PRO-ART*:

Enabling Requirements Pre-Traceability[1]

Klaus Pohl

Informatik V, RWTH Aachen, Ahornstr. 55, 52056 Aachen, Germany

email: pohl@informatik.rwth-aachen.de

Abstract: *Requirements traceability is essential for developing software systems of high quality. Whereas the traceability of the refinement, deployment, and use of a requirement is called post-traceability, the traceability of a requirement back to its origin is named pre-traceability.*
In this contribution we present a requirements engineering environment, called PRO-ART, which enables requirements pre-traceability. PRO-ART is based on three main contributions:*

- *a three-dimensional framework for requirements engineering which defines the kind of information to be recorded;*
- *a trace-repository for structuring the trace information and enabling selective trace retrieval;*
- *a novel tool interoperability approach which enables (almost) automated trace capture.*

In addition, we report on experiences made with the first prototypical implementation of PRO-ART and the resulting re-design and re-implementation, called PRO-ART 2.0, which mainly addresses scalability problems faced with in real applications.

[1] A version of this paper appears in the *Proceedings of the IEEE Intl. Conference on Requirements Engineering (ICRE'96),* Colorado, April 15–28th, 1996.

* Process and RepOsitory based Approach for Requirements Traceability

1. Introduction

Requirement traceability is a prerequisite for building high quality software systems. Capturing and maintaining traces is thus seen as an essential activity to be performed during requirements engineering (e.g. [DoD-2167A, 1988; Gotel and Finkelstein, 1994a; IEEE-830, 1984; Pohl, 1994; Pohl *et al.*, 1994b; Ramesh, 1993; Wright, 1991]). A comprehensive overview of possible usages of trace information and the expected benefits can be found in, e.g., [Gotel and Finkelstein, 1994a; Pohl, 1996a; Ramesh, 1993]. Among others, these reports indicate that traceable specifications are essential for consistent change integration, lead to less errors during system development, play an important role in contract situations, and improve the acceptance of the (software) system.

On the one hand, traceability from the requirements specification down to design and implementation and vice versa is needed to better understand and therefore accept the current system. On the other hand, the development process leading to the requirements specification must be traceable to enable the understanding of the requirements themselves, i.e. to be able to trace a requirement back to its origin. Thus, a differentiation between the two kinds of traceability is commonly made (cf. [Davis, 1990; Gotel and Finkelstein, 1994a; IEEE-830, 1984; Pohl, 1996a]).

Whereas the traceability of the refinement, deployment, and use of a requirement is called *post-traceability* (or forward traceability), the traceability of a requirement back to its origin is named *pre-traceability* (or backward traceability). By requirements pre-traceability an essential foundation for managing the system evolution is established, since it enables the elicitation of the parts of the specification which are effected by a particular change request raised by the environment, e.g. a change in organizational policy, the business process, or an extended usage of the system.

There exist some approaches [Alford, 1990; Flynn and Dorfmann, 1990] and even commercial tools (e.g. *RT* from Teledyne Brown Engineering, *RTM* from Marconi Systems Technology, *RDD 100* from Ascent Logic Cooperation) which support traceability between requirements (focusing on the trace of hierarchical decompositions) and between the requirements specification and documents produced during

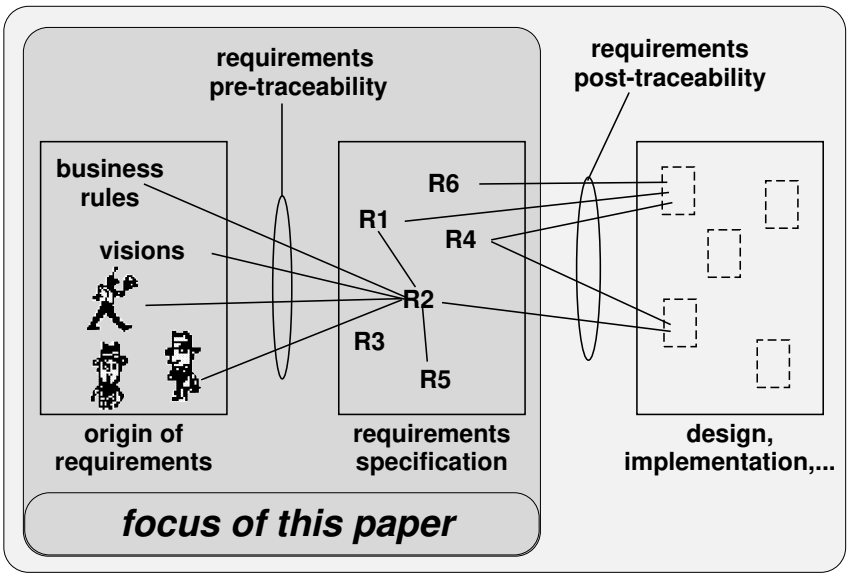

Fig. 1.1 Requirements pre- and post-traceability.

later development stages, e.g. design documents (see [Gotel and Finkelstein, 1993] for a survey on existing approaches). They support the requirements engineer in keeping track of requirements and ensuring that all requirements are properly flowed down to all levels with no requirement lost and none added. Unfortunately, these approaches tend to neglect requirements pre-traceability, i.e. the elicitation of the requirements, their definition and their evolution.

This is in contrast to the results gained by the independent surveys of Ramesh & Edwards [Ramesh and Edwards, 1993] and Gotel & Finkelstein [Gotel and Finkelstein, 1993] which indicate that requirements pre-traceability is at least as important as requirements post-traceability; especially for systems which are embedded in a continuously changing environment. Recent research contributions (e.g. [Gotel and Finkelstein, 1994b; Kaindl, 1993; Ramesh, 1993; Ramesh and Dhar, 1992; Ramesh and Edwards, 1993; Ramesh *et al.*, 1995]) and systems (e.g. *DOORS* from QSS) therefore also focus on requirements pre-traceability. All these approaches consider different aspects of requirements pre-traceability, e.g. the annotation of requirements specifications using hypertext or the capturing of contribution structures, and therefore the knowledge being captured and the support offered are almost disjunct. Thus, no comprehensive framework for requirements pre-traceability has been established so far (cf. [Gotel and Finkelstein, 1994a; Ramesh and Edwards, 1993; Ramesh *et al.*, 1995]).

