Towards Method–Driven Trace Capture[†]

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

Abstract: Traceability is a prerequisite for managing the evolution of (software) systems. Assuring overall traceability of a system development process, i.e., capturing and interrelating all possible data, is almost impossible and by far too expensive and labor intensive. To minimize the information to be recorded and to reduce the additional costs the types of trace information to be captured should be adjusted to project–specific needs, e.g., intended trace usage, time and money available.

In this paper we present an approach which supports method–driven trace capture. The project manager defines the trace information and the trace steps required for recording this information according to the actual needs in explicit traceability (method) models. In addition, the trace steps are integrated with the method definition used to guide the product development process. Based on the so gained extended method definition, the stakeholders are guided in capturing the defined trace information and trace capture is even partially automated. We report on experiments made with the prototypical implementation of the approach and discuss possible extensions.

[†] CREWS Report Series 97–03. A version of this paper appears in the *Proceedings of the 9th International Conference on Advanced Information System Engineering*, Barcelona, Spain, June 18–20, 1997.

1 Introduction

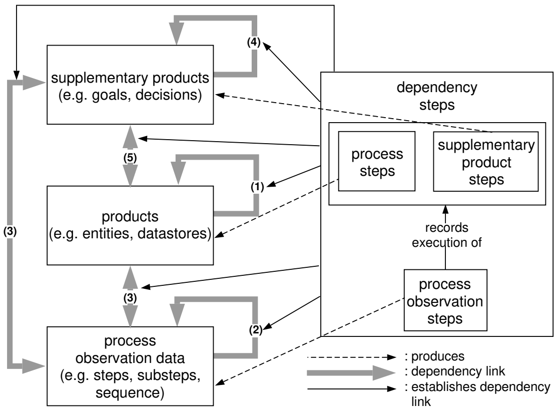
Traceable system development is required by many software development frameworks (e.g., V–Modell [2], DoD–2167A [6]) as well as quality and process improvement frameworks, like ISO 9000–3 [13] and CMM [19]. The importance of traceability, especially in requirements engineering, and the expected benefits are for example emphasized by [12;28;24]. Traceable system specifications are essential for change integration, lead to less errors during system development, play an important role in contracts, and improve the acceptance of the developed system.

Recent research contributions either define a comprehensive framework for traceability (e.g., REMAP [29], PRO–ART [23]) or consider different aspects of traceability, e.g., the annotation of requirements specifications using hypertext (e.g., [17;26]), contribution structures (e.g., [10]), design decisions (e.g., [4]), goals and organizational aspects (e.g., [35;34]), and experience–based process improvement (e.g., [15]). Together they define a comprehensive set of information which have to be recorded for assuring traceability. However, recording all this information during the whole system development process and for all system components is by far too expensive and time consuming. In addition, it is quite likely that a lot of the recorded information will never be used.

To reduce the effort for capturing and maintaining trace information, the *information to be recorded should be defined according to the expected trace usage*. For example, for mission critical system components the recording of the goals, the decisions and the contribution structures may be required, whereas for prototypical components capturing the basic product evolution may be sufficient.

Existing commercial tools (e.g., DOORS [32], RDD–100 [5], SLATE [33]) and prototypical environments (e.g., TOOR [20]) focus on the persistent recording and the maintenance of trace information. In contrast to the above research contributions, they define only a few generic information types which can be specialized by the user of the system. Thus, in principle, existing tools enable the project manager to define the kind of information to be recorded according to the actual needs. But they lack two other important features. First, they provide *no means for defining how, when, and by whom* the trace information should be captured. Most tools provide means for defining the types of information to be recorded but do neither support the definition of the specific method steps (*trace steps*) by which the information is recorded nor the definition of the situation in which the information has to be recorded. For example, a project manager can define an information type for design decisions. But s/he can neither define that design decisions have to be recorded together with their arguments nor when, e.g., for which artifacts, and by whom, e.g., by the project manager, the decisions should be recorded. Second, they provide *no computer–based, active methodical guidance for capturing the trace information*. Reminding the stakeholders about the information to be captured, enforcing or even automating the recording of the information is essential, especially if the information to be recorded varies from project to project. This could be achieved by using the defined trace steps within a computer–based environment to guide the stakeholders in capturing the defined trace information.

Project–specific and method–driven trace capture overcomes the above mentioned shortcomings. It empowers the process–owner to model the trace information and the trace steps to be performed for recording this information according to the actual needs. The integration of the so gained traceability (method) models with existing method definitions and their use in a process–integrated engineering environment provides the basis for partially automated trace capture, and for guiding the stakeholders in capturing the defined trace information.



supplementary products
(e.g. goals, decisions)
products
(e.g. entities, datastores)
process
observation data
(e.g. steps, substeps,
sequence)
dependency
steps
process
steps
supplementary
product
steps
records
execution of
process
observation
steps
(1)
(2)
(3)
(3)
(4)
(5)
: produces
: dependency link
: establishes dependency
link

Fig. 1: Extended product information and extended method steps

In section 2 we classify the kinds of information to be recorded and the method steps by which the information is recorded. We argue that trace steps should be explicitly defined and, in addition, integrated with the methods used for guiding the development process. In section 3 we outline the principle approaches for defining the trace steps and for guiding the stakeholders in their trace capture job according to the definitions. Based on these principle approaches we have implemented two prototypical trace–environments which support the stakeholders in their trace capture job (section 4). Finally, we compare our approach for method–driven trace capture with other research contributions (section 5) and provide an outlook on future work (section 6).

