# Experience-Based Method Evaluation and Improvement:

# A Process Modeling Approach*

Matthias Jarke, Klaus Pohl
Informatik V, RWTH Aachen
Ahornstr. 55, 52056 Aachen, Germany
{jarke, pohl}@informatik.rwth-aachen.de

Colette Rolland, Jean-Roch Schmitt
Centre de Recherche en Informatique
Universite de Paris - La Sorbonne
17, rue de la Sorbonne
75231 Paris cedex 05, France
{rolland, JRSchmitt}@masi.ibp.fr

*Little is known about the actual usage and evaluation of methods especially in the early phases of information systems engineering. This paper therefore advocates an experience-based approach in which methods and tools can be defined, applied, evaluated, and gradually improved. We argue that this requires three ingredients:*

- *a process meta model which can deal with many different situations in a flexible, decision-oriented manner;*
- *a process repository that links process and product traces, guidance, and improvement through carefully defined concept mappings;*
- *a tool interoperability concept in which tool behavior adapts to the present process definition and situation, and where tools automatically trace their own behavior.*

*The interplay of these ingredients is demonstrated in the NATURE requirements engineering environment .*

---

* This work was supported in part by ESPRIT Basic Research Project 6353 (NATURE) which deals with Novel Approaches to Theories Underlying Requirements Engineering.

## 1 Introduction

Despite twenty-five years of software engineering research, we know very little about the software development process and the value of the methods we are proposing (Wynekoop and Russo 1993). Methods are often unclearly defined. Even if they are clearly defined, they are not used as intended.

Evaluation criteria for methods are equally unclear and constantly changing. Even assuming we had clearly determined that some of our methods or tools have certain good or bad features, there is no clear-cut answer to the question what to do next: how to change methods or tools without discarding expensive investments in technology and people experience.

What kind of method evaluation and improvement environment would we like to have? On one hand, methods should never get in the way of human creativity and spontaneity. On the other, experience is known to be crucial to success in software projects (Curtis 1992); accumulating and transfering experiences by generalizing from successes as well as mistakes should be supported.

The Software Process Maturity research at Carnegie Mellon's Software Engineering Institute (Humphrey 1990) has shown that experience and understanding of your own work must be the basis for any further steps in method definition and process evolution. This paper proposes an environment based on process modeling and repository technology in which such experience-based process improvement can take place.

The first idea is to link process traceability and process guidance models through concept mappings based on a shared process meta model. The second contribution is how this link can be supported through a suitable repository concept, and through process-adaptable CASE tools which actively support both traceability and guidance.

Our aims are similar to those of the TAME project at the University of Maryland (Oivo and Basili 1992). However, while TAME focuses strongly on quantitative, metrics-based method evaluation mostly of downstream phases in software development, our approach is more geared towards the upstream phases, especially requirements engineering and its relationships to other phases (pre- and post-traceability (Gotel and Finkelstein 1993, Ramesh and Edwards 1993)). As a consequence, much of our approach is more qualitative and the basic process models are much more open, following a paradigm of situated decision making.

The approach has been implemented as part of the NATURE requirements engineering environment (Jarke et al. 1993, Pohl et al. 1994), and is being validated with a number of specific methods and tools. Throughout the paper, we shall use simple running examples from the NATURE prototype for illustration.

In section 2, we present the concept of process repositories used in our approach. It is designed to be compatible with the ISO standard for Information Resource Dictionaries although a different modeling formalism has been used to capture constraints that span multiple levels of abstraction.

Section 3 presents a process meta model for the upstream phases of IS development that we use as a common basis for traceability and guidance models. The specifics of traceability and guidance are discussed in sections 4 and 5. Section 6 analyzes the interaction between traceability and guidance from a method modeling and improvement perspective, and shows the implications of these interactions for the design of CASE tools. Section 7 describes the range of method evaluation applications supported by the approach and draws some general conclusions.

## 2 A Repository-Based Process Management Framework

In this section, we introduce a process-oriented repository framework which forms the basis for our approach. First, we briefly review the underlying structuring concepts of repositories, then we extend them to the case of repositories of process knowledge (short: process repositories), and finally, we give an overview of how our approach exploits this structure for our goal, method evaluation and improvement.

### 2.1 The Information Resource Dictionary Standard

Meta information systems (also called data dictionaries, information resource dictionaries, or repositories) are information systems about information systems. We shall use the term "repository" which was popularized by IBM´s AD/Cycle initiative (Sagawa 1990). More precisely, we start from the Information Resource Dictionary Framework Standard (IRDS) shown in figure 1 (ISO/IEC 1990).

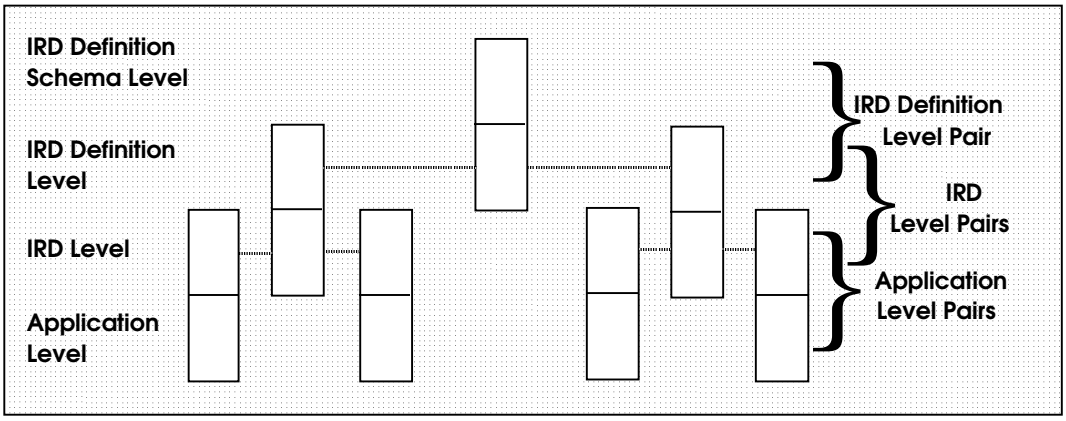


Figure 1: ISO IRDS repository framework

ISO IRDS is organized along the classification dimension of semantic data model abstractions. In a layered type universe of four levels, level *n+1* (called the *defining* level) constitutes a type system for level *n* (the *defined* level). In other words, it defines the language in which level *n* can be specified. In increasing order of abstraction, the four levels are the *Application Level*, the *IRD Level,* the *IRD Definition Level,* and the *IRD Definition Schema Level*.

For an example, consider a library system development. The three lower IRDS levels might include, respectively, descriptions of actual books and journals, an entity type *Publication* with specializations *Book* and *Journal*, and a definition of modeling terms in the ER model, such as *Entity*, *Relationship*, and *isA*.

ISO IRDS additionally promotes a so-called "level pair" architecture in which the relationships among the four levels are defined by a hierarchy of three SQL database specifications on top of each other.

*Application Level Pairs* correspond to the actual applications to be integrated; for instance, the library database and various application programs running on it.

Dictionaries or *IRD Level Pairs* contain as their data the schemata of the application level pairs, and definitions of the underlying data models as their schemata; for example, the dictionary describing the design of the library database as an entity-relationship model, under a schema that defines the structure and constraints of ER models.

A single *IRD Definition Level Pair* contains descriptions of the dictionary schemata as its instances and a fixed way of defining such schemata as its own schema. This fixed way is defined in the standard by a set of SQL tables, together with some basic features of the dictionary level pairs. In our example, these tables would contain tuples which say what an Entity or a Relationship is.

Each kind of level pairs can be associated with a group of people who mostly work with it. We call them *users, application engineers,* and *method engineers,* respectively.

If we want to enforce the semantics of data models at the level of instances, we need to be able to specify constraints that span multiple levels of abstraction. One example of such a constraint is the definition of a functional role in ER diagrams: each instance of an entity with such a role must be related to at most one instance of the relationship type this role leads to. Such "meta" constraints are defined at the IRD Definition level and used at the IRD level, but must be enforced at the Application level. They cannot be expressed in SQL. We have therefore been using the conceptual modeling language Telos which supports meta constraints (Mylopoulos et al. 1990, Jeusfeld 1992).

## 2.2 Process Repositories

As a consequence of the level pair concept, application systems as well as dictionaries can be distributed and heterogeneous. Looking at figure 1 in a different manner, we can see three overlapping environments. Firstly, two *application environments*, each coordinated by a dictionary (cf. figure 2), form the environment available at runtime.

Overlapping with the application environments, we also see a *development environment,* consisting of the IRD Definition Level Pair and the two dictionaries. This is where the development process is managed.