Towards a framework for requirements pre-traceability we organize our contribution along three questions to which each approach to requirement pre-traceability has to provide explicit (or implicit) answers:

- *What kind of information has to be recorded ?* (cf. Section 2)
- *How to structure the information ?* (cf. Section 3)
- *How to capture the information ?* (cf. Section 4)

We address these questions on two different levels. On the one hand, a generic framework is provided for each question which serves as a guideline for developing a concrete approach for requirements pre-traceability. On the other hand, we present concrete answers on which the PRO-ART environment is based. Experiences made and open research questions are summarized in Section 5.

2. What Kind of Information must be Recorded ?

Requirements pre-traceability is established if the requirements engineering process itself is traceable. Thus, an answer to the question *"what kind of information must be recorded?"* for enabling requirements pre-traceability must be based on a framework for requirements engineering processes. Based on a comprehensive survey of the current state of art and practice we have proposed a three-dimensional framework for requirements engineering [Pohl, 1994] which is outlined in Section 2.1. Using this framework we define the basic types of information to be recorded and briefly characterize existing approaches according to this definition in Section 2.2.

2.1. A Three-Dimensional Framework for Requirements Engineering

We have defined requirements engineering as a process of *establishing an overall system vision in context* [Jarke and Pohl, 1993; Pohl, 1996a; Pohl, 1996b]. The process of developing a requirements specification from the system vision can be described by three almost orthogonal dimensions (cf. figure 2.1)[1], namely

[1] A detailed description of the three dimensions, their orthogonality, consequences for supporting the requirements engineering process, possible use, etc. can

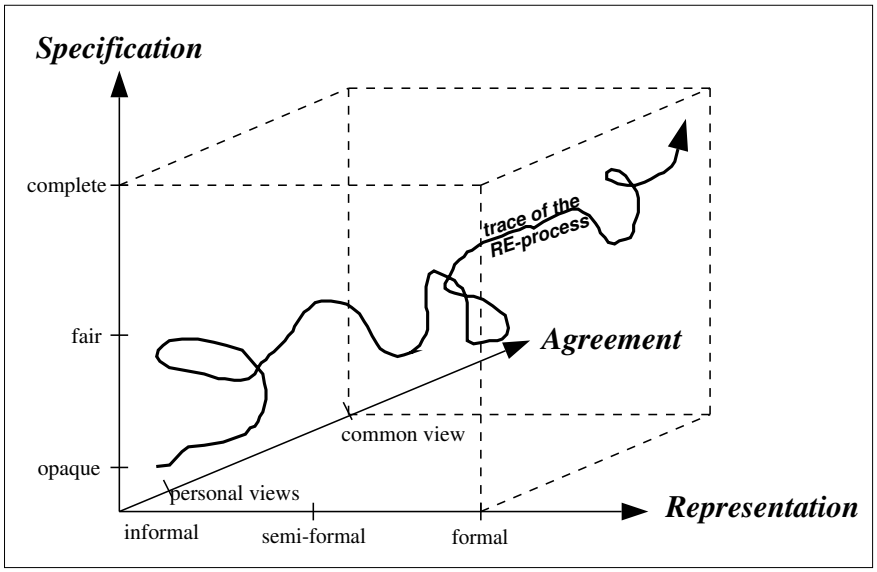

Fig. 2.1 The three dimensions of requirements engineering

- *representation dimension:* 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 into executable code. Since each representation formalism offers unique advantages, people prefer different representations in different situations. Modelling and relating these representations is mostly a technical problem albeit one whose solution requires good knowledge of human-computer interaction;
- *specification dimension:* a second obvious goal is to end up with a complete system specification, i.e. with a sufficient system understanding. There are two kinds of completeness. The first one deals with the coverage of the problem in the specification, i.e. it states wether all relevant requirements have been captured by the final specification. The second defines wether each known requirement has been well defined, i.e. wether all requirements have been defined according to the standard chosen for specifying the system. We see this orthogonal to the first dimension: 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. Moreover, the same knowledge is often described in different representations, e.g. textual and graphical. Moves along this axis mostly face cognitive and psychological problems of requirements engineering;
- *agreement dimension:* the third axis is concerned with the agreement reached on the current specification. We see requirements engineering as a negotiation and learning process which should establish sufficient agreement before the system is being built. Thus, starting with almost disjoint viewpoints of the different stakeholders a common understanding of the system must be developed during the requirements engineering process. Hence, this axis mainly deals with the social aspects of requirements engineering.

At the level of process execution, RE can be understood as a movement in the three-dimensional space which, in addition to technical, cognitive, and social barriers, is also influenced by economical factors, tool support offered, and the methods used [Pohl, 1994]. For example, the traditional RE process starts near the

be found in [Pohl, 1994]. In [Pohl, 1996b] the three dimensions are related to the four worlds of information systems modelling introduced in [Jarke and Pohl, 1993].

origin with different personal views, little system understanding and completely informal representations; it should end up with sufficient agreement on a well-understood and (mostly) formally defined specification. The trace of the process can be seen as an arbitrary curve within the cube spanned by the three dimensions (cf. figure 2.1).

2.2. Important Kinds of Trace Information

According to figure 2.1 capturing the process traces and thereby establishing requirements pre-traceability consequently means recording

- information along the representation dimension, i.e. the different representation formats used during the process and their relations;
- information along the specification dimension, i.e. the content of the specification independent of its representation according to some guidelines, standards, or domain models;
- information along the agreement dimension, i.e. the different viewpoints of the stakeholders, arguments, alternative solutions, and decisions made during the process;
- *relations* between these kind of informations, i.e. dependencies which exist between the recorded trace objects, e.g. which particular decision caused the creation of a new requirement;
- the *actual process performance*, i.e. relating the information to the process steps and the agents (humans/tools) by which it was produced.

