter is applied to the overall project. If a filter should only be valid for a particular trace fragment, the filter is related to the fragment using the *has scope* association.

3.1 Information Filters

Information filters prevent information types from being stored persistently although the corresponding trace fragments are executed and their output information is produced (figure 2; requirement 2.1). The type of information to be blocked by a given information filter is defined using the association *block information type* defined between the class *information type* and the class *information filter*. The project phases and trace fragments to which an information filter should be applied are related to the information filter using the *has scope* association. An information filter can be used to avoid the recording of a particular information produced by an atomic fragment; but it can also be used to avoid the recording of a particular information during a project phase. An information filter can block more than one information type.

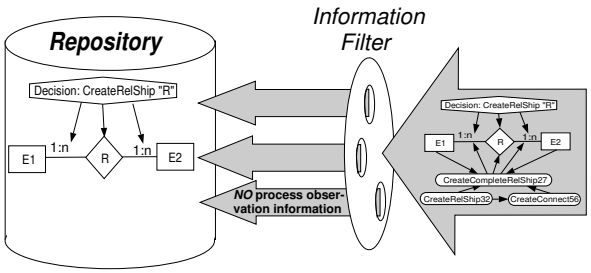

Fig. 2: Information filters

Information filters are essential for influencing the information capture of atomic fragments. Information filters are very well suited for blocking process observation information which is typically recorded automatically. They are generally applicable for dependency information, especially if dependencies are created automatically. Information filters should not be used to filter supplementary product information, since these information types are usually created interactively.

3.2 Strategy Filters

Strategy filters restrict the available alternatives of strategy selection fragments (figure 3; see requirement 2.2). By applying strategy filters to a strategy selection fragment only a subset of the defined alternatives is offered to the user. The alternative(s) which should not be offered are defined using the association *block alternative* defined between the class *strategy filter* and the association class *alternative* defined between the class *strategy selection fragment* and the class *trace fragment* (figure 1). A strategy filter can be related to more than one alternative fragment.

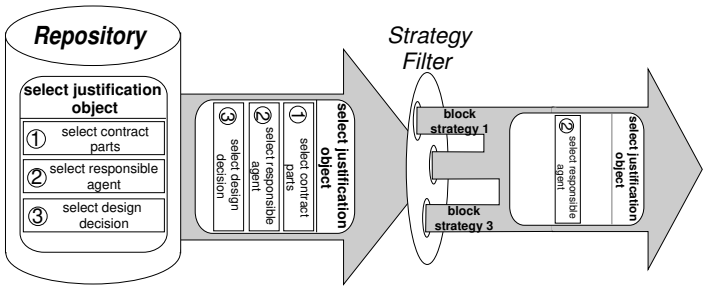

Fig. 3: Strategy filters

The scope of the filter is defined by associating the appropriate project phases and/or trace fragments with the filter. If a strategy filter is associated to one or more project phases, the blocked alternatives are not offered within the defined phases; but are offered within other phases. If it is associated to a trace fragment, the blocked alternatives are not offered whenever the strategy selection fragment is executed within the trace fragment.

3.3 Method Filters

Method filters prevent the execution of one (or more) atomic fragment, strategy selection fragment, or composed trace fragment (figure 4; see requirement 2.3). The trace fragments to be blocked are defined using the association *block fragment* defined between the class *method filter* and the class *trace fragment* (figure 1). A method filter can block more than one method fragment.

The scope of the filter is defined by using the *has scope* association to relate project phases and/or trace fragments with the filter. If a method filter is associated to one or more project phases, the blocked fragment is never executed within the defined phases; but it will be executed in other phases. If it is associated to a trace fragment, the blocked fragment is not executed within the associated fragment.

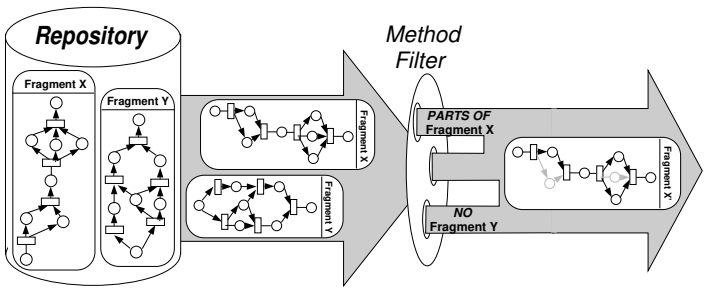

Fig. 4: Method filters

Method filters should be used to adapt the interactive capture of traces, i.e., the recording of supplementary products and *interactively created* dependency information. They provide no means to change the internal control-flow of composed trace fragments. This can (only) be achieved by a manual adaptation of the fragment definition.