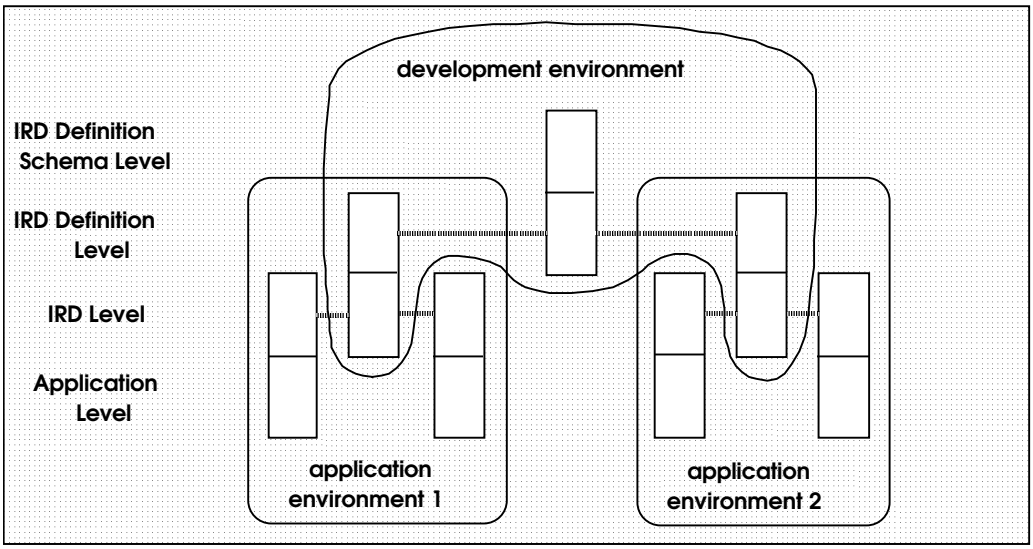


Figure 2: Application and development environment within the ISO-IRDS framework

To be consistent with the application systems coordinated by the dictionaries, the development environment must, of course, contain information about the product of IS development -- the application data structures, programs, interfaces, and maybe manual operating procedures. However, the development environment must also contain information about intermediate results, human agents and tools involved, process plans, design decisions, and steps taken to execute them. In short, the product respository must be extended to a process repository.

Ignoring for the moment the distributed nature of the development environment, figure 3 shows what kind of process information is defined at what level of the IRDS hierarchy.

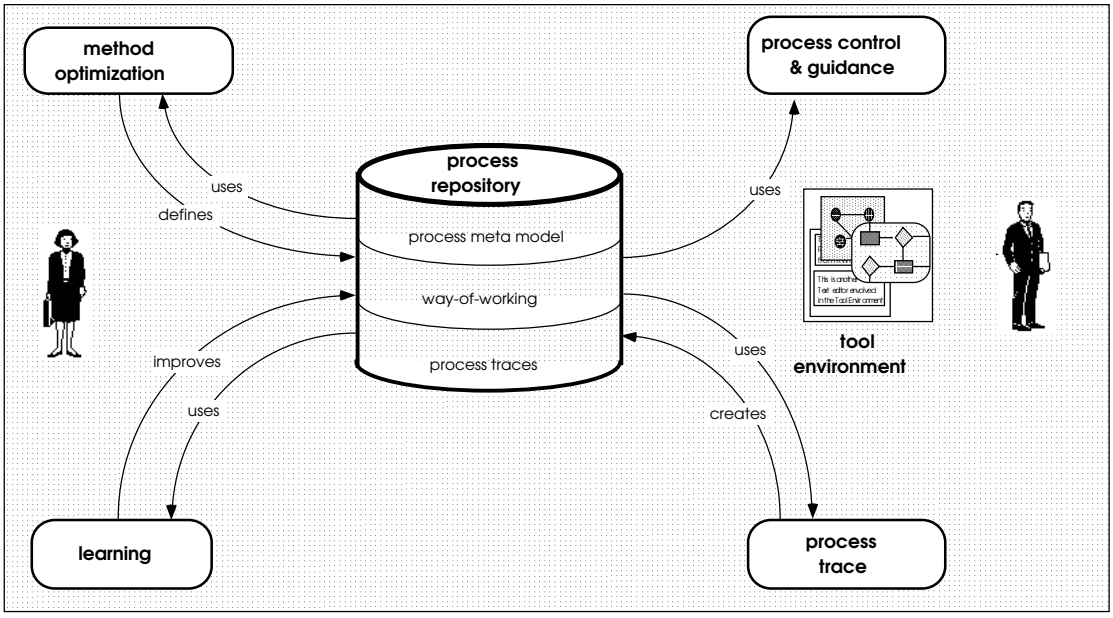


Figure 3: Process repositories and process-related activities

The IRD level (lowest in figure 3) stores the trace of the actual process; part of this trace are descriptions of the product (DB schemas and program code, for example).

The IRD Definition level defines the way-of-working models under which the trace is recorded. There can be two conceptually different kinds of such models: descriptive and prescriptive ones. Descriptive ones, from now on called *traceability models,* offer a design record structure by which the process trace is to be organized. Prescriptive ones, which we call *guidance models,* define the intended way-of-working in the process. For our purposes, it is important that both structures should not be the same, as we want to exploit discrepancies for method evaluation and improvement.

However, the two structures must at least make the actual trace and the intended trace comparable. Traceability and guidance models are therefore organized according to guidance and traceability meta models which are specializations of the same basic meta process model which is fixed at the IRD Definition Schema level (see also figure 4).

How is all this related to the topic of this paper, method evaluation and improvement ?

The Software Engineering Institute at Carnegie Mellon University claims that the maturity level of an IS development organization can be characterized through the way how the process of gradual system evolution is managed (Curtis 1992). SEI assesses the process maturity level of organizations by questionnaires followed by in-depth interviews and site inspections. We are more interested in the specific repository implications of their model. Briefly, each such level can be associated with additional kinds of actions on the process repository, as shown in figure 3.

At the *initial level*, the process is ad-hoc and no experience is transfered from one project to the next. This corresponds to the case where there is no process repository.

At the *repeatable level*, the organization understands sufficiently well what it is doing to have repeatable processes. Technically, this can be supported by capturing process traces according to some record structure and making them accessible for reuse. This is indicated in the lower right of figure 3.

At the *defined level,* process prescriptions abstracted from these collected experiences are enforced through process control and guidance (upper right of figure 3).

Once this has been established, the organization can introduce procedures for measuring the actual process execution against the defined process, thus promoting continuous organizational learning -- the *managed level* (lower left of figure 3).

Finally, there may be also an *optimizing level,* in which continuous goal-oriented and measurement-based optimization of process plans takes place (upper left of figure 3).

In the sequel, we shall be interested in supporting the managed level by relating process tracing and process guidance.

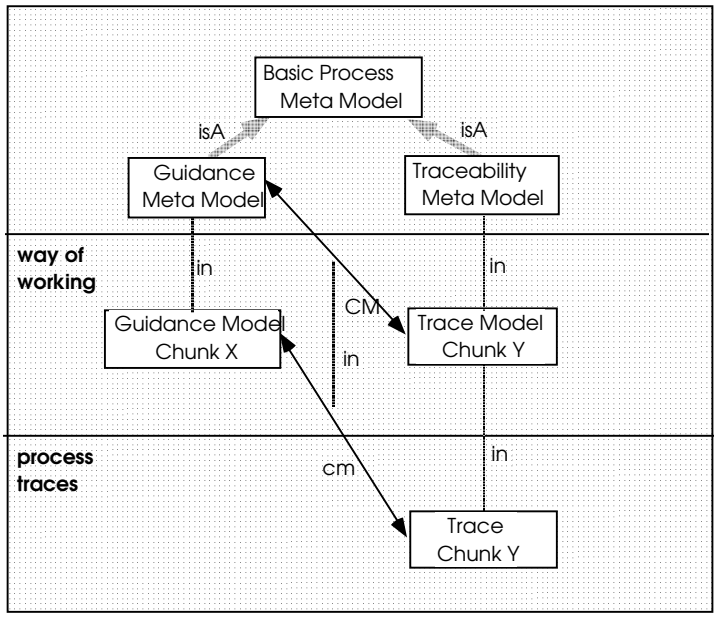


Figure 4: Concept mappings between guidance and trace models

## 2.3 Concept Mappings Between Guidance and Traceability

Figure 4 shows the basic principle of our approach to relating guidance and traceability. The basic process model introduces concepts considered critical for the management of creative processes, such as products, situations, decisions, actions, arguments and contexts. The main additions needed for the guidance meta model are granularity abstractions and control structures for organizing the process, whereas the main additional concept of the traceability model is that of processors (human agents or tools) who execute a process in a specific context at a specific time.

A prescriptive process definition or method is then an instance of the guidance meta model which we call a guidance chunk (or sometimes simply process chunk).

Similarly, the traceability meta model can be instantiated to trace model chunks, i.e. methods to organize a descriptive recording of process executions.

The main issue of interest in this paper, however, is how guidance and traceability must be *related* to each other. From figure 3 we conclude that at least two relationships must be supported. In *process execution* , we must find a way to bring the guidance chunk to bear on the actual process, and to trace the relationship of the actual process execution to the guidance chunk. In *process improvement*, we must be able to derive changes on guidance chunks from observations of the trace.