Looking at existing approaches and systems using the three-dimensional framework it becomes obvious that each of these approaches neglects to record information of one or even two dimensions. For example, Ramesh and Dhar [Ramesh and Dhar, 1992] focused mainly on the agreement dimension and totally neglected the representation dimension. Similarly, the approach of Ramesh et al. [Ramesh, 1993; Ramesh *et al.*, 1995] considers the specification and agreement dimension, but does not consider the representation dimension. The approach proposed by Kaindl [Kaindl, 1993] concentrates on the representation dimension, but almost ignores the specification and agreement dimensions. Other approaches focus on a particular aspect of requirements pre-traceability, e.g. on contribution structures [Gotel and Finkelstein, 1994b], and therefore provide no comprehensive framework.

3. How to Structure the Information ?

Three main problems occur when the recorded traces are being used during the system development process. First, as identified by many contributions (e.g. [Ramesh *et al.*, 1995]) the usage of trace information depends on the type of stakeholder and the phase of the software development process, i.e. traces are not used as they were recorded and therefore selective retrieval according to the actual needs must be supported. Second, due to the large amount of information produced during process performance, only content oriented trace capture provides the basis for appropriate reuse [Pohl, 1996a], i.e. the reuse of the recorded information is almost impossible if the trace objects are not embedded in a broader context. Third, the people involved in the trace capture are most often different from the users of the trace information [Gotel and Finkelstein, 1993].

To deal with the above problems, some kind of structuring mechanism is needed which enables the systematic person-independent recording of traces and the selective situation-dependent retrieval of the

trace information.

As a framework for structuring the trace information we propose to adopt the IRDS-Standard [ISO/IEC, 1990] briefly explained in Section 3.1.1. Based on IRDS we establish a generic trace structure for a trace repository which is based on the three dimensions of requirements engineering (cf. Section 3.1.2). The concrete traceability structure used by the PRO-ART is outlined in Section 3.2. This structure is based on an elaboration of each dimension into a set of class definitions which characterize points and relationships along these dimensions.

3.1. Establishing a Trace Repository

3.1.1. Information Resource Dictionary Standard (IRDS)

The IRDS-Standard (ISO/IEC 10027 [ISO/IEC, 1990]) was prepared by the ISO/IEC JTC 1 committee and belongs to a series of International Standards on Information Resource Dictionary Systems. The purpose of this series is to provide a common basis for the development of Information Resource Dictionaries. An *information resource dictionary (IRD)* is a shareable repository for the definition of the information resources, relevant to the whole or parts of an enterprise. An *information resource dictionary system* is a system which provides facilities for creating, maintaining, and accessing an IRD and its definition. The IRDS standard does not provide a definition of all the information to be recorded in an IRD, but it does provide a framework for defining, representing, and managing such information.

Since a detailed description of the IRDS–Standard is far beyond the scope of this paper we only sketch the main concepts in the following.

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

At the *IRD Definition Schema Level* the types of objects about which data may be recorded at the IRD Definition Level are prescribed.

At the *IRD Definition Level* schemata are defined using the schema established at the IRD Definition Schema Level. These schemata define the types of objects about which data may be stored at the IRD Level of all Information Resource Dictionaries (IRDs). The schema of a particular IRD may consist of a subset of these types, e.g. all object types (e.g., *entity-type*) defining an Entity-Relationship diagram language.

At the *IRD Level* application schemata are defined. The types of data in an IRD Level are completely defined at the IRD Schema Level. There may be any number of Information Resource Dictionaries, all described using one IRD Schema, e.g., an Entity-Relationship model describing the application schema for a library system and another one describing a petrol filling station.

At the *Application level* instances of business applications are recorded. For example, the people (e.g., Bill Jones) having loaned a particular book (e.g., The Pelican Brief).

Based on these four data levels, the IRDS standard promotes a so-called level pair architecture. A level pair consists of two adjacent data levels. The upper level defines the *types* for the level below; it is called the schema for the lower level. At the lower level information is recorded as *instances* of the types defined at the level above. Hence, a level pair can be interpreted as a database specification and the four levels can be defined by a hierarchy of three database specifications on top of each other.

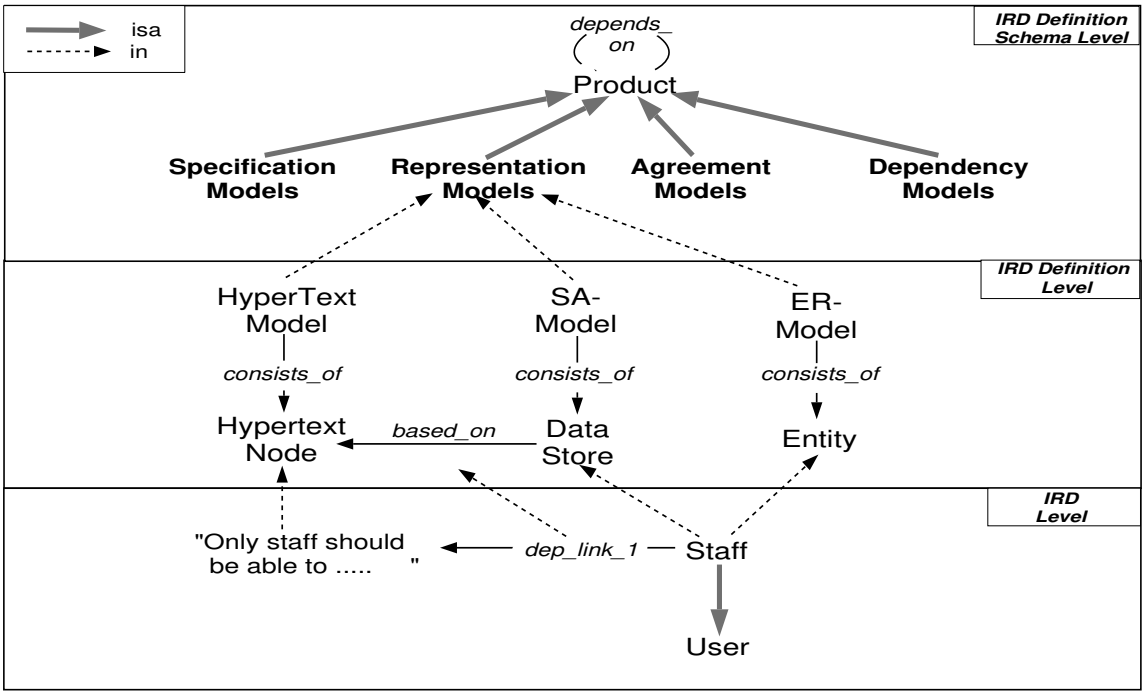

Fig. 3.1 Structuring trace information using meta models.