4 Nesting Information, Strategy, and Method Filters

The trace fragments which define project-specific trace capture can be nested by defining composed trace or strategy selection fragments. Each of these nested trace fragments *tf* can be defined as the scope for a set of filters (denoted as *filterset*(*tf*)). As a consequence, the filters defined for the trace fragments are also nested. Consequently, we need to define propagation rules to determine the filters which have to be applied if a particular trace fragment is executed (section 4.1).

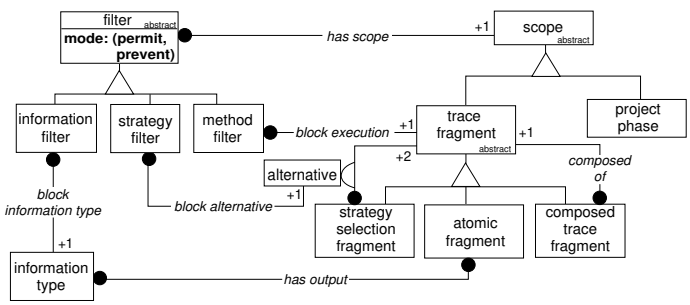

Fig. 5: Filter modes

The (propagated) filter can not be used to enforce the recording of information types, that alternatives of a strategy selection fragment are offered, or the execution of particular trace fragments. For example, assume a filter associated to a trace fragment which is contained within a composed trace fragment. The filter defines to block the recording of a particular information type. If we want to ensure that the trace information will be recorded whenever the nested fragment is executed, we must provide means to define which information should be recorded during the execution of a composed trace fragment regardless of the filter definitions associated to its contained fragments. We therefore introduce a *filter mode* attribute (figure 5). Using this attribute the project manager is able to define explicitly that a filter *prevent*s or *permit*s (a) the defined information type to be recorded (information filter); (b) a certain alternative to be offered (strategy filter); (c) a trace fragment to be executed (method filter).

Due to the filter mode the filters which have to be applied to a trace fragment may contradict each other. In section 4.2 we define these contradictions and describe how they can be resolved. Finally, we provide some rules towards a systematic application of trace filters (section 4.3).

4.1 Fragment Scopes and Valid Filter Sets

For nested filters we determine the *valid filter set* of a trace fragment. The valid filter set consists of the filters to be applied to a fragment whenever the fragment is executed. We first define a *containment–relationship* between trace fragments:

- A trace fragment *tf directly contains* another trace fragment *tf'* (denoted as *tf'* $\in$ *tf*) iff
  a.) *tf* is a composed method fragment and there exists a *composed of* link between *tf* and *tf'*.
  b.) *tf* is a strategy selection fragment and there exists an *alternative* link between *tf* and *tf'*.

  In other words, *tf'* is used to define *tf*.
- A trace fragment *tf indirectly contains* another trace fragment *tf'* (denoted as *tf'* $\overset{*}{\in}$ *tf*) iff *tf'* $\in$ *tf* or $\exists$*tf''* with *tf''* $\overset{*}{\in}$ *tf* $\wedge$ *tf'* $\in$ *tf''*

Figure 6 depicts examples of this containment relationship for some trace fragments, e.g., $tf_2 \in tf_8$, $tf_1 \in tf_2$, and $tf_1 \overset{*}{\in} tf_8$.