In both cases, we need a *concept mapping* between a guidance chunk (defined at the way-of-working level) and a trace chunk (defined at the process traces level), as indicated by the *cm* double arrow in figure 4. Read from left to right, this mapping implies how the guidance chunk influences the process execution and thus its trace. Read from right to left, the concept mapping implies how a guidance chunk is derived from the trace.

This concept mapping does not come out of nowhere. It would in fact be close to impossible if guidance meta model and traceability models were developed completely independently. On the contrary, the traceability model has to be constructed in a way that a "way-of-concept-mapping" (denoted *CM* in figure 4) can be naturally defined for both directions -- guidance and learning.

The former direction corresponds to the definition of a concept how the guidance model influences process behavior and thus process traces. In the NATURE project, we have developed a concept of process-adaptable, interoperable, and automatically traced CASE tools for this purpose. For this reason, the CASE tool environment has been placed directly in the process loop on the right side of figure 3, rather than going indirectly only through the human application engineer as in current environments.

The latter direction corresponds to the definition of "improvement methods" for processes, as discussed in (Jarke and Pohl 1993). We shall show later that such methods can be treated with the same guidance model as any other methods.

Actual concept mappings (*cm* in figure 4) are then identified as instances of the identified chunks of concept mapping classes (*CM* in figure 4), making the influence of guidance relatively easy to trace, and focusing the search for method evaluation and improvement opportunities.

The traceability model must therefore be constructed with two goals in mind. First, it must be derived from the guidance meta model, as just discussed, in order to support concept mapping. Second, it must represent a basic theory about the domain in which the process takes place, so that information of possible interest for method evaluation and improvement is actually captured (and not much more, for efficiency reasons!).

In the remainder of the paper, we shall demonstrate how this basic approach has been realized for the task of requirements engineering in the NATURE project. We believe that many of the models and methods presented generalize beyond the domain of requirements engineering but have not validated that claim in much detail yet.

## 3. The Basic Process Meta Model

When we want to evaluate and improve a wide array of methods empirically, we first of all need a meta process model by which not only the guidance offered by arbitrary method fragments, but also chunks of observed behavior can be structured and interpreted in a meaningful way. The model mus therefore bind process chunks to some context on which the application engineer might want to focus, offer guidance in this context through possible actions and advice on their evaluation criteria, but also leave room for human decision-making and extensions.

An in-depth study of the state-of-the-art showed that existing process models can be classified into three categories (Dowson 1987):

- activity-oriented models,
- product-oriented models,
- decision-oriented models.

*Activity-oriented* models come from an analogy with problem solving, i.e. finding and executing a plan of actions leading to the solution. They are sequential in nature and provide a frame for manual management of projects developed in a linear fashion. They are based on the assumption made by Osterweil (1987) that development processes can be programmed and thereafter executed and mainly differ on the type of language used to express process programs. Frequent types of such languages are algorithmic (Williams 1988), first-order predicate logic (Ben-Arieh 1988) or Petri nets (Deiters et al. 1989).

*Product-oriented* process models strongly couple the definition of the development product and the definition of the activities that make it evolve. They allow the specification of the possible states of the product and relate the execution condition of the transformation activities to these states. The models presented in (Humphrey and Kellner 1989; Finkelstein et al. 1990; Peugeot and Franckson 1991) belong to this category.

A more recent class of process models follows a *decision-oriented* paradigm. The successive transformations of the product are looked upon as consequences of decisions. The models presented in (Potts 1989; Jarke et al. 1990; Desclaux et al. 1990; Lee 1991) belong to this class of models and are often based on the IBIS model (Kunz and Rittel 1970). These models are semantically more powerful than the previous ones because they explain not only how the process proceeds but also why transformations happen.

Considering the limitations of the first two classes of process models, we chose to base our process modeling approach on the decision-oriented one. However, even the decision-based approaches offer only limited hints about when and how to decide on what. We recognize that application engineers react *contextually* according to the domain knowledge they acquire and react by analogy with previous situations they have been involved in. In order to take into account this very nature of the RE we have chosen to emphasize the contextual aspect of decisions by strongly coupling the context of a decision to the decision itself. The NATURE approach aims at capturing not only activities performed during the process but also why these activities are performed (the decisions) and when (the decision contexts).

Figure 5 introduces the key concepts of the basic process model and their relationships with a binary entity-relationship (ER) notation. This approach is an extension of the one presented in (Grosz 1990) and implemented in the ALECSI prototype (Rolland and Cauvet 1991). The five basic concepts are the ones of situation, decision, context, action and argument.

A *situation* is most often a part of the product it makes sense to make a decision on. Situations are thus built from parts of the product undergoing the development process. At the beginning, the product is made of the collection of the user requirements. At the end, the product is a specification of the IS that satisfies some quality criteria.

A *decision* reflects a choice an application engineer makes at a given point in time. A decision can be looked upon as the intention of product transformations.

A *context* is the association of a situation and a decision made on it.

An *action* performs a transformation on the IS description. It is the materialization of a decision. Performing an action changes the product and may generate new situations, subjects to new decisions.

An *argument* allows to motivate the making of a decision on a given situation or to object to this decision making.

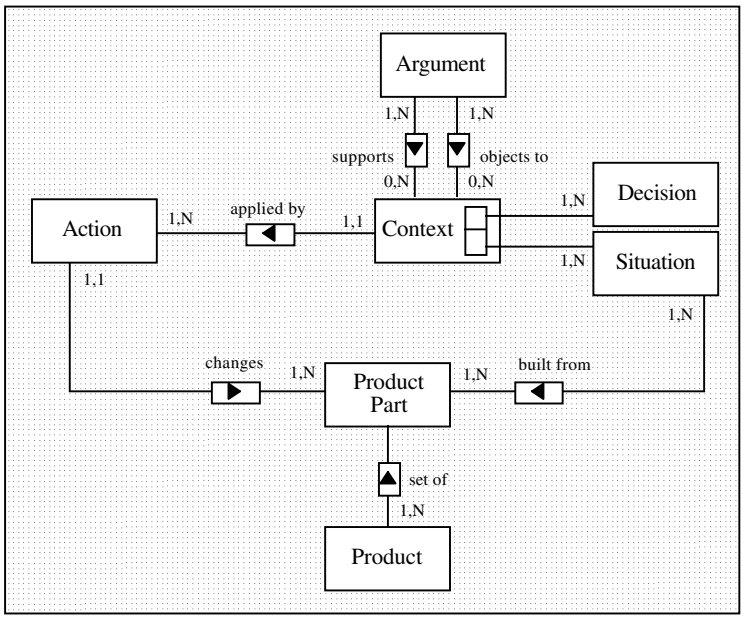


Figure 5: The basic process model

Returning to the library example, assume that the entity-type *Publication* has been created within the ER schema. The application engineer ascertains that this entity-type represents real-world entities that are indeed slightly different : books are characterized by their authors and their editor while journals are characterized by their publisher. One way of expressing this difference is to add corresponding attributes to the entity-type *Publication*. This possible transformation is expressed in the way-of-working definition by a context whose elements and argument are the following:

| | |
|---|---|
| situation : | an entity-type; slightly different variants |
| decision : | define discriminant attributes for the entity-type |
| argument : | the attributes to be added distinguish slightly different real-world entities represented by the entity-type |

## 4 The Process Guidance Meta Model

The application engineer follows a method which prescribes the use of models and specifies the route he can or must follow to transform the initial users' requirements into a final product that satisfies some quality criteria. The prescription of this route is the definition of the way-of-working of the method.

We call *guidance* the capability of the development environment to support the application engineer to proceed according to the way-of-working definition.

The way-of-working definition stored in the process repository can be a collection of "reusable process chunks" (Rolland and Prakash 1993). The guidance mechanism is basically a pattern-matching engine which matches product parts against situation-types defined in the repository and suggests decisions according to the chunk definitions.

The basic process model lacks important features that limit its use for efficient guidance. Firstly, all decisions of a way-of-working are equally treated, although some of them are more complex that others. Secondly, the process modeling concepts of the basic model do not allow the representation of heuristics that can greatly help the application engineer in his decision-making process. Finally, the basic process model is not capable of representing different approaches such as the top-down or bottom-up ones. These approaches have impacts on the ordering of decisions to be made.

According to the extended guidance meta-model, the guidance is based on goals. Clusters are first selected according to goals application engineers want to achieve (and to the mode they choose). Then, within a selected cluster, chunks are executed.

Further, contexts are subtyped into micro-contexts and macro-contexts. Macro-contexts are associated to their alternatives, supported or objected-to by choice criteria (combinations of arguments).

The model concepts and their relationships and summarized in figure 6. The white boxes correspond to the basic process meta model concepts; the others are the extensions and are described in detail below.