3.1.2. Structuring Traces based on IRDS

For adopting the IRDS concepts to the problem of structuring the trace information of requirements engineering processes, the three upper IRDS layers must be populated (see [Pohl, 1996a] for details).

IRD Definition Schema Level: This level is populated by the *traceability meta model* which defines the concepts to be used for establishing a concrete traceability structure at the IRD Definition Level. There are two possibilities for defining a concrete traceability structure: One is to define a monolithic model consisting of disjoint concepts; the other is to define a set of orthogonal models along each dimension and to express the relationships between the concepts of these models by appropriate dependency links. Due to the orthogonal nature of the information the first approach leads to an explosion of concepts. The resulting monolithic model is hard to understand and to modify.[2] Having identified that the information can be classified into three orthogonal dimensions (representation, specification, agreement), only the second approach appears suitable.

Therefore, our traceability meta model (simplified to the object `product` in figure 3.1) distinguishes between four kinds of orthogonal traceability models, namely models for capturing information along the representation, specification and agreement dimension and dependency models which define the principle relations between trace objects (a detailed description of the traceability meta model can be found in [Pohl, 1996a]).

IRD Definition Level: At the IRD Definition Level a *concrete traceability structure* is defined as an instantiation of the traceability meta model (cf. Section 3.2 for a description of traceability structure of the PRO-ART environment). A concrete traceability structure should be defined according to the particular

[2] Many research contributions have ignored the orthogonality of the trace information and therefore propose one monolithic model (e.g., [Conklin and Begeman, 1988; Delisle and Schwartz, 1986; Fischer *et al.*, 1989; Garg and Scacchi, 1990; Kaindl, 1993; Ramesh and Dhar, 1992]).

needs of the organization. However, this structure explicitly defines the types of information and the relations to be recorded during the requirements engineering process.[3]

For capturing the knowledge along a single dimension also orthogonal models should be defined. For example, in figure 3.1 three simplified models for representing the information along the representation dimension are shown: a hypertext model, a Structured-Analysis (SA) model and an Entity-Relationship (ER) model.

In addition, possible relations between the trace objects (e.g., the dependency (relation) type *based_on* shown in figure 3.1) have to be defined in a dependency model (not shown in figure 3.1). Such a model should further restrict the usage of the dependency types (e.g., by integrity constraints) as well as enforce their creation (e.g. by deductive rules).

IRD Level: Figure 3.1 furthermore illustrates how a particular trace is recorded at the IRD Level using the schema defined at the IRD Definition Level. There are five trace objects, *Staff*, *User*, *"Only staff should be able to ...."*, *dep_link_1*, and the *isA* link between *Staff* and *User*. The trace object *Staff* represents a *DataStore*, but also an *Entity*, and is therefore recorded as instances of the two classes defined on the level above. The dependency link *dep_link_1* is recorded as an instance of the dependency type *based_on* and relates the object *Staff* and the object *"Only staff should be able to ..."* which is recorded as an instance of the object *HypertextNode*. This short examples shows that the relationship between trace objects can be represented in two ways: using dependency-links and/or *multiple instantiation*.

3.2. The Traceability Structure of the PRO-ART Environment

The traceability structure of the PRO-ART environment, defined at the IRD Definition level, consists of orthogonal models by which information along the agreement, representation, and specification dimension can be captured. In addition, a dependency model is provided which enables a relation of the trace objects. Currently, the structure consists of over 250 class definitions (concepts) and more than 80 integrity constraints and deductive rules by which a consistent instantiation of the traceability structure is ensured. Describing all these definitions in this contribution is impossible. We therefore give an overview of the various models and provide references in which the models are described in detail.

For structuring the knowledge of the *representation* dimension we have focused on quasi industrial standards to demonstrate that our traceability approach also improves well known methods like Entity-Relationship modelling or the OMT technique. Currently we support

- a unique hypertext model [Pohl and Haumer, 1995] for structuring informal representations like natural language descriptions, pictures, tables etc.;
- an extended Entity-Relationship model [Pohl, 1996a] for modelling the data view of the system;
- a Structured Analysis model which consists of a data flow model (DFD) and a Data Dictionary [Pohl, 1996a] for modelling the functional view of the system;
- the object model and behavior model of OMT for modelling the object-oriented view as well as the behavior of the system;
- O-Telos [Jeusfeld, 1992; Mylopoulos *et al.*, 1990] for defining integrity constraints and deductive rules about the system.[4]

[3] An important research task is to establish a classification by which the type of information to be traced can be identified depending on the intended usage. Since traceability always depends on the usage and causes additional costs, some evaluation criteria are needed to define information to be traced during process performance with respect to the expected pay-offs [Pohl *et al.*, 1994b].

[4] Since we have implemented the repository in ConceptBase [Jarke *et al.*, 1994], an implementation of O-Telos, there is no explicit O-Telos definition on the IRD Definition Level.

For structuring the information along the *specification* dimension, we considered developing a specification model based on a well known existing standard, e.g. IEEE-830 [IEEE-830, 1984]. Such a standard on the one hand defines the type of requirements to be captured in a software specification and on the other hand defines how these requirements should be documented. Unfortunately, we recognized that the existing international standards and guidelines differ in the stated demands for software requirements specifications (a collection of such standards can be found in [Dorfman and Thayer, 1990]). Thus we have analyzed 21 existing standards and guidelines and developed a comprehensive specification model, called the RSM (requirements specification model), for recording the information along the specification dimension which comprises these standards (see [Pohl, 1996a]).

Communication, conversation, and negotiation between the people involved in the requirements engineering process lead to decisions about a particular requirement. A conversation for possible solutions (alternatives) comprises a set of arguments. For structuring such argumentations Rittel has proposed the Issue Based Information System (IBIS), on which many approaches for recording the rationale of design processes are based (e.g., [Conklin and Begeman, 1988; Fischer, 1993; Potts and Bruns, 1988; Ramesh and Dhar, 1992]).