2 Project–Specific Definition of Trace Information and Trace Steps

In this section, we discuss (1) extensions to the concepts of the product under development and the development process which are required for traceability and (2) how these extensions can be related to existing process models (and tools).

2.1 Definition of Extended Product

We classify the information to be recorded for enabling traceability into four categories, namely *product, supplementary product, process observation,* and *dependency* information (left part of figure 1).

Product information includes all the information which belongs to the product under development and is determined by the (set of) method(s) applied during system development. If we use, for example, an Entity–Relationship (ER) method, the product information subsumes the (name of the) ER–diagram, the defined entities and relationships, the associations between entities and relationships, the cardinalities of these associations, etc.

Supplementary product information subsumes additional (product) artifacts which must be recorded to enable traceable system development, for example annotations which provide explanations (e.g., [17]), contribution structures (e.g., [10]), goals and organizational information (e.g., [35;34]), and design decisions (e.g., [4]).

Process observation information subsumes data about the process by which the (supplementary) product has been produced. This may include information about the executed method steps (e.g., a review), the date and the time of their execution, the substeps of complex method steps, the sequence in which method steps were executed, and the agents (i.e., humans and/or tools) performing the process.

Dependency information represents relations between the three information categories defined above. More precisely, dependency links (see left part of figure 1) are used to interrelate

(1) product parts, e.g., to represent that two product parts are inconsistent or contradictory, or to capture a versioning–relationship between product parts;
(2) process observation information, e.g., to link a sequence of steps to the agent who performed it;
(3) (supplementary) products and process observation information, e.g., to associate product parts with the process steps by which they were produced;
(4) supplementary products, e.g., to associate decisions with informal meeting notes which support or object a decision;
(5) supplementary products and products, e.g., to justify that a certain model part was introduced according to a decision or business goal.

We call the four categories of information *extended product (information)* and the latter three categories *trace information*.

2.2 Definition of Extended Method Steps

For recording this information four basic types of (extended) method steps are required (see right part of figure 1).

Process steps record information about the product under development. Existing methods define the guidance provided for the stakeholders during product development through the definition of *process steps* and their ordering. For example, ER–process steps provide guidance for creating, modifying, and deleting ER diagrams, entities, and relationships.

Supplementary product steps define how to create and capture information of a particular supplementary product (part), e.g., decisions and their arguments or contribution structures.

Process observation steps capture information about the execution of process steps and supplementary product steps by monitoring the actual process execution. For example, a process observation step may record all process steps performed during the integration of a change.

Dependency steps capture the dependencies between the artifacts created by process steps, supplementary product steps, and process observation steps. For example, a dependency step may define that the products changed during a particular change integration must be related to the change approval.

We name the latter three types of method steps *trace steps*.

2.3 Automated versus Interactive Trace Capture

The degree of automated trace capture depends on the kind of information to be recorded, i.e., on the type of (extended) method step.

Process and supplementary product steps: Capturing the information about products and supplementary products requires the involvement of a stakeholder. Exceptions are (1) products or supplementary products which can be derived based on the formal product structures, e.g., structural relationships between product parts or between different products like the relations between ER–diagrams and DFDs; and (2) artifacts which are produced through transformation approaches, e.g., by transforming ER diagrams into a relational schema or requirements documents into design documents.

Process observation steps: Information about the execution of process and supplementary product steps can be automatically recorded (by process observation steps). A process observation step monitors the process execution and thus must be aware about the executed process and supplementary product steps, their sequence, and their defined relations. User interaction is only required if the product is created and modified outside the computer–based environment or if information not known within the environment has to be recorded. For example, capturing the agents performing a process step may require user interactions by which the agents are made known to the system (if they can not be deduced, e.g., from the user–login).

Dependency steps: If a dependency can be defined at the *type level*, i.e., it holds for all instances of the type, a particular dependency link between the instances of these types can be automatically created. Similarly, a dependency link can be automatically created if it can be derived from the formal definition of process, supplementary product, and/or process observation steps. For example, a dependency step can define that the process observation data of each process and supplementary product step have to be related to the products which were affected by them. In this case, the dependency link between the artifacts and the process observation data can be automatically created since it was defined on the type level. Likewise, a change integration step may define that all the products created, modified or deleted during its execution are related to the change approval by which the modifications are justified, i.e., the dependency links between the product parts and the change approval can be automatically created.

In contrast, user interaction is required if the creation of a dependency depends on the actual *instance of a type*, i.e., if only a "may–be" relation can be defined at the type level. For example, if informal contractual statements should be related to the design artifacts, the user has to select the parts of the informal statements which should be related to a particular part of the design document. In general, user interaction is required if the creation of a dependency link requires semantic knowledge which is not represented in the product, supplementary product or extended method definitions.