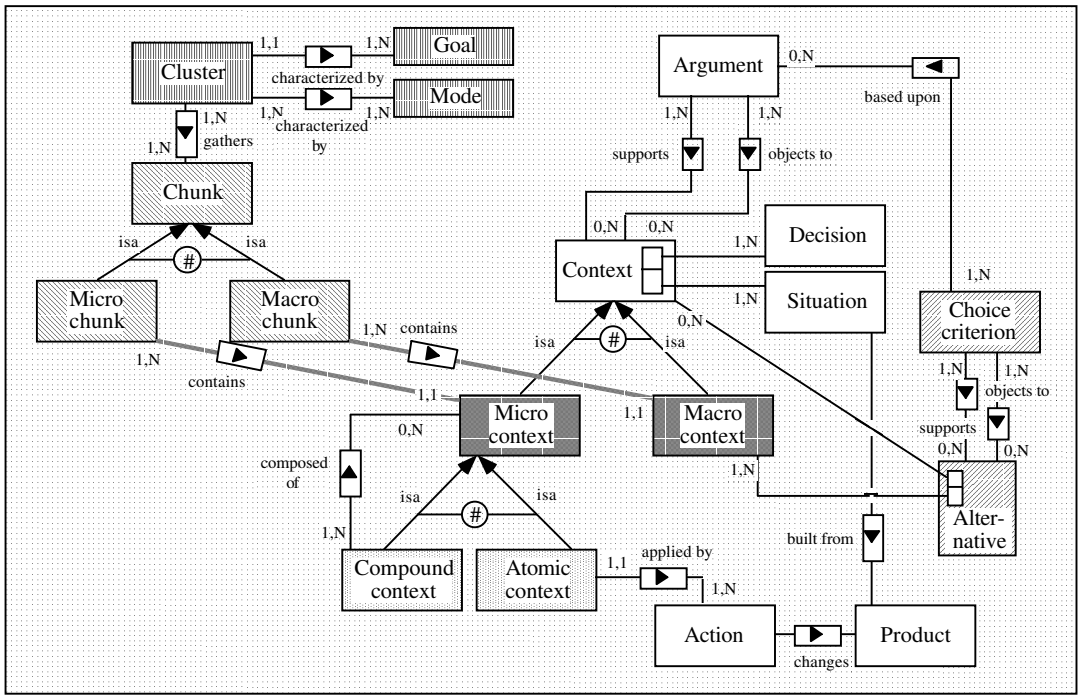


Figure 6: The guidance meta model

## 4.1 Handling Granularity in Decision-Making

In the basic process model, all contexts are treated the same: they all are applicable by the performance of actions modifying the product. This view is a simplification of the kind of contexts that are encountered in ways-of-working. In fact, it is true that some contexts are directly applicable by actions, like the definition of a role cardinality. But other contexts are more complex. For example, the partitioning of an entity-type into several, mutually exclusive subtypes is decomposed as follows:

(1) subtype the entity-type
  (1.1) create a new entity-type
  (1.2) create the isA link between the initial entity-type and its new subtype
(n) repeat (1) as many times as there are subtypings to be done
(2) constrain the isA links by a disjunction constraint
  (2.1) create the disjunction constraint
  (2.2) link the disjunction constraint to the isA links

To handle this kind of granularity in decisions, we refine the *context* concept of the basic process model by *atomic-context* and *compound-context* concepts. A compound-context is a context composed of contexts; contexts that cannot be decomposed are atomic-contexts. We represent the fact that component contexts can themselves be compound, leading to composition-based hierarchies of contexts, by adding a *composed of* relationship between *compound-context* and *context*.

The second refinement of contexts is the creation of two new subtypes, *micro-context* and *macro-context.* This extension of the basic process model intends to represent the fact that some contexts can be applied by a unique set of product transformations while other contexts have several alternative ways to be applied. Contexts of the first kind are called micro-contexts (atomic- and compound-contexts as introduced above are of this kind) . Contexts which require that the application engineer chooses among a set of candidate alternatives are called macro-contexts.

For example, the context whose decision is to add an entity-type in a schema may have as alternatives the one that creates the new entity-type from scratch and another one that reuses an entity-type already created in the course of a previous RE process. In turn, this latter "reuse" context may have as alternatives the one to perform a "domain-based reuse" (Maiden and Sutcliffe, 1992), and the one to perform a "similarity-based reuse" (Spanoudakis and Constantopoulos, 1993).

Alternatives of a macro-context are contexts too, thus contexts may be involved in an alternative-relationship, leading to alternative-based hierarchies of contexts. Along with the specialization of contexts, this extension to the process model is ensured by the *alternative* relationship between *macro-context* and *context*. Choice criteria are associated to alternatives to help the decision-making process by supporting or objecting to the different alternatives of a given macro-context. These choice criteria are combinations of the arguments attached to the contexts as presented in section 3.

The introduction of alternatives and choice criteria allows the representation of process heuristics that help the application engineer by providing him with several motivated ways of applying decisions.

### 4.2 The Chunk as the Elementary Process Element

With the process model extensions we just presented, ways-of-working can be defined as collections of micro-contexts (atomic or compound ones) and macro-contexts.

In the terminology of (Feiler and Humphrey 1993), atomic-contexts correspond to "process steps" (they have no visible substructure) while the composition- and alternative-hierarchies introduced above correspond to "process elements" : they are built from contexts like "elements" are built from "steps".

With these composition and alternative hierarchies, decision-making is guided as a process of hierarchical decomposition of compound- and macro-contexts. A compound-context is applied through the successive application of all its component contexts, until atomic-contexts are applied. A macro-context is applied through the recursive choice of its alternatives.

We call these hierarchies *chunks*. A micro-chunk is the composition hierarchy of a top-level compound-context. A macro-chunk is the embedding of the hierarchy of alternatives of a top-level macro-context and of the choice criteria associated to each level of the hierarchy.

With this introduction of the *chunk* concept as the new unit of process definition, guidance models (prescriptive ways of working) are defined as collections of chunks.

## 4.3 Modeling Process Element Flows

The last kind of way-of-working knowledge we need to model deals with the way chunks (process elements) can be meaningfully grouped and ordered.

We believe it is relevant to group chunks that together accomplish a common goal. We thus introduce the *cluster* concept into the process model.

For example, consider the goal to complete a schema in an object-oriented method providing such concepts as "Class" and "Event" (e.g. the O* method (Brunet 1991)). This goal is reachable through the specification of all the classes and all the events of the schema, i.e. through the application of the corresponding two macro-chunks. These chunks are thus gathered into a "Complete_Schema" cluster.

Furthermore, keeping the same example, the way-of-working may either prescribe that all classes must be specified before the specification of the events can be undertaken, or it may prescribe that classes and events specifications can be undertaken in parallel. In other words, the way-of-working may prescribe that classes and events can be specified either in a depth-first manner or in a width-first one.

Similarly, the way-of-working may either prescribe that classes must be specified by creating them before defining their attributes; or it may prescribe that attributes must first be defined and thereafter grouped to create classes. Here, the way-of-working may prescribe that classes can be specified either in a top-down manner or in a bottom-up one.

This example shows that a given goal may be accomplished differently according to several possible "modes". In the example, two mode pairs can be recognized: depth-first/ breadth-first and top-down/ bottom-up. The modes of such pairs are exclusive — they cannot be followed simultaneously.

We model this influence of modes by characterizing clusters not only by their goal but also by the particular *modes* combination they induce. For a given goal and the corresponding same set of gathered chunks, there may be several clusters for several combinations of modes.

### 4.4 Example

The introduction of the *chunk* and *cluster* concepts allows, in particular, the definition of ways-of-working that embed heuristics and that are not limited to technical aspects.

To be able to express heuristics is an important improvement in way-of-working definitions. It allows application engineers to be much more involved in the process, because their decision-making is made more explicit and is supported by the method.

Returning to the example of section 3, there is a second, alternative way to distinguish the real-world entities, that is to partition the entity-type *Publication* into mutually exclusive subtypes of entities. According to our process guidance model, a macro-chunk can be defined that embeds the top-level macro-context (aiming at distinguishing the real-world entities) and its two alternative contexts, along with the corresponding choice criteria. The first alternative is the context defined at the end of section 3.

The definition of this macro-chunk is as follows:

situation : an entity-type with variants
decision : model variants of the entity-type
1st alternative context :
  situation : the entity-type; slightly different variants
  decision : define discriminant attributes for the entity-type
  choice criteria : PROs : leads to a fewer number of types in the schema
    CONs : the real-world entities are to be differently treated
2nd alternative context :
  situation : the entity-type; variants treated different within the system
  decision : partition the entity-type into subtypes
  choice criteria: PROs : better representation of the real world
    CONs : greater complexity of the schema

We also believe that modeling the RE process according to our approach is not limited to technical aspects, but can also tackle project-management issues. For example, a macro-chunk could be defined to handle a situation in which a project deliverable completion exceeds its planned deadline. Then, two possible alternatives could be to hire a highly-skilled manager to control the RE team or to double the team size. Corresponding choice criteria would then be associated to these alternatives.

## 5. The Traceability Model

Traceability is an important issue in IS development processes for many reasons (Ramesh and Edwards 1993): product change and reuse support; identification of mission-critical requirements and components; recording of design rationale; project

tracking and cost estimates; responsibility and accountability for achieving requirements; interdependencies between documents and components within and across life-cycle stages. All of these issues influence the design of traceability models tracing mechanisms. Here, we focus on traceability as a means for experience-based process improvement.