Our model for capturing information along the *agreement* dimension takes a subset of the IBIS model (see [Pohl, 1996a]) and focuses on the decisions made during the requirements engineering process and provides concepts for representing the revision of a decision.

The *Dependency Model* is based on a literature survey of contributions on structuring information in the area of requirements engineering, text analysis, knowledge acquisition, recording design rationale, and hypertext. It consists of 18 different dependency links (see [Pohl, 1996a] for a detailed description) which are categorized into five disjunct classes, namely

- *condition links* which are used to relate restrictions to a particular object, e.g. constraints;
- *content* links which are introduced between objects to represent relations between the contents of the objects, e.g. similarities or contradictions;
- *documents* links which are used to relate different kinds of documentation to a trace object, e.g. examples or purpose information;
- *evolutionary* links which are used to express that a particular requirement (source object) was replaced by the target object, e.g. that the definition of an object is *based_on* the existence of another trace object;
- *abstraction* links which are used to represent abstraction between trace requirements, like generalizations or refinements.

Summarizing, we propose to structure the trace information in a three layer architecture according to the IRDS-Standard. Consequently, we have developed a traceability meta model at the IRD Definition Schema Level under which a concrete traceability structure at the IRD Definition Level can be defined. Moreover, we have established a concrete traceability structure by defining orthogonal models for structuring the information along each of the three dimensions and a dependency model which defines, among other things, a taxonomy of dependency types by which the various trace objects can be related. This structure is currently being used within the PRO-ART environment. An actual process trace is recorded at the IRD Level by instantiating the traceability structure. Relations between the trace information can be recorded by multiple instantiation, i.e. a trace object is defined as an instance of more than one class of the traceability structure, and/or by creating appropriate dependency links during process execution.

In comparison to other approaches a trace repository organized in the way described above offers two main advantages: (1) each trace object can be related to any other trace object, e.g. an entity can be related

to a sentence (or even a single word) of an informal document, or/and to a formally defined constraint, as well as to a decision or a single argument used by the decision; (2) the layered architecture and the formal definition provide a rich structure which is a prerequisite for the selective retrieval of the recorded trace information.

4. How to Record Trace Information ?

Our approach for recording the trace information is based on two assumptions.

First, we assume that the information must be recorded during process execution since it is almost impossible to construct the trace after the requirements specification has been established.

Second, trace capture must be, as far as possible, automated (cf. [Pohl *et al.*, 1994b; Ramesh, 1993; Wolf and Rosenblum, 1993]). On the one hand, the manual recording of trace information would cause an additional work-load for the people performing the process and would therefore make traceability costly and labor intensive. On the other hand, manually recorded traces are often subjective and idealized, i.e. traces are adapted to certain expectations, e.g. to conform to predefined plans. Thus, manually recorded traces often do not reflect the reality.

However, fully automated trace capture is impossible due to the large amount of informal information used during requirements engineering, e.g. minutes of meetings or interview notes. Structuring and relating this information can only be done by human experts. Nevertheless, the contribution in this section shows that a suitable requirements engineering environment can be made responsible for the majority of the trace capture job and can help the requirements engineer in capturing correct traces.

We first illustrate the basic approach for recording the traces (Section 4.1). Then we briefly describe the approach for (almost) automated trace capture underlying the PRO-ART environment (Section 4.2).

4.1. Automated Trace Capture: Basic Idea

The task of capturing the trace information and recording it in the predefined traceability structure must be accomplished by the process execution environment, i.e. by the requirements engineering environment (REE) used. Each computer based environment provides a fixed set of actions by which the product is created, deleted, and/or modified. These actions are mostly implemented by certain programs (modules, procedures); i.e. they are defined in a particular programming language.

The basic idea behind our approach to automated trace capture is that the actions provided by the REE are made responsible for recording the trace information. This requires that the REE assures that

- the performance of each action is recorded in the trace repository together with the input used and the output produced by the action;
- if through the performance of an action a dependency is established between two or more trace objects appropriate dependency links are created between these objects;
- the stakeholder performing the action and the embedding of the action in the process context are recorded (e.g. the relation of the action to other actions performed before).

To enable the recording of this information in the trace repository we have to extend the traceability schemata provided at the IRD Definition Level. We have defined such an extension (see [Pohl, 1996a] for details) which, among others, introduces the concepts of *tool* (e.g. *ER_Editor*), *actions* provided by

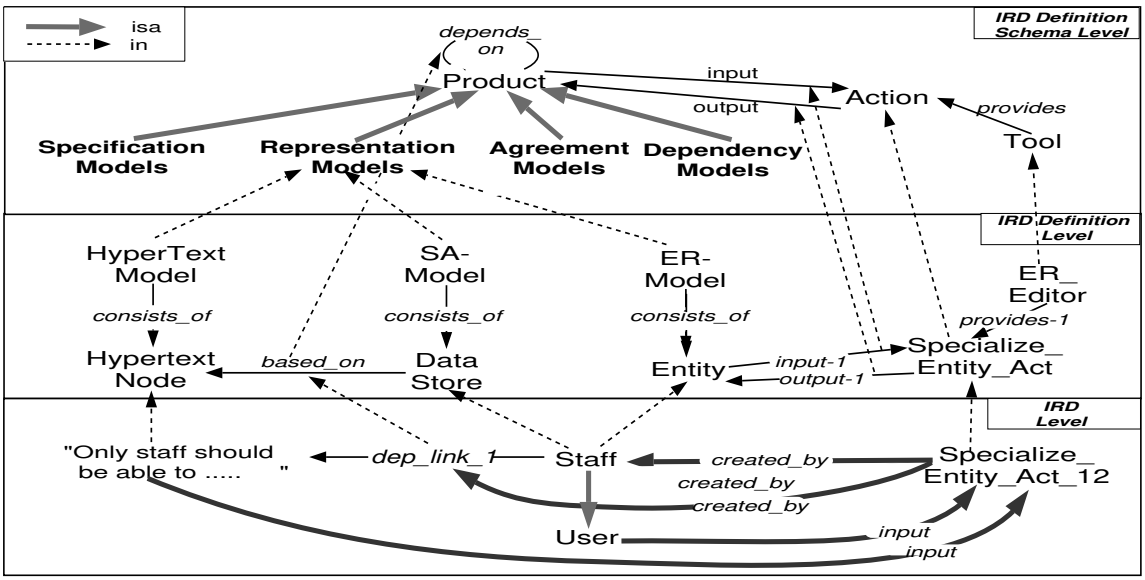

Fig. 4.1 Recording process traces based on tool actions (simplified structure).

the tool, and the possible *inputs* and *outputs* used by the actions (cf. figure 4.1). Thus, the execution of each action can be recorded in the process repository as an instance of the defined action type (cf. figure 4.1) and the actual inputs and outputs can be related to the action.