2.4 Extended Methods: Interrelating Trace and Process Steps

To enable the guidance of the stakeholders during system development, it is not sufficient to define the trace steps and the types of information to be recorded. In addition the process owner has to define *when*, i.e., in which (process) situations, the defined trace steps have to be executed. This requires that the trace steps are integrated with the process steps defined by the method. The principal kinds of interrelations are depicted in figure 2 and described below.

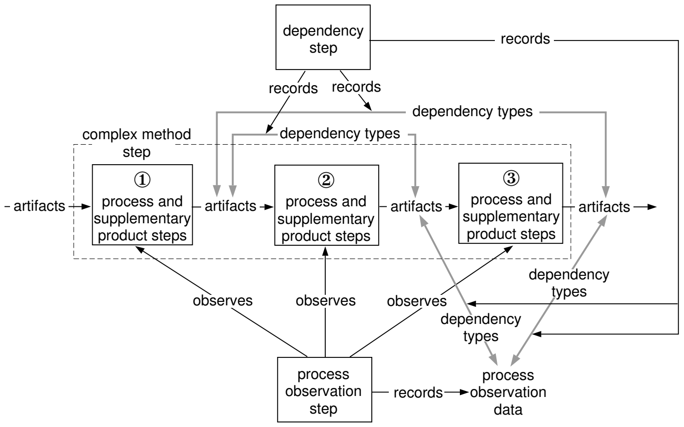

dependency step
records
records
records
dependency types
dependency types
complex method step
artifacts
① process and supplementary product steps
artifacts
② process and supplementary product steps
artifacts
③ process and supplementary product steps
artifacts
observes
observes
observes
dependency types
dependency types
process observation step
records
process observation data

Fig. 2: Interrelating trace and process steps

Process steps and *supplementary product steps* (see middle of figure 2) are composed into complex method steps which guide the user in creating the products and the required trace information. The output of one step is often (partially) used as input of the subsequent process or supplementary product step. For example, the following sequence of steps guides the user in integrating meeting minutes into the product: (1) record the minutes of a meeting; (2) structure the decisions recorded in the minutes; (3) integrate the decisions into the existing products. Each of the three steps can itself be further decomposed. Such complex steps may include branches or loops (not shown in figure 2).

Process observation steps (see lower part of figure 2) monitor and record the execution of process and supplementary product steps. To assure that the relevant observation data is recorded, the process owner has to define which steps should be observed and, in addition, which information has to be recorded for each step. The execution of process observation steps can be automated if the process steps are executed within the computer–based environment. The involvement of the stakeholders is only required if (additional) information not known within the environment has to be recorded, e.g., process steps which have been performed outside of the environment.

Dependency steps (see upper part of figure 2) create dependencies between the artifacts used/produced by a single or a sequence of process and/or supplementary product steps, or between the process observation data and the artifacts used/produced by process/supplementary product steps. Consequently, dependency steps are heavily interrelated with all other process steps. They have to be aware which artifacts are produced and used by each process and supplementary product step and about the actual observation data recorded for a process step.

The interrelation of the various process and trace steps results in the definition of what we call *extended method*.

2.5 Modelling Trace Capture According to the Actual Needs

According to our definition of an extended method, the process owner (e.g. the project manager) must define the (product and trace) information to be captured and the types of extended method steps for recording this information. In addition, through the interrelations of the four types of method steps the process owner must define when a particular process or trace step has to be executed.

There are two complementary ways for defining the trace information and the trace steps according to the project–specific needs. The process owner can define the trace steps to be executed and thereby implicitly define the information to be captured. On the other hand, s/he can define the trace information and retrieve the trace steps by which this information can be captured from a method base. The second way is especially interesting if a large repertoire of trace steps exists. Of course, the process owner can also apply a combination of both.

As a consequence, the modelling language used for defining the extended method must provide means for expressing the different kinds of method steps, for specifying the interrelations between those steps, and for composing them to larger method fragments. Since the execution of an extended method step additionally depends on the current process situation, the language must provide concepts for representing the situations in which a certain step should be executed.

3 Method–Driven Trace Capture

So far we have defined four types of extended product information and the corresponding types of extended method steps by which the information is automatically and/or interactively recorded and interrelated. In addition, we have argued that trace and method definitions have to be integrated to enable guided trace capture.

The aim of this section is twofold. First, we propose a contextual language (process meta model) for defining the trace and process steps and sketch its use for the definition of a project–specific extended method (section 3.1). Second, we outline an approach for a computer–based environment which is able to support the stakeholders in their trace capture job according to the explicit method definitions (section 3.2).

3.1 Defining and Interrelating Trace and Process Steps

It is obvious, especially for creative processes, that the system development method cannot be fully predefined. Instead, method fragments should be defined and, depending on the actual process situation, be activated to guide the stakeholders performing the process (see [24] for details).

A large amount of modelling languages have been proposed from various disciplines (e.g., [9;24;1]). We therefore do not aim in defining a new method definition language. Instead, we propose to use the NATURE process meta model (cf., [30;25;24;16]) for defining the method fragments since it provides concepts for the situative definition of method fragments, empowers an easy integration of trace and process steps due to the uniform definition of automated, semi–automated and composed method steps, and has proven useful for modelling creative processes (see [21;24] for a detailed description and comparison with other languages).