During process execution, process data are recorded. This trace builds the basis for experience-based process improvement later on. Hence, it is very important to cover all the information needed for improving the method definition.

First of all, it is obvious that the structure provided for recording the trace information (defined at the same modelling level as the guidance chunks, cf. figure 4) must be broader than the current definition of the way-of-working, the method definition.

Second, a framework is needed which helps in identifying the kind of information which must be recorded during process execution. For the case of requirements engineering we have developed such a framework which is described in section 5.1.

Third, the information to be recorded must be structured according to its content as well as according to the guidance structure it relates to. For example, methods are executed by humans who need freedoms to organize their way-of-working and collaborating. The process of such collaboration can be captured by recording conversations. All information, e.g., arguments exchanged in a particular conversation, must be recorded and related to the conversation they belong to. Only if this is done, the conversation can be re-constructed. In section 5.2 we briefly describe the meta models developed in the NATURE project for context-oriented recording of the execution of requirements engineering processes.

Fourth, there should be a graceful degradation of the trace, when application engineers do not follow the defined method. Exceptions and aberrations must be expressible and rationalizable. Generalization hierarchies of product and process models seem to provide a basis for graceful degradation of information. Moreover, the experience covered by the process trace, e.g., exceptions, can only be reflected on the current process definition, if the trace is related to the process definition.

Fifth, capturing process trace adds to the work load of the people. Additionally, the people with the additional workload share only a small part of the pay off. Hence, the capture of the trace must be automated at least partially.

## 5.1 A Framework for Requirements Process Tracing

Within our basic process framework, the traceability model must be able to characterize, in a very broad way, the principal kinds of situations an application engineering team could be in, the kinds of decisions they might face, and the impact their actions could have on the situation. Moreover, there is an important difference between a method and a process trace. While a method is general, a process is always associated with processors who execute it. In requirements engineering which we consider here as an example, these processors are teams of humans with only limited automated support.

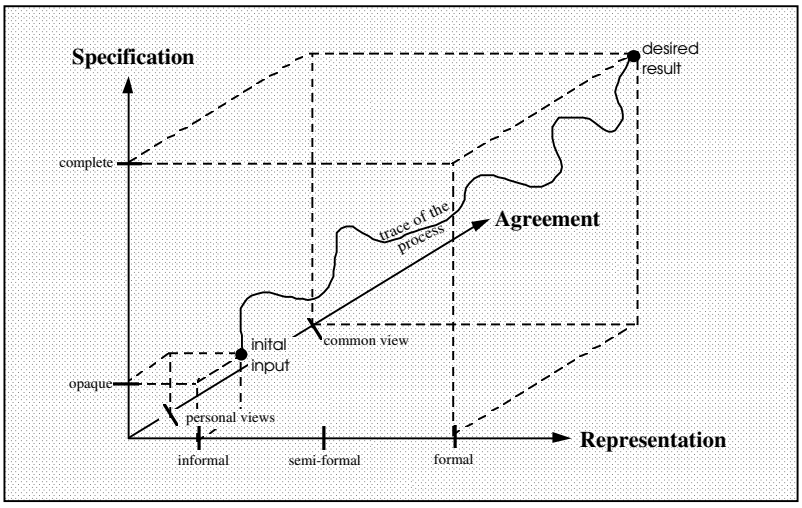


Figure 7. The three dimensions of requirements engineering.

Within the NATURE project, we have developed a framework for requirements engineering processes (Pohl 1994). This framework derives three dimensions of interest from the major sources of problems in requirements engineering (cf. figure 7):

- As a basis for formal system development, requirements engineering moves along a *representational axis* typically from informal to more formal representations, ideally ending with a formal specification that can be transformed to executable code. Modeling and relating these representations is mostly a technical problem albeit one whose solution needs good knowledge of human-computer interaction.
- A second obvious goal is to end up with a complete system *specification*. We see this as orthogonal to the first problem: it is perfectly possible to hide poor understanding behind a lot of formalism, as well as to describe a very deep understanding in quite informal terms. Moves along this axis mostly face cognitive and psychological problems of requirements engineering.
- The third axis is concerned with the *agreement* reached on the current specification. Requirements engineering is understood here as a negotiation process which should lead towards at least sufficient agreement to start building a system. Hence this axis mainly deals with the social aspects of requirements engineering.

At the level of process execution, requirements engineering can be understood as a movement in this space which, in addition to the technical, cognitive, and social barriers, is also influenced by economical factors and the methods used. The process trace is understood as an abritrary line within the cube spanned by the three dimensions. Hence, the execution of the process can be recorded by capturing and interrelating the process knowledge along the *representation, specification* and *agreement* dimension.

## 5.2 Models for Recording the Trace

According to this framework, the process situation can be characterized as a position in the three-dimensional cube, the effect of a process action as a move within this space.

Guidance can be understood as advice on what alternative moves are available in particular situation patterns, and what consequences these moves may have.

To make this framework operational, the traceability model elaborates each dimension into a set of class definitions that characterize points and relationships along this dimension by Telos structures, deduction rules, and integrity constraints. The traceability model in the current NATURE prototype comprises about 50 class definitions. The process trace instantiates these class definitions. Each trace object can be instance of multiple classes, and dependencies along each dimension document a process step. Rules and constraints ensure the formal correctness of the trace.

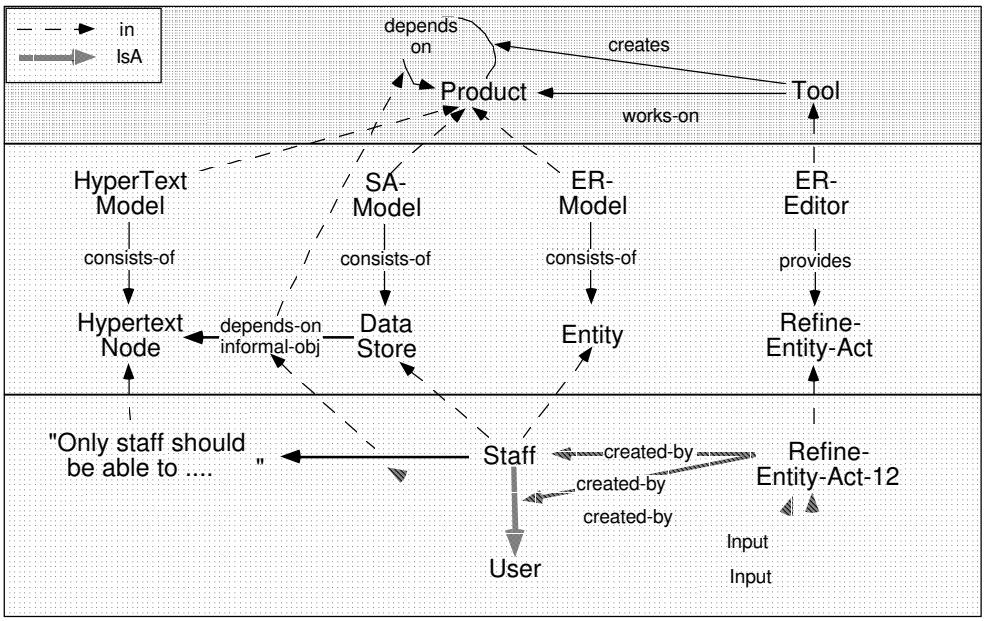


Figure 8: An simplified example of process trace

Figure 8 illustrates the model structure with a simple example from the representation dimension. The basic process model is extended by the concept of a processor (*Tool*), and by dependencies among products caused by actions of tools. The meta concepts are instantiated by models of requirements representations (*Hypertext, SA, ER*), an example of a dependency type (*depends-on-onformal-obj*), and an example of a tool (*ER Editor*).

The trace object *Staff* is both an SA data store and an ER entity and thus instance of two classes. Moreover, a specific action of the ER tool has derived it from an informal statement. If a specific guidance chunk had been used, this would be similarly recorded.

**Representation Dimension**

As shown in figure 7, there exist three kinds of representations used during requirements engineering, and other design processes. At the beginning of the process the knowledge is mostly represented using informal representations like natural language, graphics etc. Semi-formal representations come along with graphical notations which provide an easy-to-understand abstract view on the design definitions.

Examples for this kind of representations are ER or data-flow diagrams. Formal representations offer a richer formal semantics, and thus enable automated reasoning.

To make the requirements engineering process traceable, all three kinds of information must be recorded and formally integrated through meta models. For example, some information about an object may be formally defined whereas other parts of the same object may be defined informally.

In NATURE, we have integrated a model of hypertext for structuring and recording the informal information, models of ER and SA for semi-formal information, and Telos itself for representing constraints and integrity rules about the specification.

**Specification Dimension**

Models in the specification dimension are intended to capture how far we got in understanding and writing down the requirements. In practice, this is often determined by standards and guidelines, such as (IEEE-830, 1984; BS-6719, 1984; DOD-2716A, 1988) but there are many of these and they are so informal that it is hard to compare them.

We have therefore condensed over 25 international standards and guidelines into a comprehensive specification model expressed in Telos. It comprises over 30 different kinds of requirements with about 100 attributes organized in a generalization hierarchy. Relationships between the requirements are also represented using attributes.