Taking the example used in Section 3.1.2, we now see that the entity *Staff* was created as a specialization (denoted by the *isa* link) of the entity *User* by the *SpecializeEntity_Act_12* and that the action has used the entity *User* and the hypertext node *"Only staff should be able to ..."* as input. Moreover, two dependencies have been created between the trace objects, namely one of the type *depends_on_informal_obj* between the object *Staff* and *"Only staff should be able to ..."*, labelled *dep_link_1* (cf. figure 4.1); the other between the entity *User* and the entity *Staff* of the same type (not shown in figure 4.1).

Note, that these extensions to the traceability structure of the trace repository provide helpful abstractions for the selective retrieval of trace information, especially if the recorded actions are complex actions. For example, if all products and dependencies which are created during a single review process are related to this complex action the retrieval of all changes made and errors detected during the review is possible. This can simply be done by relating the actions performed during the review to the complex action representing the review task (e.g. by *part_of* dependency links).

4.2. PRO-ART: Trace Capture by Tools and Process Enactment Mechanism

The approach described above works well for recording the execution of a single action. However, dependencies are often caused by the context in which an action is performed or by the performance of a particular action sequence, i.e. dependencies which are caused through the performance of nested actions can not be captured by the approach described above.

For example, a dependency between an informal text (recorded in a hypertext node) and another trace object depends on the context in which the hypertext node is related to this object. If the informal text has caused the creation of an entity (as in the example above) the node should be related to the entity using a *based_on* link. If the informal text was entered as an explanation for the entity it should be related

to the entity using a link of the type *explains*. In both cases the same action is performed for selecting the hypertext node, namely the *select hypertext nodes* action of the hypertext editor. In other words, the output of the *select hypertext node* action is the selected hypertext node (or its identifier). Similarly, the created entity is the only output from the *create entity* action. The problem which occurs is that the dependency should be created between the outputs of two (or more) actions and that none of the actions (e.g. create entity, select hypertext node) can record the dependency between these objects (hypertext node and entity). Thus, an approach is needed which enables the recording of dependencies which are caused by the execution of nested actions.

Such an approach has to provide solutions for two main problems: (1) interoperable tools are needed as a technical prerequisites for relating the inputs/outputs of the actions provided by the different tools; (2) the requirements engineering environment must be process centered since the dependencies which are created between the inputs/outputs of different actions depend on the way the process was performed, i.e. on the sequence of actions.

A solution to these problems is offered by the PRO-ART environment which enables automated trace capture based on a new approach for *situated and process integrated* tools. In the following we explain the underlying ideas of this approach. The technical details are outlined in [Pohl, 1996a].

The area of software processes was divided by Dowson into three domains, namely the *process modelling, process performance*, and *process enactment* domains (cf. [Dowson, 1992]). The *process modelling* domain captures all the activities performed for modelling the software processes. This subsumes the definition of process models using a particular formalism as well as the improvements of the models based on the analysis of the process performance. *Process enactment* encompasses what takes place in a process to support process performance based on the process definitions. This is essentially a mechanical interpretation of an instantiated process model that guides, enforces, or partly automates

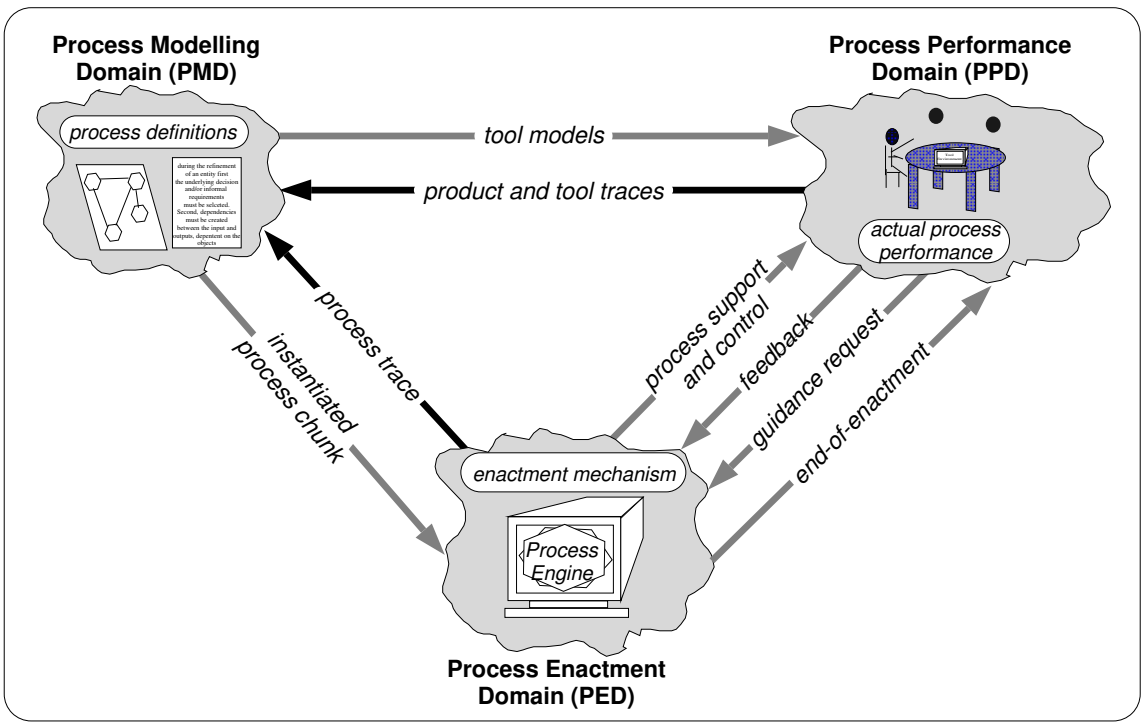

Fig. 4.2 New interaction patterns between the three domains of process support (see [Pohl, 1996a] for details).

process performance. The *process performance* domain is defined as the set of activities conducted by human agents and non-human agents (computers). It is unimportant whether the performance is purely manual or influenced (i.e. guided, enforced, or partially automated) by a process centered engineering environment (cf. figure 4.2).