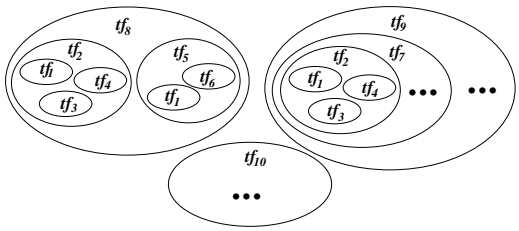

Fig. 6: Containment and scopes of trace fragments

To determine a valid filter set we need to define the *scope* of a nested trace fragment *tf*. The scope of *tf* is a set of trace fragments which directly or indirectly contain each other. The scope of *tf* is defined *relative* to a particular trace fragment *tf'* because *tf* may be (directly or indirectly) contained in more than one trace fragment. For example, a trace fragment to record design decisions can be used within a composed trace fragment which defines how to integrate changes into an existing specification as well as within a trace fragment which describes the creation of a new version of a specification. In figure 6 are $tf_1 \in tf_2$ and $tf_1 \in tf_5$. Consequently, a trace fragment *tf* usually has more than one scope.

We define

$$scope(tf, tf') = (tf_{k_0}, \ldots, tf_{k_n})$$

with $tf = tf_{k_0}$, $tf_{k_{i-1}} \in tf_{k_i}, \forall i \in \{1, \ldots, n\}$, and $tf_{k_n} = tf'$. If $tf = tf_{k_0} = tf_{k_n} = tf'$ then $scope(tf, tf') = (tf) = (tf_{k_0}) = (tf_{k_n}) = (tf')$. Figure 6 depicts some examples: *scope*(tf_1,tf_8) = (tf_1,tf_2,tf_8) and *scope*(tf_1,tf_5) = (tf_1,tf_5)

Based on *filterset*(*tf*) we define the *valid filter set* of a nested trace fragment *tf* (denoted as *VFS*(*tf*,*tf'*)) as the union of the filtersets of its corresponding scope:

VFS(*tf*,*tf'*) = *filterset*(*scope*(*tf*,*tf'*))

and $\textit{filterset}((tf_{k_0}, \ldots, tf_{k_n})) = \bigcup_{i=0,\ldots,n} \textit{filterset}(tf_{k_i})$.

VFS(*tf*,*tf'*) contains also the filters which are associated with the project phase in which *tf* (and the fragments of *scope*(*tf*,*tf'*)) is actually executed.

By defining and using the scope of a trace fragment to define its valid filter set we have established the propagation rules to determine the filters which have to be applied when a particular trace fragment is executed.

4.2 Contradicting Information, Strategy, and Method Filter Definitions

A valid filter set may contain filters having contradicting filter modes. For example, if the recording of design decisions is enforced on the level of a composed trace

fragment (i.e., the filter mode is set to *permit*) but prohibited on the level of trace fragments which are contained in the composed trace fragment it is not clear which filter should be applied.

We thus need to define those contradictions and how to resolve them. In the following we denote *filterobjects*(*F*) of an information, strategy, or method filter *F* as the set of all information types, alternatives, or trace fragments to be permitted or prevented by *F*.

Contradictions between two filters of the same type are defined and resolved as follows:

- Let *IF* and *IF'* be two information filters, *filterobjects*(*IF*) = {it_1,...,it_n}, $n \geq 1$, and *filterobjects*(*IF'*) = {it'_1,...,it'_m}, $m \geq 1$. The filter definitions of *IF* and *IF'* are contradictory iff the filter mode of *IF* is *permit*, the filter mode of *IF'* is *prevent* (or vice versa), and $\{it_1,...,it_n\} \cap \{it'_1,...,it'_m\} = \{it_{k_1}, \ldots, it_{k_l}\}$, $l \geq 1$.
  If *IF, IF'* $\in$ *VFS*(*tf,tf'*) are two contradictory information filters, *scope*(*tf, tf'*) = {tf_1,...,tf_m}, *IF'* $\in$ *filterset*(tf_i), *IF* $\in$ *filterset*(tf_j) and $i > j$, the filter definitions of *IF'* for $\{it_{k_1}, \ldots, it_{k_l}\}$ replace the filter definitions of *IF* for $\{it_{k_1}, \ldots, it_{k_l}\}$ by removing them from *VFS*(*tf,tf'*) (figure 7 a.)).