The NATURE Process Meta Model

Figure 3 depicts the key concepts of the NATURE process meta model and their relationships using the OMT notation.

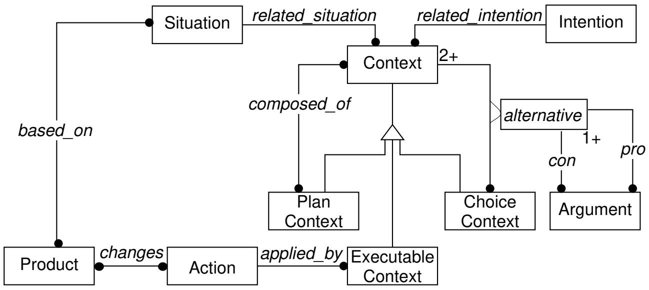


Fig. 3: The NATURE process meta model

A *situation* is built from 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 which is applicable in the given situation. A context is *activated* if its situation is valid and its intention is chosen. The notion of context is further specialized into executable, choice, and plan contexts.

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.

Choice contexts represent the part of the process definition, in which the user has to make a decision. Within a choice context, at least two *alternative*s exist. An alternative can be another choice, executable, or plan context. When a choice context is activated the user can choose one of the alternatives, or even introduce a new one. For each alternative, *argument*s (pros and/or cons) can be provided to guide the user in choosing one of the alternatives.

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 user to deal with the contexts in the order defined by the plan.

Defining Trace Steps Using the NATURE Process Meta Model

Using the NATURE process meta model the process–owner can define the project–specific trace steps by which the required trace information is being recorded as well as the required extended method fragments. For each trace step and method fragment, the process owner must define the situations in which it has to be applied.

Executable contexts are used to define atomic trace steps, i.e., trace steps which can be executed by a tool and by which a certain type of information is being recorded, e.g., process observation data. Executable contexts can be automatically executed (e.g., if embedded in a plan context) or their execution can be initiated by the user performing the process.

If there are alternative ways to record the required information, the process–owner defines the possible alternatives using *choice contexts*. As a result, the user performing the process has to choose one of the defined alternatives. Among others, a choice context can be used to enable the user to decide (depending on the actual process status, e.g., the time and resources available) if fine–grained (defined by a plan context) or coarse–grained (defined by an executable context) trace information are recorded. For example, the process–owner may leave it to the user if a decision is just recorded by an unstructured text, or by stating the available alternatives together with their rationales.

Finally, the process–owner is empowered to define complex trace steps (plan contexts) by combining elementary steps (executable contexts), user selections (choice contexts) and other plan contexts. Plan contexts are also used to integrate trace steps with the method definitions into extended method fragments. For example, a plan context can be used to define the following sequence of extended method steps: (1) record the minutes of the meeting in textual form (complex trace step defined as plan context); (2) structure the decisions made in the meeting (plan context which defines a set of supplementary trace steps required for recording a decision); (3) relate the meeting minutes with the decisions (automated dependency step defined as executable context); (4) integrate the decisions into the existing product (complex plan context which relates method steps, e.g., the definition of an entity, and dependency steps, e.g., the creation of a dependency between the created entity and the recorded decision).

By defining additional contexts and/or removing or adapting existing ones the process–owner is empowered to adjust the trace information to be recorded according to project–specific needs.

3.2 Model–Based Method Execution

Besides the definition of the extended method (trace steps and their integration with the method definition), the adaptation of the user guidance offered by a computer–based environment according to the method definition is essential for providing project–specific trace capture support; especially if the traces to be recorded vary from project to project.

Process–Centered Engineering Environments (PCEEs, [9]) in principle enable such an adaptation. They can be divided into three conceptually distinguishable domains (figure 4a, [24]).

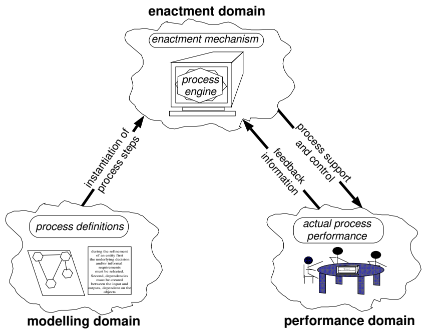

enactment domain
enactment mechanism
process engine
instantiation of process steps
feedback information
process support and control
process definitions
actual process performance
modelling domain
performance domain

Fig. 4a: Three domains of PCEEs and their interactions

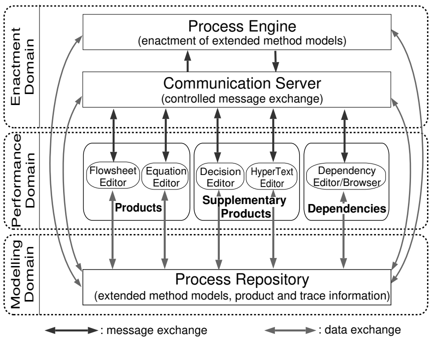

Enactment Domain
Process Engine
(enactment of extended method models)
Communication Server
(controlled message exchange)
Performance Domain
Flowsheet Editor
Equation Editor
Products
Decision Editor
HyperText Editor
Supplementary Products
Dependency Editor/Browser
Dependencies
Modelling Domain
Process Repository
(extended method models, product and trace information)
: message exchange
: data exchange

Fig. 4b: Principal architecture of TECHMOD

The *modelling domain* comprises all activities for defining and maintaining process models using a formal language with an underlying operational semantics which enables the mechanical interpretation of the models. The *enactment domain* encompasses what takes place in a PCEE 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 (e.g., computers).