Through multiple instantiation, each kind of representation can be used for each kind of requirement in the specification model. A specification can therefore be composed of informal, semi-formal and formal components. For example, a requirement for the system might be represented within a hypertext node in natural language. Since our hypertext model is formalized, a formal object represents the hypertext node, and therefore the informal requirement. The formal objects can be related to our formal specification model depending on its content.

**Agreement Dimension**

Progress in the agreement dimension can be recorded by tracing the argumentation within the design team. Usually, many such arguments are exchanged, and sometimes the team decides to take a particular view on these arguments, accepting some and rejecting others, thus chosing and justifying a position taken.

In terms of our basic process model, this means that in the traceability model, a decision is seen as a view on the argumentation space. Note that this is quite different from the guidance model where the arguments are more an auxiliary construct. Of course, we shall construct our method improvement environment in a way that selected arguments in the traceability model are abstracted into arguments of the guidance model.

Formally speaking, our traceability model along the agreement dimension (cf. figure 9) is a variant of the IBIS model similar to the REMAP model (Ramesh and Dhar 1992). The argumentation space is covered by the left side. *T-Arguments* are related to trace positions either as *pro* or as *contra* position, depending on their role. An

argument can of course have more than one role; in other words, an argument can be related to more than one position. The *t-positions* themselves are related to the *t-issue* they deal with. Again, each position can be related to more than one issue. The arguments, positions, and issues together with their relationships are called *argumentation space*.

A *t-decision* is a view of the development team, at a certain time, on this argumentation space. A decision *solves* a particular issue. It is recorded which position was *choosen* by the decision to solve the issue. Other positions, which exist within the argumentation space, may be considered by a decisions either as *supporting* or *declining* positions. But not all positions related to the issue (by the *responding_position* link) must be considered by a decision solving the issue. A decision could only consider a subset of them.

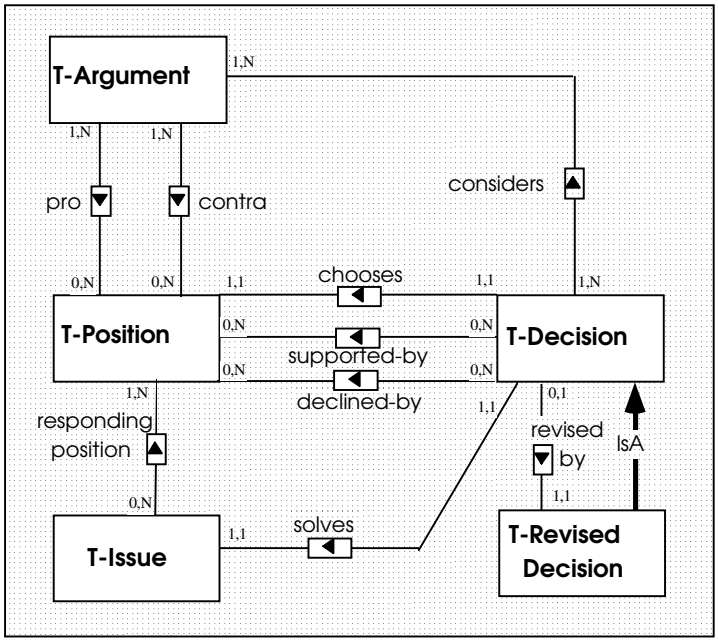


Figure 9: Traceability model along the agreement dimension

Moreover, a decision can only *consider* a subset of the arguments which are related to a particular position (by pro or contra links). For example, if for a particular position a new argument is known, it can be related to the position, without changing the view recorded decisions have on the argumentation space.

Over time, a decision can be *revised* by the requirements engineering team. To record the revision, we have introduced the object *t-revised decision* as specialization of decision. Therefore, also a revised decision represents a view on the argumentation space. During the revision of a decision, new arguments and even positions may be introduced. This arguments and positions, together with existing ones, are related to the revised decision; without changing the view of the old decision.

Summarizing, we have taken a subset of the IBIS model as basis for representing, what we call argumentation space. Decisions made during the design process are

recorded as views on this argumentation space. We are less interested in recording how new issues are raised by arguments, but focus on the decisions taken.

## 6 Managing Relationships Between Guidance and Trace

The agreement dimension model is a good example of a traceability model that

- captures an important aspect of the process at hand -- agreement is crucial to the success of a requirements engineering process and failure to take it into consideration a major reason for project failure
- is designed for concept mapping with the guidance meta model -- the guidance model concepts of choice criteria, positions, and decision are clearly related to the trace components of argument, position, and decision.

The second point is of interest in this section.

In *guided process execution*, the application engineer would use alternatives defined in the guidance models as possible positions for resolving a given issue. He would use the given choice criteria as arguments to choose among those alternatives, and the final decision would be recorded in the trace.

Of course, he may add further alternatives or arguments to justify a decision possibly quite different from that recommended in the guidance model. Or he may follow the guidance model to the letter but the outcome may nevertheless be unsatisfactory.

In *method evaluation and improvement*, it is these differences that are of most interest. Positions taken by application engineers may be sufficiently interesting to be included as alternatives in guidance models. Arguments may be of sufficient generality to change choice criteria. Decisions may be sufficiently convincing to change the decision recommendations in a guidance chunk, or even the structure of guidance chunks.

In the following, we shall illustrate these two ways of concept mapping with a longer example and briefly discuss the necessary tool support.

### 6.1 Deriving Traces from Guidance

The cost of traceability with a rather sophisticated model can be high and is a major impediment to acceptance of our approach. Studies indicate that, without a suitable tool environment, recording the trace is often impossible (Ramesh and Edwards 1993).

Adding to complexity is the fact, that the recorded information must be related to the current process definition (the guidance chunks in use). It must be assured, that the design team uses the actual method definition during process execution. Only in this case, the execution can be recorded as an instantiation of the defined way-of-working.

Therefore, the tool enviroment does not only enable the capture the process trace but must also assure that the guidance model is actually used during process execution.

In the NATURE project we have developed a framework for tool environments which enables the capture of process execution as well as process guidance (Jarke et al. 1994). It is based on two ideas:

- The tool records the execution of an action using the trace model: its input and output, and the dependencies between them. To enable this, we need a *tool model* which defines (1) the actions a tool provides, (2) their inputs and outputs, and (3) the dependencies created by executing the action. Figure 7 is a simplified example.
- Since actions are often nested, the tools must interoperate in a multi-step control. A *communication server* controls and guides the necessary message exchange. In contrast to existing approaches, such as Hewlett Packard´s SoftBench, inter–operation is possible at the granularity of individual actions rather than just for tools as a whole.

Based on this framework a first prototype was implemented on SUN workstations using C++ (Pohl et al. 1994). To illustrate, consider again the specification of a library system using our tool environment and the guidance and trace models.

During the requirements engineering process an entity type *publication* was defined. The informal requirements specification states that some publications can be loaned and returned (e.g., books), whereas others cannot (e.g., journals). The requirements engineer wants to integrate this informal requirement in the ER model. The requirement indicates that there are different variants of the entity-type *publication*. Moreover, they are treated differently by the system : only loanable publication can be checked out and returned.

Using the defined way-of-working as guidance, the requirements engineer identifies the situation captured by the following process chunk (cf. section 4.4):

situation : an entity-type; variants treated different within the system
decision : partition the entity-type into subtypes
choice criteria: PROs : better representation of the real world
CONs : greater complexity of the schema

and decides to partition the entity-type *publication* into *loanable publication* and *non-loanable publication*; the environment records this activity according to the trace models.

In a requirements engineering meeting, the student representative complains that books and proceedings are frequently on loan. To be able to get a proceedings volume, the users have to return many times to the library, only to determine if the book had been returned. Often, the book was in fact returned but given to someone else in between.

This complaint leads to the definition of a new informal requirement (*NR*), stating that a function for making reservations on loaned books should be provided by the final system. Information about the maximum loan duration has to be recorded so that students can predict when to expect a reserved book.

The new requirement is integrated by the requirements engineer, again using the tool environment. Following the process chunk (cf. section 4.4.)

situation : an entity-type; slightly different variants
decision : define discriminant attributes for the entity-type
choice criteria : PROs : leads to a fewer number of types in the schema
CONs : the real-world entities are to be differently treated

he decides to add two new attributes, *loan-duration* and *reservation-name* to the entity type *loanable publication.* The informal requirement (*NR*) is mapped to the new attributes and the decision is recorded as defined in the way-of-working.

For the moment, all seems fine. However, during the implementation of the reservation function, the team detects some difficulties. "Null" values must be introduced for each publication currently not on loan, or for which no reservation exists.