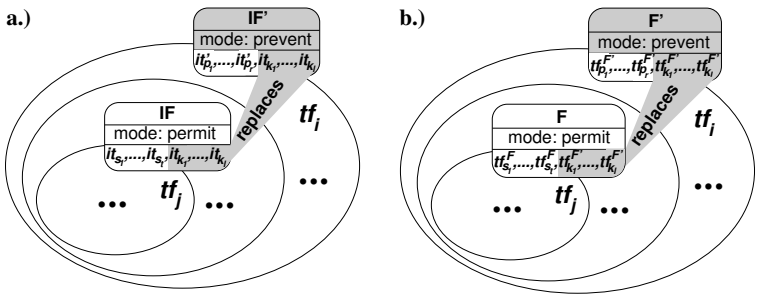

Fig. 7 Contradictory filter definitions

- Let *F* and *F'* be two strategy filters (or two method filters), *filterobjects*(*F*) = {$tf^F{}_1$,...,$tf^F{}_n$}, $n \geq 1$, and *filterobjects*(*F'*) = {$tf^{F'}{}_1$,...,$tf^{F'}{}_m$}, $m \geq 1$. The filter definitions of *F* and *F'* are contradictory iff the filter mode of *F* is *permit*, the filter mode of *F'* is *prevent* (or vice versa), and $\{tf^F{}_1,...,tf^F{}_n\} \cap \{tf^{F'}{}_1,...,tf^{F'}{}_m\} = \{tf^F_{k_1}, \ldots, tf^F_{k_l}\}$, $l \geq 1$.
  If *F, F'* $\in$ *VFS*(*tf,tf'*) are two contradictory strategy filters (or method filters), *scope*(*tf,tf'*) = {tf_1,...,tf_p}, *F'* $\in$ *filterset*(tf_i), *F* $\in$ *filterset*(tf_j) and $i > j$, the filter definitions of *F'* for $\{tf^F_{k_1}, \ldots, tf^F_{k_l}\}$ replace the filter definitions of *F* for $\{tf^F_{k_1}, \ldots, tf^F_{k_l}\}$ by removing them from *VFS*(*tf,tf'*) (figure 7 b.)).

If $tf_i = tf_j$ (i.e., $i = j$) the same fragment is associated with contradicting filter definitions. In this case the project manager has to decide which of the filter definitions should be replaced.

Two filters of different types are contradictory iff *IF* is an information filter, *F* is a strategy filter (or method filter), *filterobjects*(*IF*) = {it_1,...,it_n}, *filterobjects*(*F*) = {tf'_1,...,$tf'_{m'}$}, the filter mode of *IF* is *permit*, the filter mode of *F* is *prevent*, and there exists an atomic fragment $af \overset{*}{\in} tf'_l$, $l \in \{1,...,m'\}$ which produces $\{it_{j_1}, \ldots, it_{j_k}\} \subset \{it_1, \ldots, it_n\}$.

If *IF*, $F \in$ *VFS*(*tf,tf'*) are two contradicting filters, where *IF* is an information filter and *F* is a strategy filter (or method filter), *scope*(*tf,tf'*) = {tf_1,...,tf_m}, *IF* $\in$ *filterset*(tf_i),

$F \in filterset(tf_j)$ and $i \geq j$ the contradiction can not be resolved automatically but the project manager has to decide how to resolve the contradiction. Thereto s/he needs to determine the trace fragment tf'_l which contains the affected atomic fragment $af \overset{*}{\in} tf'_l$ and has to figure out how to adapt the filter definitions of *IF* and *F*, e.g., by preventing the execution of all trace fragments which are contained within tf'_l except *af*.

There will be no contradictions between the filter definitions of a method filter *MF* and a strategy filter *SF*. We demand that alternatives of strategy selection fragments can only be prevented by strategy filters (see section 4.3) and strategy filters are only able to restrict the alternatives of a strategy selection fragment. Even if a method filter defines to prevent the execution of an alternative it is still offered to the user.

If the filter definitions associated with the project phase in which a trace fragment *tf* (and the fragments of *scope*(*tf*,*tf*')) is actually executed contradict with *VFS*(*tf*,*tf*') the filter modes of the project phase generally replace the filter modes of the trace fragments.

The above definitions can be used to analyze the filters defined for the trace fragments. Contradicting filters can thus be detected before the trace fragments are actually applied. The project manager can resolve the contradictions before the fragments are executed during a project.