The interactions between these domains characterize the way in which model–based process support is provided. A process model is first instantiated within the modelling domain and passed to the enactment domain, i.e., process parameters like resources and time scheduling are bound to project–specific values. Based on the interpretation of the instantiated model, the enactment domain supports, controls, and monitors the activities of the performance domain. The performance domain provides feedback information about the current process performance status to the enactment domain as a prerequisite for adjusting process model enactment to the actual process performance and enabling branches, backtracks, and loops in process model enactment.

Thus, this coarse–grained, principal mechanism provided by PCEEs is suitable for supporting process execution based on explicit method definitions.

4 Implementation of Method–Driven Trace Environments

4.1 Process–Integrated Modelling Environments

In addition to the coarse–grained process support provided by PCEEs, fine–grained support is required to enable method–driven trace capture. Hence, as mentioned before, the computer–based tools used to perform the process must adapt their behavior according to the contextual definition of the extended method and must allow the activation of extended method steps by the user. For example, they must be able to acquire additional (supplementary) product information and to create dependency links between product parts as defined in the extended method. These capabilities are not provided by existing PCEEs (see [24] for details).

We have extended the PCEE approach resulting in a framework for process–integrated environments, called PRIME (*PR*ocess *I*ntegrated *M*odelling *E*nvironments, see [27] for a detailed description). In comparison with existing PCEEs, PRIME supports the recording of all four types of extended product information through process–integrated tools which are able to adapt their user interfaces according to the method definition. For example, the PRIME–tools are able to restrict the selectable menu items and product parts according to the choice context definitions, or they request additional information from the user if defined, e.g., as a plan context. Moreover, the enactment domain records the execution of plan contexts and is empowered to influence the behavior of the process–integrated tools according to the actual process situation, e.g., according to the plan context currently being enacted. In addition, the process–integration provided by PRIME empowers the user to initiate the execution of predefined method fragments: the user can initiate the execution of a method fragment which guides her/him in the trace capture task whenever the current process situation requires the recording of a particular kind of trace information.

4.2 Two Prototypical Process–Integrated Modelling Environments

Based on the PRIME approach we have implemented two prototypical *trace environments*: TECHMOD (*T*raced *E*ngineering of *CH*emical process *MOD*els) and PRO–ART 2.0 (*P*rocess and *R*ep*O*sitory based *A*pproach for *R*equirements *T*raceability; [23]). Our overall approach of method–driven trace capture has been validated by using the two environments in small case studies [7;24].

The TECHMOD environment was implemented in cooperation with a chemical engineering department [14;7]. It supports the stakeholders in capturing the defined trace information during the modelling of complex chemical processes according to the extended method definition. The coarse–grained architecture of TECHMOD is depicted in figure 4b.

The *performance domain* of TECHMOD consists of various editing and browsing tools for creating and modifying product parts (e.g., a flowsheet and an equation editor for defining chemical process models), for capturing supplementary products (e.g., a hypertext editor and a decision editor for recording design decisions) and a (graphical) dependency editor for user driven creation and inspection of dependencies. The elementary actions provided by the tools for recording the defined information are associated with executable contexts, while choice contexts are employed by these tools to offer alternative ways for recording the required information.

The *enactment domain* consists of a process engine which interprets the extended method definition (plan contexts), and controls and monitors the tools of the performance domain. During the interpretation the process observation data of plan contexts are recorded by the process engine. The necessary message exchange between the tools of the performance domain and the process engine is controlled by a communication server. The extended method model and the product and trace information are persistently stored in a process repository of the *modelling domain*.

4.3 Method–Driven Trace Capture: A Brief Example

We characterize the support for method–driven trace capture provided by TECHMOD using a small example. The example shows how a chemical engineer is supported in the recording of project–specific trace information according to the extended method definition. It deals with the integration of meeting notes into an existing chemical process model and illustrates the capabilities of TECHMOD in guiding and reminding the stakeholder about the recording of

- supplementary products through the execution of project–specific trace steps;
- the creation of dependency links by automated and interactive dependency steps;
- the recording of project–specific process observation data.

The structure of a chemical process is modelled (like data flow diagrams) at different levels of detail (cf., [18]) using three different classes of material entities: *devices* (e.g., reactors, evaporators; depicted by a large rectangular box) *connections* (e.g., a pipe between an evaporator and a reactor; depicted as filled bars) and *environmental terminals* (depicted as squares with one– or double–sided arrows). A device like a reactor stores extensive quantities like mass or energy and transforms its internal state variables (e.g., temperature, pressure) according to known fluxes acting from the environment on the device. In contrast, a connection provides the fluxes to the adjacent devices according to driving forces determined by the states of those devices. Both, devices and connections, can be decomposed into aggregates of devices and connections of arbitrary complexity. Each elementary device and connection is described by a distinct set of mathematical equations. Environmental terminals represent (like terminators in SA) external entities which receive and/or provide fluxes to the chemical process.