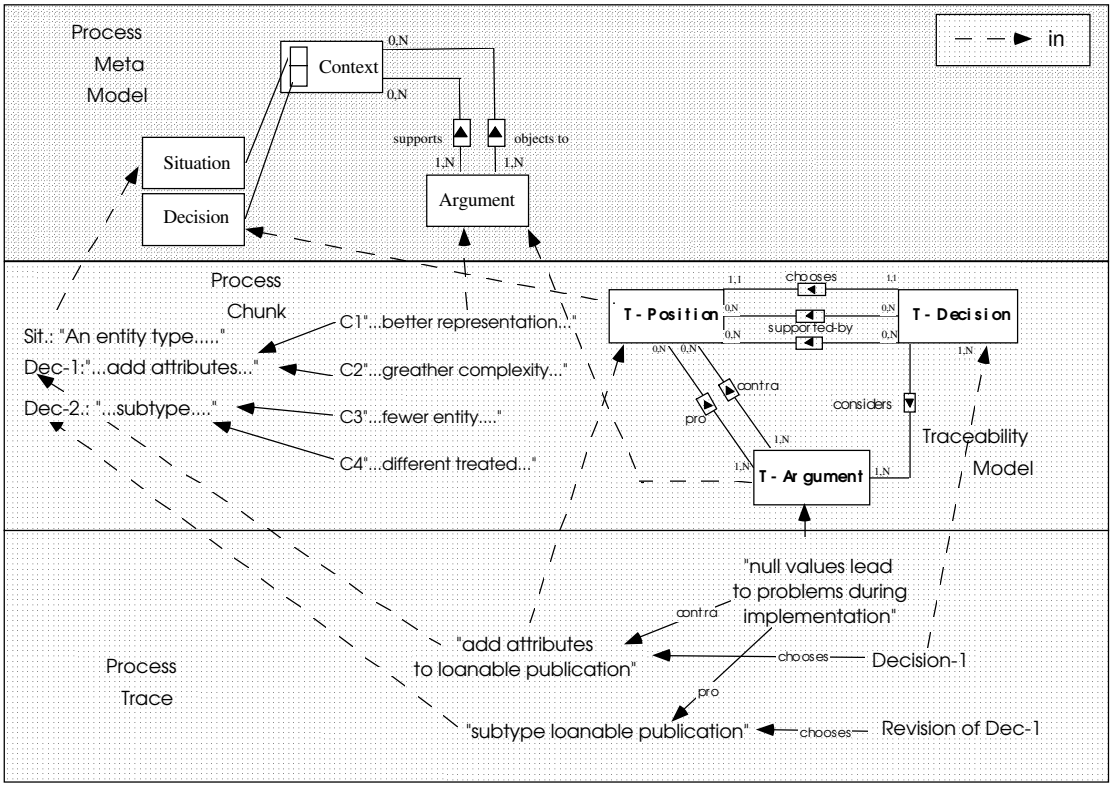


Figure 10: A simplified trace of the argumentation during the revison of the decision

To avoid this problem, the specification is changed. Using the process trace, the team is able to retrieve the situation which led to the introduction of the attributes *loan duration* and *reservation-name*.

Looking at the trace, they recognize that the problem with "null" values is not captured by the process trace. They agree, that the new argument "*null values lead to problems during implementation*" is important enough to change the specification.

Now the team decides to integrate the requirement *(NR)* by subtyping the entity-type *loanable publication* instead of adding new attributes. The old decision is therefore revised, by adding a "CON" argument to the choice criteria for the decision "add attribute" and symmetrically, a "PRO" argument to the choice criteria for decision "subtype entity". The simplied trace of this revision is depicted in figure 10.

Then, the team subtypes the entity-type *loanable publication* into *available-loanable publication* and *loaned publications*. Thus, the actual way-of-working is

different from the defined one. In other words, the trace in figure 10 records a decision within a given situation, which was not defined by the guidance model.

## 6.2 Deriving Guidance from Traces

Improving the current way-of-working based on experience captured by the trace should not be ad-hoc. Using our process meta model, improvement activities can be modeled like normal process chunks using the five basic concepts of situation, decision, context, action and argument (cf. figure 5). Improvement process chunks are modeled at the same IRDS level as the process guidance chunks. The products changed by improvement actions are the defined process guidance chunks.

Jarke & Pohl (1993) have identified three different kinds of improvement situations. First, *lack of quality* of an (intermediate) product recognized by a control action may lead to process improvement, such that quality is produced in the first place. Second, where a *violation of the defined way-of-working* was recognized, the unexpected performance of the process may motivate improvement. Third, *changes in the environment*, for example a policy change, must be integrated in the defined way-of-working.

The idea of concept mapping between guidance and traceability models allows us to make the second kind of improvement more precise. If process execution differs from the definition, for instance:

(1) an action not covered by the process model was preformed,

(2) a new argument was introduced during a decision,

(3) a decision not covered by the process model was made,

the corresponding process improvement chunk is instantiated by the tool environment. For example, if an additional argument is added to a decision, the action by which the argument is recorded creates an instance of the improvement chunk *"new arguments made"*. This situation is related to the corresponding trace chunks as well as to the guidance chunk, in which the argument was added.

We illustrate violation-based process improvement by continuing our example of section 6.1. The method engineering team looks at the recorded improvement situations. Since the improvement situation are related to the process trace as well as to the guidance model, the team is able to retrieve the parts of the process traces which differ from the defined ones. Hence, the decision described at the end of section 6.1, together with the recorded arguments, is retrieved as a process exception of the defined decision *"add attributes"* which is related to the situation *"entity type; different variants"*.

The method engineers look at all instantiations of the improvement chunk *"new arguments made"*. Using the guidance chunk *"add attributes"* as a selection constraint, they select all instances of the improvement chunk "new arguments made" which are related to the guidance chunk *"add attributes"*.

They find two such improvement situations : the one explained above and a second one recorded during the specification of a car rental system. In that system, a

partitioning between cars *on loan* and *available* car was made, mostly because cars on loan need information about the renter, whereas available cars need information where the car is currently parked, so that new renters can find it.

Although different arguments have been used in both cases, the same action has been taken. Looking for common parts in both situations, the team detects that in both cases, a relationship was assosiated with the entity through *0:n* roles. Generalizing, the method engineering team decides, that for each *0:n* realationship, the guidance chunk should suggest a refinement of the entity.

The improvement leads to the modification of the macro-chunk described in section 4.4. First, the situation part of its macro-context is enlarged to contain not only the initial entity-type but also the relationship, the cardinality "0:n" and the related entity-type. Second, the choice criterion supporting the partition alternative is augmented to address the null-value problem. Symmetrically, the choice criterion objecting to the attribute definition alternative is modified accordingly.

The macro-chunk then becomes (changes wrt. section 4.4 are shown in *italics*):

situation : two entity-types; *relationship with a "0:n" cardinality on one side*
decision : model variants of the entity-type *next to the "0:n" cardinality*
1st alternative context :
  situation : the entity-type *next to the "0:n" cardinality*
  decision : define discriminant attributes for this entity-type
  choice criteria: PROs : leads to a fewer number of types in the schema
  CONs : -the real-world entities are to be differently treated
  *-can lead to NULL values problem at implementation*
2nd alternative context :
  situation : the whole situation
  decision : partition the entity-type *next to the "0:n" cardinality*
  choice criteria: PROs : - better representation of the real world
  *- avoids NULL value problems*
  CONs : greater complexity of the schema

## 7 Implications and Conclusions

The literature has distinguished descriptive process models for process traceability from prescriptive process models for process programming or guidance. In this paper, we have related traceability and guidance with the goal of supporting method-driven process execution as well as experience-based process improvement.

On the modeling side, the main idea has been that of a concept mapping between traceability model instances and guidance model chunks.

To realize this idea, we exploited the abstraction concepts of repository standards, the flexibility of our basic process meta model, and the concept of process-adaptable, interoperable, and trace-generating tools.

There are several method evaluation scenarios in which our approach appears applicable:

- Given a set of experimental methods and CASE tools supporting them, we can *debug* them on the basis of recorded experiences with their use.
- We can *formally* compare different methods by analysing the scope of process contexts they can address, and the range of actions they offer.
- We can compare different methods *in practice* by tracing their usage on common examples; within a given method, the usefulness of tools, learning materials, etc. could be compared as well.
- We can derive *ideas* for new useful tools from the process models learned through traces; for example, the informal-to-formal integration in the NATURE prototype through a novel kind of process-integrated hypertext editor was invented as a consequence of our models.

In short, our approach sets up an infrastructure for method evaluation and improvement. In the NATURE project which is concerned with Novel Approaches to Theories Underlying Requirements Engineering, the implementation of this approach is intended as a platform for evaluating and possibly improving many different kinds of requirements engineering techniques.

So far, prototypes of the main infrastructure mechanisms (repository models, communication server, guidance and traceability interfaces along the three dimensions of RE) and of several prototypical process-integrated tools (ER, SA, hypertext, in the near future also OOA) have been implemented with a total of more than 100.000 lines of code.

While they have shown the basic feasibility of the approach, a re-implementation is currently being prepared based on lessons learned from this work. It will be designed to include several enhancements of the models presented here, especially strengthening the notions of goal-oriented, distributed and reuse-based development.