4.3 Rules for applying Filters

Based on our experience we provide some rules for applying information, strategy, and method filters:

Apply filters not to product information: Product information should never be affected by filters. Product information is the main output of the development process. Hence, it makes no sense to block their recording. For example blocking product information during the development of a Entity–Relationship model would lead to an incomplete and inconsistent model. Filters should only affect the recording of supplementary product, process observation, and dependency information. If a change in product information is required (e.g., define inheritance (links) in Entity–Relationship–diagrams) new fragments have to be introduced and/or existing fragments have to be adapted. This definition and/or re–definition of a method is not within the scope of a filter mechanism.

Apply information filters only to automated trace fragments: If the information of interactive trace fragments is blocked by information filters it is very likely that users reject to enter the information next time. This might lead to the rejection of the entire filter–based approach for capturing traces. Information filters should thus never be used to block interactively entered information.

Apply method or strategy filters when complete output information is blocked: A fragment whose complete output is blocked by (nested) information filters should not be executed. Instead, a method filter should be defined to prevent the execution, or if the fragment is an alternative of a strategy selection fragment, an appropriate strategy filter should be defined.

Apply method filters when all alternatives of a strategy selection fragment are prevented: If the entire set of alternatives of a strategy selection fragment is prevented by (nested) strategy filters, the fragment should not be executed. Instead of defining strategy filters which block all alternatives, a method filter should be defined to prevent the execution of the strategy selection fragment.

Check effects on composed trace fragments: If any kind of filters prevent the storage of information or the execution of a trace fragment within a composed trace fragment,

the project manager must check if the blocking of the information (or the fragment) does not lead to a "deadlock". In other words, s/he must assure that a composed trace fragment could be executed although a trace fragment is blocked and/or information is not recorded. In the case of a detected deadlock s/he must change the control flow of the composed trace fragment.

Do not apply method filters to block alternatives of strategy selection fragments: Method filters should not be misused as strategy filters, i.e., they should not be used to block an alternative of a strategy selection fragment. By defining a strategy filter, the alternative is not offered to the user, whereas in the case of a method filter, the alternative is offered to the user. The user can choose the alternative, but the chosen alternative will not be executed.

Together with the scope and contradictions defined in sections 4.1 and 4.2 the rules provide the basis for developing an environment which supports the project manager in defining consistent trace filters of any type.

5 Model–based Filtering: An Example

We illustrate our model-based filter mechanism using a small example.

The composed trace fragment *integrate change request* guides the application engineer during the integration of changes (figure 8). The application engineer is first reminded to justify the changes. The strategy selection fragment *select justification object* defines three alternative strategies for the justification: (1) to select appropriate parts of a contract; (2) select the stakeholder who initiated the change; or (3) to select a specific design decision. A process observation fragment automatically records the execution of the strategy selection fragment and the chosen alternative. During the integration of changes an automated dependency step relates the object representing the justification with the modified and/or created specification parts.

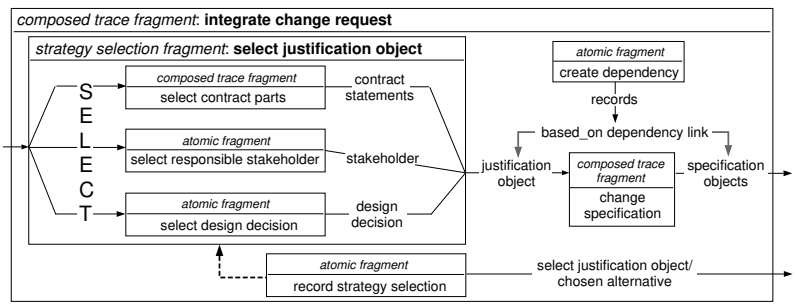

Fig. 8: Composed trace fragment *integrate change requests* (simplified).

The fragment described above is reused for the proposal phase of another project. The project management decides that in this project it is sufficient to justify the change by stating the responsible stakeholder. In other words, the two other alternatives of the strategy selection fragment should not be offered. Since two of the three alternatives of the strategy selection fragments are blocked, the chosen alternative needs not to be recorded by the process observation step. Moreover the project manager decides that no dependencies should be created between the stakeholder initiated the changes and the modified or created specification parts.

We use our filter mechanism to adapt the method fragment *integrate change request* according to the new requirements of the project manager. We define

- one strategy filter which blocks the alternatives *select contract parts* and *select design decisions* of the strategy selection fragment *select justification object*;
- one method filter which prevents the execution of the atomic fragment *create dependency*; instead of associating an information filter with the atomic fragment to block its entire output;
- one information filter which blocks the recording of the information about the chosen alternatives. This filter is associated to the *record strategy selection* fragment.