Recording Project–Specific Supplementary Products

The chemical engineer enters the notes of the last meeting using the hypertext editor. After editing, the chemical engineer saves the new minutes. According to the extended method definition TECHMOD reminds the chemical engineer to explicitly structure the decisions stated in the meeting notes using the decision editor. Moreover, the recording of the decisions in an IBIS like structure is guided according to a plan context definition; i.e., the engineer is requested to enter the issue of the decision, the chosen position, the alternative positions, as well as the pro and contra arguments for each alternative (cf., the decision model heat flux exchange, upper left part of figure 5).

Interactive Recording of Dependencies

According to the extended method definition, the engineer is additionally asked by the TECHMOD environment to provide (informal) rationales for the decisions. Since this information cannot be derived automatically from an informal text the chemical engineer has to interactively select the pieces to be linked to the decision (cf., lower right part of figure 5). The selected parts of the meeting notes are then related to the decision by executing a dependency step which creates the appropriate dependency link (cf., dependency editor/browser in the lower left part of figure 5).

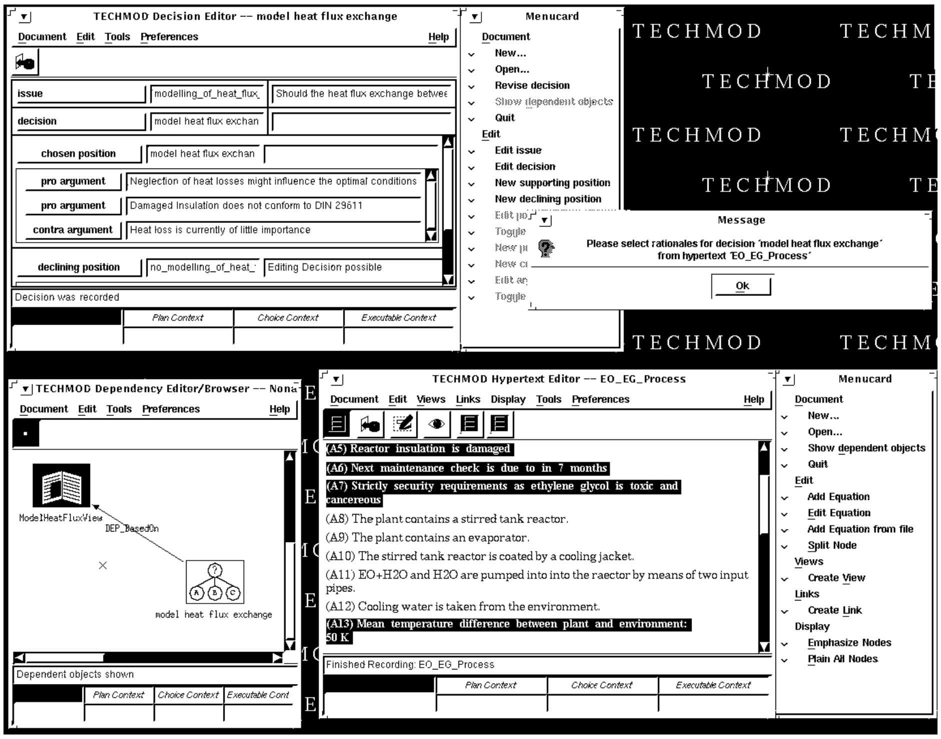
Adapting the method definition according to the needs of another project or of later project phases would automatically result in a different behavior of the TECHMOD environment. For example, if the extended method defines that the recording of the name of a decision is sufficient, the TECHMOD environment will only request the user to enter the name of the decision; instead of providing a detailed decision structure.

Automated Recording of Dependencies

During the integration of the decision into the existing chemical process model (defined by a plan context) all model components changed or created are automatically related to the decision. More precisely a connection heat flux, its corresponding mathematical equations and an environmental terminal heat source would be created and are related to the decision model heat flux exchange through the automated execution of dependency steps defined in the plan context. In other words, these dependencies are derived from the process execution and the creation of the dependencies requires no user interactions.

Recording Process Observation Data

Besides the above mentioned trace information also process observation data is auto-


TECHMOD Decision Editor –– model heat flux exchange
Document Edit Tools Preferences Help
Menucard
Document
New...
Open...
Revise decision
Show dependent objects
Quit
Edit
Edit issue
Edit decision
New supporting position
New declining position
Message
Ok
Plan Context
Choice Context
Executable Context
TECHMOD
TECHMOD Dependency Editor/Browser –– Nona
ModelHeatFluxView
DEP_BasedOn
model heat flux exchange
TECHMOD Hypertext Editor –– EO_EG_Process
Document Edit Views Links Display Tools Preferences Help
Menucard
Document
New...
Open...
Show dependent objects
Quit
Edit
Add Equation
Edit Equation
Add Equation from file
Split Node
Views
Create View
Links
Create Link
Display
Emphasize Nodes
Plain All Nodes

issue: modelling_of_heat_flux_ | Should the heat flux exchange betwee
decision: model heat flux exchan
chosen position: model heat flux exchan
pro argument: Neglection of heat losses might influence the optimal conditions
pro argument: Damaged Insulation does not conform to DIN 29611
contra argument: Heat loss is currently of little importance
declining position: no_modelling_of_heat_ | Editing Decision possible