## References


[Ben-Arieh 1988] Ben-Arieh, D.; Product and Process Design in Intelligent CAD Workstations ; in Intelligent CAD Systems II, Springer Verlag, 1988, pp 209-216

[Brunet 1991] Brunet, J. ; Modeling the World with Semantic Objects ; Proc. *IFIP WG 8.1 Working Conf. Object-Oriented Approach in Information Systems*, Quebec, Canada, 1991, pp 361-379.

[BS-6719, 1984] British Standard Institute BS 6719: *British Standard Guide to Specifying User Requirements for a Computer Based System*, 1984. In: [Dorfman and Thayer 1990]

[Curtis 1992] Curtis, B.: The CASE for processes. *Proc. IFIP 8.2 Working Conf. Impact of Computer Supported Techniques on Information Systems Development*. Minneapolis 1992, 333-343.

[Deiters et al. 1989] Deiters, W.; Gruhn, V.; Schaefer, W.: Systematic Development of Formal Software Process Models; *Proc. 2nd European Software Engineering Conf.)*, Springer 1989, pp 100-117

[Desclaux et al. 1990] Desclaux, C.; Guthauser, A.; Picoud, T.; Pomian, J.; Storr, P.; Thierry, A.: Design and Maintenance Process Rationale under MACS; *Conf. Le Genie Logiciel et ses Applications*, Toulouse, France, 1990, pp. 81-95.

[DOD-2716A, 1988] US Departement of Defense. *Military Standard: Defense System Software Development*. DOD-STD-2716A, 1988. In. [Dorfman and Thayer 1990].

[Dorfman and Thayer 1990] Dorfman, M.; Thayer, R.H., eds.: *Standards, Guidelines, and Examples on System and Software Requirements Engineering*. IEEE Computer Society Press, Tutorial, 1990.

[Dowson 1987] Dowson, M.: Iteration in the Software Process; *Proc. 9th Int. Conf. Software Engineering*, San Francisco, Ca, 1987, pp 36-39.

[Feiler and Humphrey 1993] Feiler, P.H.; Humphrey, W.S.: Software Process Development and Enactment: Concepts and Definitions. *Proc. 2nd Conf. Software Process* (Berlin, Germany), IEEE Computer Society, 1993.

[Finkelstein et al. 1990] : Finkelstein, A.C.W.; Kramer, J.; Goedicke, M.: Viewpoint-Oriented Software Development; *Conf. Le Genie Logiciel et ses Applications*, Toulouse, France, 1990, pp 337-351.

[Gotel and Finkelstein 1993] Gotel, O.; Finkelstein, A.C.W.: An Analysis of the Requirements Traceability Problem, Technical Report TR-93-41, Dept Computing, Imperial College, London 1993.

[Grosz 1990] Grosz, G.: Using Artificial Intelligence Techniques to Formalize the Information System Design Process; *Proc Intl. Conf Databases and Expert SystemsApplications*, Vienna 1990.

[Humphrey, 1990] Humphrey, W. S.: *Managing the Software Process*. Addison-Wesley, 1990.

[Humphrey and Kellner 1989] : Humphrey, W.S.; Kellner, M.I.: Software Process Modeling: Principles of Entity Process Models; *Proc. 11th Int. Conf. Software Engineering*, 1989, pp 331-342

[IEEE-830, 1984] IEEE-830 1984: Guide to Software Requirements Specification. *IEEE Software Engineering Standards Collection*, IEEE, New York, 1991

[ISO/IEC 1990] ISO-IEC 10027; *Information Technology - Information Resource Dictionary System (IRDS) - Framework.* ISO/IEC International Standard, 1990.

[Jarke and Pohl 1993] Jarke, M.; Pohl K.: Vision Driven System Engineering. *Proc. IFIP 8.1 Working Conf. Information System Development Process* (Prakash N., Rolland C. and Pernici B., eds.), Como, Italy, 1993, pp. 3-22.

[Jarke et al. 1990] Jarke, M.; Jeusfeld, M.; Rose, T.: A Software Process Data Model for Knowledge Engineering in Information Systems; *Information Systems* 15(1), 1990, pp. 85-116.

[Jarke et al. 1993] Jarke, M.; Pohl, K.; Jacobs, S.; Bubenko, J.; Assenova, P.; Holm, P.; Wangler, B., Rolland, C.; Plihon, V.; Schmitt, J.R.; Sutcliffe, A.G.; Jones, S.; Maiden, N.A.M.; Till, D., Vassilou, Y.; Constantopoulos, P.; and Spandoudakis, G.: Requirements Engineering — An Integrated View of Representation, Process, and Domain. *Proc. 4th European Software Engineering Conf.* (Garmisch-Partenkirchen, Germany), Springer 1993.

[Jarke et al., 1994] Jarke, M.; Nissen, H.; Pohl, K.: Tool Integration in Evolving Information Systems Environments. *Proc. 3rd GI Workshop Information Systems and Artificial Intelligence: Administration and Processing of Complex Structures* (Hamburg, Germany), Springer 1994.

[Jeusfeld, 1992] Jeusfeld, M.: *Änderungskontrolle in Deduktiven Objektbanken*; DISKI Volume 17, Bad Honnef, Germany: INFIX Publ. (Diss. Univ. Passau, in German)

[Kunz and Rittel 1970] Kunz, W.; Rittel, H.: Issues as Elements of Information Systems. Working paper No. 131, Institute of Urban and Regional Development, Univ. of California, Berkeley 1970;

[Lee, 1991] Lee, J.: Extending the Potts and Bruns Model for Recording Design Rationale; *Proc. 13th Intl. Conf. Software Engineering* (Austin, Tx), pp 114-125 IEEE, 1991

[Maiden and Sutcliffe 1992] Maiden, N.A.M.; Sutcliffe, A.G.: Exploiting Reuseable Specifications through Analogy. Communications of the ACM 35(4), 1992, pp. 55-64.

[Mylopoulos et al., 1990] Mylopoulos, J.; Borgida, A.; Jarke, M.; Koubarakis, M.: Telos: Representing Knowledge About Information Systems; *ACM Trans. Inform. Systems* 8(4), 1990, pp. 325-362.

[Oivo and Basili, 1992] Oivo, M; Basili, V. R.: "Representing Software Engineering Models: The TAME Goal Oriented Approach" *IEEE Trans. Software Engineering,* 18(10), 1992.

[Osterweil 1987] Osterweil, L.: Software Processes are Software too; *Proc. 9th Intl. Conf. Software Engineering*, San Francisco, Ca, 1987, pp 2-13

[Peugeot and Franckson 1991] Peugeot, C.; Franckson, M.: Specification of the Object and Process Modelling Language, ESF Report n° D122-OPML-1.0, 1991.

[Pohl 1994] Pohl, K.: The Three Dimensions of Requirements Engineering: A Framework and its Applications, *Information Systems*, 19(2), 1994.

[Pohl et al. 1994] Pohl,K.; Assenova, P.; Doemges, R.; Johannesson, P.; Maiden, N.; Plihon, V.; Schmitt, J.-R.; Spanoudakis, G.: Applying AI Techniques to Requirements Engineering: The NATURE Prototype; Submitted to *ICSE-16 Workshop on Artificial Intelligence and Software Engineering*, Sorrento, Italy, 1994

[Potts 1989] Potts, C.: A Generic Model for Representing Design Methods; *Proc. 11th Intl. Conf. Software Engineering*, 1989, pp 217-226.

[Ramesh and Dhar, 1992] Ramesh, B.; Dhar, V.: "Process-Knowledge Based Group Support in Requirements Engineering", *IEEE Trans. Software Engineering,* 18(6), 1992.

[Ramesh and Edwards 1993] Ramesh, B.; Edwards, M.: Issues in the Development of a Requirements Traceability Model. *Proc. IEEE Symp. Requirements Engineering*, San Diego, Ca, 1993.

[Rolland and Cauvet 1991] Rolland ,C.; Cauvet, C.: ALECSI - An Expert System for Requirements Engineering, *Proc. 3rd Intl. Conf. Advanced Information Systems Engineering* (Trondheim, Norway), Springer 1991.

[Rolland and Prakash 1993] Rolland, C.; Prakash, N.: Reusable Process Chunks. *Proc. Intl. Conf. Database and Expert Systems Applications,* Prague 1993.

[Sagawa 1990] Sagawa, J.W.: "Repository Manager Technology", *IBM Systems Journal* 29(2), 1990, pp. 209-227.

[Spandoudakis and Constantopoulos] Spandoudakis, G.; Constantopolous, P.: Similarity for Analogical Software Reuse: A Conceptual Modelling Approach. *Proc. 5th Intl. Conf. Advanced Information Systems Engineering* (Paris, France), Springer 1993, pp. 483-503.

[Williams 1988] Williams, L.G.: Software Process Modeling: A Behavioral Approach; *Proc. 10th Intl. Conf. Software Engineering,* IEEE Computer Society 1988, pp 174-186

[Wynekoop and Russo 1993] Wynekoop, J; Russo, N.: System Development Methodologies: Unanswered Questions and the Research-Practise Gap. *Proc. 14th Intl. Conf. Information Systems,* Orlando, Fl, 1993, pp. 181-190.