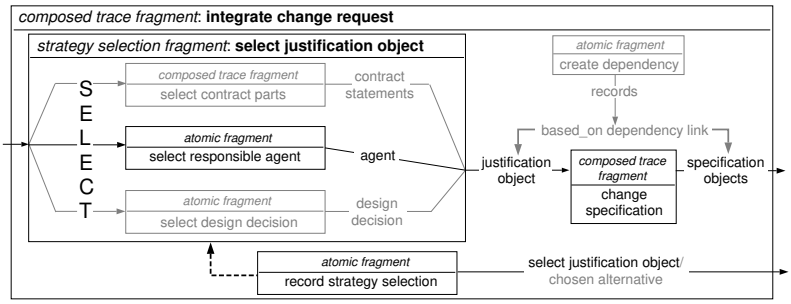

Fig. 9: Adapted trace fragment *integrate justified change* (simplified)

The application of these filters leads to the trace fragment(s) depicted in figure 9. The parts of the trace fragment which are not executed, i.e., prevented by the filters, are depicted in grey. This changes could be achieved without any re–modeling of the composed trace and strategy selection fragments and without any reprogramming of the atomic method fragments.

6 Conclusion and Future Work

Our approach to method–driven trace capture [Pohl *et al.*, 1997] enables the definition of project–specific trace information and trace capture strategies. Based on this definitions the user is guided in capturing the required project-specific trace information. Originating from its application in case studies two main shortcomings of the approach were recognized: adapting trace fragments to varying traces during a project required a significant effort for (re–)modeling and (re–)programming; managing and maintaining the trace fragments became almost impossible due to redundant parts of the introduced fragments and a rapidly increasing amount of trace fragments.

The filter–mechanism presented in this paper avoids the two shortcomings. Based on a set of requirements for trace filters we have defined three types of filters:

- *information filters* block certain information types from being stored in the repository;
- *strategy filters* restrict the alternative trace strategies offered to the user;
- *method filters* prevent a trace fragment from being executed.

A filter can be defined for particular project phases or specific trace fragments. The filter definitions influence the recording of the traces during a project phase and/or during the execution of a trace fragment.

To enforce the recording of certain information we have defined two filter modes: *prevent* and *permit*. We defined propagation rules for nested filters to determine all filters to be applied for a trace fragment whenever it is executed and specified how to resolve resulting contradictory filter definitions.

To support the systematic definition of filters we provided a set of rules for their application.

The filter mechanism was validated by integrating it into the TECHMOD and PRO–ART 2.0 environments and by applying it to small examples. Early experience confirms that trace filters significantly reduce the necessary effort to adapt trace fragments and facilitates the management and maintenance of the method base.

The development of *tool support* for the definition and application of filters will be focus of our future work. Such support should employ the defined rules for applying filters and provide mechanisms to check the effects of filters on the trace fragment definitions.

Acknowledgments

This work was supported by the DFG-Projekt "Prozeß-Integration von Modellierungs-Arbeitsplätzen", the ESPRIT Long Term Research Project 21903 CREWS (**C**ooperative **R**equirements **E**ngineering **W**ith **S**cenarios), and the DAAD/ACLS/NSF program "Technische und empirische Grundlagen der Requirements Traceability: Modelle und Mechanismen".
The authors are grateful to their colleagues P. Haumer, M. Jarke, K. Weidenhaupt, and S. Zlatintsis for many fruitful discussions and contributions.

References

[Ascent Logic Corporation, 1994] Ascent Logic Corporation. RDD–100 Marketing Brochure, 1994.

[Bröhl and Dröschel, 1993] A.P. Bröhl and W. Dröschel. *Das V–Modell*. Oldenbourg Verlag, 1993.

[Collery, 1988] A. Collery. Traceability, the New Strategic Challenge for Companies, its Tool, Character Reading in Industrial Circles. In *Proc. of the 19th Intl. Symposium on Automotive Technology and Automation, with Particular Reference to Cell Control and Quality Management Systems for the Manufacturing Industries*, volume 1, pages 251–260, Monte Carlo, Monaco, October 1988. Allied Automation.