Decision was recorded

Please select rationales for decision 'model heat flux exchange' from hypertext 'EO_EG_Process'

Dependent objects shown

(A5) Reactor insulation is damaged
(A6) Next maintenance check is due to in 7 months
(A7) Strictly security requirements as ethylene glycol is toxic and cancereous
(A8) The plant contains a stirred tank reactor.
(A9) The plant contains an evaporator.
(A10) The stirred tank reactor is coated by a cooling jacket.
(A11) EO+H2O and H2O are pumped into into the raector by means of two input pipes.
(A12) Cooling water is taken from the environment.
(A13) Mean temperature difference between plant and environment: 50 K

Finished Recording: EO_EG_Process

Fig. 5: Interrelating (informal) rationales with decisions

matically recorded by process observation steps. For example, object representations of the executed process steps are created and linked to the inputs and outputs of the steps. As stated before, the recording of the process observation data requires no user interactions. Among others, the process observation data make the process itself traceable and provide an invaluable source for experience–based process improvement.

Summarizing, the tools of the TECHMOD environment adapt their behavior according to the actual definition of the extended method. Thereby TECHMOD actively reminds the stakeholder about the recording of the defined trace information and automatically records trace information during process execution according to the actual extended method definition, i.e., TECHMOD supports project–specific, method–driven trace capture.

5 Related Work

Method engineering (e.g., [11;1]) and process modelling (e.g., [31;9]) concentrate on the definition of the "classical" system development method and provide typically no means for defining and/or adapting the kinds of trace information to be recorded to project–specific needs. Existing traceability frameworks (e.g., [28;22;10]) provide comprehensive descriptions of the informations to be recorded but offer no means for supporting the recording of these information.

Computer–based environments which support traceability differ in terms of the structure provided for trace information, their abilities to adapt the trace information to be recorded according to project–specific needs, and the degree of guided and/or automated support for trace capture. Environments like DOORS [32], RDD–100 [5], RTM [3], SLATE [33], TOOR [20], and XTie–RT [8] support the definition of project–specific trace information. However, they only provide generic operations for capturing and retrieving the trace information and thus do not support the method–driven capture of the defined trace information. In other words, if existing CASE or traceability tools support the project–specific definition of trace information, the traces must be manually recorded, apart from simple text–parsing mechanisms or transformation facilities, and no guidance is provided for the stakeholders in capturing the defined information.

Process–Centered Engineering Environments partially support the automated and guided recording of trace information, but only on a coarse–grained level due to their inability to adapt the behavior of the interactive tools used for process performance. For method–driven trace capture process–integrated tools, as provided by the PRIME approach, are needed which enable active guidance based on the extended method definitions.

6 Conclusion and Future Work

In this paper we presented an approach for a project–specific and method–driven capturing of trace information. We proposed to explicitly define the trace information together with the trace steps by which the defined information is recorded according to project–specific needs. We distinguished between four kinds of information, namely *products*, *supplementary products*, *process observation data*, and *dependency links* and the corresponding method steps by which this information is interactively or automatically recorded.

We then argued that the defined trace steps need to be integrated with the (classical) method definition to form an *extended method.* The formal definition of the extended method builds the foundation for automated trace capture and for guiding/reminding the stakeholder during the process execution in capturing the defined trace information at the right times.

For defining the extended method we use the NATURE process meta model since it allows a situative definition of the identified steps and empowers an easy integration of trace and process steps due to its uniform definition of automated, semi–automated and composed method steps.

For providing active guidance to the stakeholder and for automating the recording of the trace information we use our PRIME approach, an extension of PCEEs (Process–Centered Engineering Environments). In contrast to PCEEs, PRIME offers process–integrated tools which are able to adapt their behavior according to the extended method definition. This capability empowers a computer–based environment to provide active and project–specific trace capture support for the stakeholders according to an extended method definition.

Based on the PRIME approach, we have implemented two prototypical environments (TECHMOD and PRO–ART 2.0). These environments were used in small case studies which confirmed our assumption that project–specific, method–driven trace capture helps the stakeholders to record the requested trace information, avoids the recording of unnecessary trace information, and due to the automated recording of traces reduces the work load of the process performers.

Besides positive feedback we identified two main issues which will be the focus of our future investigations. First, methodical and tool support is needed for the definition of the trace information, the trace steps and their integration with method definitions. Among others a trace definition environment should support the reuse of extended method fragments. Second, similar to the guidance provided for the trace capture, the user has to be supported in the trace usage. Therefore trace usage has to be integrated into the extended method definition and appropriate visualizations of the recorded (trace) information have to be provided.