We have adopted this basic differentiation, but our approach to situated and process integrated tools is based on new interaction patterns between the three domains. In contrast to all other approaches, the tools of the process enactment domain request process support (*guidance request*) from the process enactment domain. Thus, the engineers performing the process have more control over the process and a more flexible way of working is enabled (see [Pohl, 1996a; Pohl *et al.*, 1994a] for details).

In addition to the process adaptability of our tools, the process enactment domain as well as the process performance domain capture the traces during process execution. The performance of single actions is recorded by the tools constituting the requirements engineering environment used in the process performance domain (*product and tool traces*), and the performance of the nested actions is recorded by the process enactment mechanism running in the process enactment domain (*process traces*). To enable this, the tools used in the process performance domain must be explicitly modelled in the process modelling domain (in addition to the process models which define the nested actions). Moreover, the part of the trace structure to be instantiated whenever an action (single or nested) of a certain type is performed is attached to the process and tool models. The relation of the models enables the tools to initiate the performance of nested actions by sending a *guidance request* to the process performance domain and to capture the traces according to the definitions (see [Pohl, 1996a]).

Thus, the traces which are (almost) automatically captured by the PRO-ART environment consist of (a) the products produced, (b) the dependencies between product parts created by single or nested actions, and (c) the human agent and the tool actions by which the trace was captured, plus other annotations (e.g., relevance of the dependency).

5. Conclusions

In this paper we have presented an approach for requirements pre-traceability which is based on three main contributions, namely

- a three-dimensional framework for requirements engineering which defines the kind of information to be recorded (cf. Section 2);
- a trace-repository for structuring the trace information and enabling selective trace retrieval (cf. Section 3);
- a novel tool interoperability approach which enables (almost) automated trace capture (cf. Section 4).

The applicability of these results has been demonstrated by a prototypical process centered requirements engineering environment, called PRO-ART. PRO-ART records the traces in a central process repository, offers interactive and situated methodical advice to the requirements engineer, and supports consistent change integration.

The implementation of PRO-ART has empowered us to make practical trace capturing experiments. The first self experiences of modelling small systems, e.g. the standard library system, have shown that some functions of the PRO-ART environment were not suitable for every days use, e.g. displaying dependencies in a textual form. After improving PRO-ART by correcting these early pitfalls, e.g. by

providing a graphical dependency editor with rich browsing facilities, we have demonstrated the PRO-ART environment at numerous events.

Among others, these demonstrations resulted in an intensive collaboration with chemical engineers. Within this collaboration, the process of constructing a simulation model for chemical processes in industrial plants has been analyzed and an adaptation of the PRO-ART environment for supporting the modelling of chemical processes, called PRO-ART/CE, was made. Experiments with the PRO-ART/CE prototype have indicated that the traceability ideas underlying the PRO-ART environment provide helpful support for the chemical engineers for modelling chemical processes [Jarke and Marquardt, 1995; Lohmann and Marquardt, 1996].

The experiments as well as the reactions to the various presentations of PRO-ART have provided us with significant and helpful feedback both from industry and research which can be summarized as

- some trace informations are obvious and should therefore not be recorded, e.g. that an entity was created by the *create entity* action;
- trace strategies should be explicitly definable depending on the needs of a particular project;
- the large amount of trace data requires special version and configuration management support;
- embedding of multi-user extensions is essential, e.g. task lists, cooperation facilities;
- the integration of existing tools in the PRO-ART environment should be enabled, e.g. by suitable wrapping techniques;
- access rights to trace information should be controlled (privacy of the traces).

The feedback obtained has led to a re-design and re-implementation of the PRO-ART environment, called PRO-ART 2.0. The main advantages of the new implementation are:

- a standardized tool architecture for process-integrated tools;
- the trace data are recorded in relational databases; thereby a large amount of data can be handled and transaction control is enabled;
- trace strategies can be explicitly defined in the trace repository;
- unique look and feel of all PRO-ART tools;
- fully process integrated tools (in PRO-ART 1.0 part of the tool interactions were hardcoded).

Acknowledgments: This work was supported in part by the European Community under ESPRIT Basic Research Project 6353 (NATURE) and the European-Australian Cooperation Project ISI (ECAUS003).
Thanks are due to Matthias Jarke for encouragement and freedom which have made the work on PRO-ART possible and to Stefanie Kethers and Klaus Weidenhaupt for their helpful comments on earlier drafts of this paper. I am grateful to my colleagues Peter Haumer, Ralf Klamma, Klaus Weidenhaupt and Ralf Dömges who have designed and implemented major parts of PRO–ART 2.0 and to the numerous students who have contributed to PRO-ART 2.0 and implemented PRO-ART 1.0.

References

[Alford, 1990]
M. W. Alford. Software Requirements Engineering Methodology (SREM) at the Age of Eleven – Requirements Driven Design. In P. A. Ng and R. T. Yeh (Eds.), *Modern Software Engineering*. Van Nostrand Reinhold, 1990.

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

[Davis, 1990]
A. M. Davis. The Analysis and Specification of Systems and Software Requirements. In R. H. Thayer and M. Dorfman (Eds.), *Systems and Software Requirements Engineering*, pp. 119–134. IEEE Computer Society Press — Tutorial, 1990.

[Delisle and Schwartz, 1986]
N. M. Delisle and M. D. Schwartz. Neptune: A hypertext system for CAD applications. In *Proc. of the ACM SIGMOD '86*, pp. 132–142, Washington, 1986.

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

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

[Dowson, 1992]
M. Dowson. Consistency Maintenance in Process Sensitive Environments. In *Proc. of the Process Sensitive SEE Architectures Workshop*, Boulder, CO, September 1992.