[Conklin and Begeman, 1988] J. Conklin and M.J. Begeman. gIBIS: A Hypertext Tool for Exploratory Policy Discussion. *ACM Transactions on Office Information Systems*, 6(4):303–331, 1988.

[DoD-2167A, 1988] DoD-2167A. Military Standard: Defense System Software Development. 1988. U.S. Dept. of Defense.

[Dömges *et al.*, 1996] R. Dömges, K. Pohl, M. Jarke, B. Lohmann, and W. Marquardt. PRO–ART/CE — An Environment for Managing Chemical Process Simulation Models. In *Proc. of the 10th Europ. Simulation Multiconference*, pages 1012–1017, Budapest, Hungary, June 1996.

[Gotel, 1996] O. Gotel. *Contribution Structures for Requirements Engineering*. PhD thesis, Imperial College of Science, Technology, and Medicine, London, England, 1996.

[IEE, 1991] IEE. *Proceedings of the IEE Colloquium on Tools, Techniques for Maintaining Traceability During Design*. London, England, December 1991.

[ISO, 1991] ISO. *ISO9000–3: Quality Management and Quality Assurance Standards*. International Institute for Standardization, Genf, Switzerland, 1991.

[Jarke *et al.*, 1994] 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, The Netherlands, 1994.

[Kaindl, 1993] H. Kaindl. The Missing Link in Requirements Engineering. *ACM SIGSOFT Software Engineering Notes*, 19(2):30–39, 1993.

[Marconi Systems Technology, 1996] Marconi Systems Technology. RTM (Requirements & Traceability Management) – Marketing Information, 1996.

[Paulk *et al.*, 1993] M. Paulk, B. Curtis, M. Chrissis, and C. Weber. Capability Maturity Model for Software: Version 1.1. Technical Report SEI-93–TR-24, Software Engineering Institute, Carnegie Mellon University, Pittsburg, Pennsylvenia, USA, February 1993.

[Pinheiro and Goguen, 1996] F.A.C. Pinheiro and J.A. Goguen. An Object–Oriented Tool for Tracing Requirements. *IEEE Software*, pages 52–64, March 1996.

[Pohl and Dömges, 1997] K. Pohl and R. Dömges. An Environment for Model–Based Trace Capture. In *Proc. of the Intl. Conf. on Software Engineering and Knowledge Engineering*, Madrid, Spain, June 1997.

[Pohl and Weidenhaupt, 1997] K. Pohl and K. Weidenhaupt. A Contextual Approach for Process–Integrated Tools. In *Proc. of the 6th Europ. Software Engineering Conference*, Zürich, Switzerland, September 1997.

[Pohl *et al.*, 1997] 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.

[Pohl, 1996a] K. Pohl. PRO–ART: Enabling Requirements Pre–Traceability. In *Proc. of the 2nd Intl. Conf. on Requirements Engineering*, Colorado-Springs, Colorado, USA, April 1996.

[Pohl, 1996b] K. Pohl. *Process Centered Requirements Engineering*. RSP by J. Wiley & Sons Ltd., England, 1996.

[Quality Systems & Software, 1996] Quality Systems & Software. DOORS (Dynamic Object Oriented Requirements System) – Marketing Information, 1996.

[Ramesh *et al.*, 1996] B. Ramesh, C. Stubbs, T. Powers, and M. Edwards. Implementing Requirements Traceability: A Case Study. *Annals of Software Engineering*, 9:1–19, 1996.

[Rolland and Grosz, 1994] 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, October 1994. IEEE Computer Society Press.

[TD Technologies, Inc., 1996] TD Technologies, Inc. SLATE (System Level Automation Tool for Engineers) – Marketing Information, 1996.

[Tilbury, 1989] A.J.M. Tilbury. Enabling software traceability. In *Proc. of the IEE Colloquium on The Application of Computer Aided Software Engineering Tools*, pages 7/1–7/4, London, England, February 1989.

[Yu and Mylopoulos, 1994] E. Yu and J. Mylopoulos. Using Goals, Rules, and Methods to Support Reasoning in Business Process Reengineering. In *Proc. of the 27th Hawaii Intl. Conf. on System Sciences*, volume IV, pages 234–243, Maui, Hawaii, USA, January 1994.