Acknowledgments: This work was partially supported by the DFG–Project "Prozeß–Integration von Modellierungs–Arbeitsplätzen", the ESPRIT Long Term Research Project 21903 CREWS (Cooperative Requirements Engineering With Scenarios) and the DAAD/ACLS/NSF program "Technische und empirische Grundlagen der Requirements Traceability: Modelle und Mechanismen". The authors are grateful to M. Baumeister, R. Bogusch, P. Haumer, B. Lohmann, R. Klamma, and K. Weidenhaupt for fruitful discussions and contributions. Major portions of the PRO–ART 2.0 and TECHMOD prototype were implemented by our students Sven Ewald, Markus Hoofe, Tobias Roetschke, Klaus Schreck, and Wolfgang Thyen.

7 References

[1] S. Brinkkemper, K. Lyytinen, and R.J. Welke, editors. *Method Engineering: Principles of construction and tool support — Proc. of the IFIP TC8, WG8.1/8.2 Working Conf. on Method Engineering*, Atlanta, Georgia, USA, August 1996. Chapman & Hall, London, England.

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

[3] Inc. Cayenne Software. RTM (Requirements & Traceability Management) – Marketing Information. WWW–address: http://www.cayennesoft.com/products/rtm.htm, 1996.

[4] 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.

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

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

[7] 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.

[8] Teledyne Brown Engineering. XTie–RT Requirements Tracer – Marketing Information. WWW–address: http://www.tbe.com/products/xtie, 1996.

[9] A. Finkelstein, J. Kramer, and B. Nuseibeh, editors. *Software Process Modelling and Technology*. Number 3 in Advanced Software Development Series. J. Wiley & Sons Inc., Taunton, England, 1994.

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

[11] F. Harmsen, S. Brinkkemper, and H. Oei. Situational method engineering for information system project approaches. In A.A. Verrijn-Stuart and T.W. Olle, editors, *Proc. of the IFIP WG8.1 Working Conf. on Methods and Associated Tool for the Information Systems Life Cycle*, pages 169–194, Maastricht, The Netherlands, September 1994. North Holland, Amsterdam, The Netherlands.

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

[13] ISO9000-3. *Quality Management and Quality Assurance Standards*. International Institute for Standardization, Genf, Switzerland, 1991.

[14] M. Jarke and W. Marquardt. Design and Evaluation of Computer–Aided Process Modeling Tools. In *Proc. of the Conf. on Intelligent Systems in Process Engineering*, Snowmass, Colorado, USA, 1995.

[15] 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.

[16] M. Jarke, C. Rolland, and A. Sutcliffe, editors. *The NATURE of Requirements Engineering*. Springer–Verlag, 1997. (in preparation).

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

[18] W. Marquardt. *Towards a Process Modeling Methodology*, pages 3–40. NATO–ASI Series. Kluver Press, 1995.

[19] 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.

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

[21] V. Plihon and C. Rolland. Modelling Ways–of–Working. In J. Iivari, K. Lyytinen, and M. Rossi, editors, *Proc. of the 7th Intl. Conf. on Advanced Information Systems Engineering*, number 932 in LNCS, pages 126–139, Jväskylä, Finland, June 1995. Springer–Verlag, Berlin, Germany.

[22] K. Pohl. The Three Dimension of Requirements Engineering: A Framework and its Application. *Information Systems*, 3(19):243–258, June 1994.

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

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

[25] K. Pohl, R. Dömges, and M. Jarke. Decision Oriented Process Modelling. In *Proc. of the 9th Intl. Software Process Workshop*, Arlie, Virginia, USA, October 1994. IEEE Computer Society Press.

[26] K. Pohl and P. Haumer. HYDRA: A Hypertext Model for Structuring Informal Requirements Representations. In *Proc. of the 2nd Intl. Workshop on Requirements Engineering: Foundation of Software Quality (REFSQ'95)*, Jyväskylä, Finland, 12. Augustinus Verlag, Aachen.

[27] K. Pohl, R. Klamma, K. Weidenhaupt, R. Dömges, P. Haumer, and M. Jarke. A Framework for Process–Integrated Tools. Technical Report 96–13, RWTH Aachen, 1996. Aachener Informatik Bericht.

[28] B. Ramesh and M. Edwards. Issues in the Development of a Requirements Traceability Model. In *Proc. of the 1st Intl. Symp. on Requirements Engineering*, San Diego, California, USA, January 1993. IEEE Computer Society Press.

[29] B. Ramesh and K. Sengupta. Multimedia in a design rationale decision support system. *Decision Support Systems*, 15(3):181–196, 1996.

[30] 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.

[31] W. Schäfer, editor. *Proc. of the 8th Intl. Software Process Workshop: State of the Practice in Process Technology*, Wadern, Germany, March 1993. IEEE Computer Society Press.

[32] Quality Systems & Software. DOORS (Dynamic Object Oriented Requirements System) – Marketing Information. WWW–address: http://www.qss.co.uk/qss/qss_home.html, 1996.

[33] Inc. TD Technologies. SLATE (System Level Automation Tool for Engineers) – Marketing Information. WWW–address: http://www.slate.tdtech.com, 1996.

[34] A. van Lamsweerde, R. Darimont, and P. Massonet. Goal-Directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons Learnt. In *Proc. of the 2nd Intl. Symp. on Requirements Engineering*, pages 194–203, York, England, 27. 29. March 1995.

[35] 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.