[Fischer *et al.*, 1989]
G. Fischer, R. McCall and A. Morch. JANUS: Integrating Hypertext with a Knowledge-based Design Environment. In *Proc. of the Hypertext*, pp. 105–117, Pittsburgh, PA, November 1989.

[Fischer, 1993]
G. Fischer. Integrating Construction and Argumentation in Domain-Oriented Design Environments. In *Proc. of the First Intl. Symp. of Requirements Engineering*, pp. 284, San Diego, CA, January 1993. IEEE Computer Society Press.

[Flynn and Dorfmann, 1990]
R. F. Flynn and D. Dorfmann. The Automated Requirements Traceability System (ARTS): An Experience of Eight Years. In R. H. Thayer and M. Dorfman (Eds.), *Systems and Software Requirements Engineering*, pp. 423–438. IEEE Computer Society Press — Tutorial, 1990.

[Garg and Scacchi, 1990]
P. K. Garg and W. Scacchi. A Hypertext System to Manage Software Life-Cycle Documents. *IEEE Software* (May 1990), pp. 90–98.

[Gotel and Finkelstein, 1993]
O. Gotel and A. Finkelstein. An Analysis of the Requirements Traceability Problem. Technical Report TR-93–41, Imperial College, Department of Computing, 1993.

[Gotel and Finkelstein, 1994a]
O. Gotel and A. Finkelstein. An Analysis of the Requirements Traceability Problem. In *Proc. of the First Intl. Conf. on Requirements Engineering*, pp. 94–102, Colorado Springs, CO, April 1994. IEEE Computer Society Press.

[Gotel and Finkelstein, 1994b]
O. Gotel and A. Finkelstein. Modeling the Contribution Structure Underlying Requirements. In *Proc. of the First Intl. Workshop on Requirements Engineering: Foundation of Software Quality*, Utrecht, The Netherlands, August 1994. Augustinus-Verlag.

[IEEE-830, 1984]
IEEE-830. Guide to Software Requirements Specification. 1984. ANSI/IEEE Std. 830.

[ISO/IEC, 1990]
. ISO/IEC. *Information Technology – Information Resource Dictionary Systems (IRDS) – Framework*. ISO/IEC Intl. Standard, 1990.

[Jarke and Marquardt, 1995]
M. Jarke and W. Marquardt. Design and Evaluation of Computer Aided Process Modelling Tools. In *Intelligent Systems in Process Engineering, IPSE '95*, Snowmass, USA, 1995.

[Jarke and Pohl, 1993]
M. Jarke and K. Pohl. Establishing Visions in Context: Towards a Model of Requirements Processes. In *Proc. of the Intl. Conf. on Information Systems*, Orlando, FL, December 1993.

[Jarke *et al.*, 1994]
M. Jarke, S. Eherer, R. Gallersdörfer, M. Jeusfeld and M. Staudt. ConceptBase – A deductive Object Base Manager. *Journal on Intelligent Information Systems* (1994). also appeared as technical report, RWTH-Aachen, Aachener Informatik Berichte, No. 93–14.

[Jeusfeld, 1992]
M. Jeusfeld. *Change Control in Deductive Object Bases*. INFIX Pub, Bad Honnef, Germany, 1992. (in German).

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

[Lohmann and Marquardt, 1996]
B. Lohmann and W. Marquardt. On the Systematization of the Process of Model Development. In *Proc. of the Sixth Europ. Symp. on Computer Aided Process Engineering*, Rhodes, Greece, 1996.

[Mylopoulos *et al.*, 1990]
J. Mylopoulos, A. Borgida, M. Jarke and M. Koubarakis. Telos: Representing Knowledge about Information Systems. *Transactions on Information Systems 8*, 4 (1990), pp. 325–362.

[Pohl and Haumer, 1995]
K. Pohl and P. Haumer. HYDRA: A Hypertext Model for Structuring Informal Requirements Representations. In *Proc. of the Second Intl. Workshop on Requirements Engineering: Foundation of Software Quality (REFSQ'95)*, June, 12–13, Jyväskylä, Finland, 1995. Augustinus, Aachen, Germany.

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

[Pohl *et al.*, 1994b]
K. Pohl, G. Starke and P. Peters. REFSQ'94: Workshop Summaries. *EMISA Forum*, 2 (August 1994), pp. 87–95. also appeared in ACM-Sigsoft Notes, Jan. 1995.

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

[Pohl, 1996a]
K. Pohl. *Process Centered Requirements Engineering*. RSP marketed by J. Wiley & Sons Ltd., UK, 1996.

[Pohl, 1996b]
K. Pohl. *Encyclopedia of Computer Science and Technology*, Chapter Requirements Engineering. Marcel Dekker, Inc., N.Y., 1996. to appear.

[Potts and Bruns, 1988]
C. Potts and G. Bruns. Recording the Reasons for Design Decisions. In *Proc. of the Tenth Intl. Conf. on Software Engineering*, Singapore, April 1988.

[Ramesh and Dhar, 1992]
B. Ramesh and V. Dhar. Supporting Systems Development by Capturing Deliberations During Requirements Engineering. *IEEE Transactions on Software Engineering 18*, 6 (1992), pp. 498–510.

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

[Ramesh *et al.*, 1995]
B. Ramesh, T. Powers, C. Stubbs and M. Edwards. Implementing Requirements Traceability: A Case Study. In *Proc. of the Second IEEE Int. Symposium On Requirements Engineering (RE'95)*, pp. 89–95, March 27–29, York, UK, 1995.

[Ramesh, 1993]
B. Ramesh. A Model of Requirements Traceability for Systems Development. Technical report, Naval Postgraduate School, Monterey, CA, September 1993.

[Wolf and Rosenblum, 1993]
A. L. Wolf and D. S. Rosenblum. A Study in Software Process Data Capture and Analysis. In L. Osterweil (Ed.), *Proc. of the Second Intl. Conf. on the Software Process*, pp. 115–124, Berlin, Germany, February 1993. IEEE Computer Society Press.

[Wright, 1991]
S. Wright. Requirements Traceability – What? Why? and How? In *Proc. of the Colloquium on Tools and Techniques for Maintaining Traceability during Design*, pp. 1–2, London, UK, December 1991. IEE Professional Group C1 (Software Engineering), IEE, London, UK.