nks

Nordisk kernesikkerhedsforskning
Norrænar kjarnöryggisrannsóknir
Pohjoismainen ydinturvallisuustutkimus
Nordisk kjernesikkerhetsforskning
Nordisk kärnsäkerhetsforskning
Nordic nuclear safety research

NKS-133
ISBN 87-7893-195-9

MORE
Management of Requirements in NPP Modernisation Projects - Project Report 2005

Atoosa P-J Thunem, Rune Fredriksen and Harald P-J Thunem
Institute for Energy Technology, IFE, Norway

Olli Ventä, Janne Valkonen and Jan-Erik Holmberg
VTT Technical Research Centre of Finland

April 2006

Abstract

The overall objective of the project MORE is to improve the means for managing the large amounts of evolving requirements in Nordic NPP modernisation projects. In accordance to this objective, the activity will facilitate the industrial utilisation of the research results from the project TACO. On the basis of experiences in the Nordic countries, the overall aim of the TACO project has been to identify the best practices and most important criteria for ensuring effective communication in relation to requirements elicitation and analysis, understandability of requirements to all parties, and traceability of requirements. The project resulted in the development of a traceability model for handling requirements from their origins and through their final shapes. Particular emphasis for the MORE project in 2005 was put on utilising a prototype of a tool (TRACE) intended to support an adopted approach to dependable requirements engineering, suitable for modelling and handling large amounts of requirements related to all stages of the systems development process and not only those traditionally including requirements at high-level stages.

Key words

MORE, tracability of requirements, dependable requirements engineering, TRACE

NKS-133
ISBN 87-7893-195-9

Electronic report, April 2006

The report can be obtained from
NKS Secretariat
NKS-775
P.O. Box 49
DK - 4000 Roskilde, Denmark

Phone +45 4677 4045
Fax +45 4677 4046
www.nks.org
e-mail nks@nks.org

AUTOMATION

MORE
Management of Requirements in NPP Modernisation Projects
- Project Report 2005 -

Atoosa P-J Thunem, Rune Fredriksen, Harald P-J Thunem
IFE

Olli Ventä, Janne Valkonen, Jan-Erik Holmberg
VTT

Foreword

This document constitutes the 2005 report for the project MORE: Management of Requirements in NPP Modernisation Projects (NKS-R project number NKS_R_2005_47, started on July 1, 2005). The project aims at the industrial utilisation of the results from the project TACO: Traceability and Communication of Requirements in Digital I&C Systems Development (NKS-R project number NKS_R_2002_16, completed in June 30, 2005).

The purpose of the report is to document the research work and related activities in the period July 1 – December 31 in 2005, including dissemination activities. The work has been concentrated on adopting a new approach for requirements engineering and its supporting tool. This tool makes use of the main elements of the traceability model proposed in the project TACO. Furthermore, efforts have been put on very active dissemination of the background and objectives of the project MORE within the nuclear community and towards NPPs that do carry out modernisation projects. Together with the dissemination materials in terms of papers and presentations, this report provides a sound basis for the work in 2006.

Halden, January 2006

Atoosa P-J Thunem

Table of contents

Abbreviations

IFE	Institute for energy technology
MORE	Management of Requirements in NPP Modernisation Projects
NKS	Nordic nuclear safety research
NPP	Nuclear power plant
SKI	Swedish Nuclear Power Inspectorate
STUK	Radiation and Nuclear Safety Authority of Finland
TACO	Traceability and Communication of Requirements in Digital I&C Systems Development (NKS project number NKS_R_2002_16)
VTT	Technical Research Centre of Finland

Summary

This document constitutes the 2005 report for the project MORE: Management of Requirements in NPP Modernisation Projects (NKS-R project number NKS_R_2005_47, started on July 1, 2005). The project aims at the industrial utilisation of the results from the project TACO: Traceability and Communication of Requirements in Digital I&C Systems Development (NKS-R project number NKS_R_2002_16, completed in June 30, 2005).

The overall objective of the project MORE is to improve the means for managing the large amounts of evolving requirements in Nordic NPP modernisation projects. In accordance to this objective, the activity will facilitate the industrial utilisation of the research results from the project TACO. On the basis of experiences in the Nordic countries, the overall aim of the TACO project has been to identify the best practices and most important criteria for ensuring effective communication in relation to requirements elicitation and analysis, understandability of requirements to all parties, and traceability of requirements. The project resulted in the development of a traceability model for handling requirements from their origins and through their final shapes. The traceability model is in terms of a *requirement change history tree* built up by linking the different requirements together through the definition of a simplest syntactical form for a requirement being a *paragraph*, through a complementary set of basic requirement *change types*, and through mechanisms for requirement *categorisation*.

On the basis of compiled experiences on the problem of handling large amounts of information in relation to Nordic modernisation projects, the project MORE will investigate how to handle large amounts of evolving requirements in modernisation projects, where the original requirements and their patterns of development are subject to change. Developing pragmatic mechanisms for change management is therefore an important prerequisite for the success of the project MORE.

As the aim is to improve and maturate the results from the project TACO, the efforts during the period July 1 – December 31 in 2005 have therefore been put on the following:

- Adopting an approach for dependable requirements engineering and its supporting tool. The tool makes use of the main concepts of the traceability model proposed in the project TACO, but also responds to other aspects and includes other features.
- Disseminating the background and objectives of the project MORE, in order to establish collaboration with NPPs involving in modernisation activities. Such collaboration is a prerequisite for the success of the project.

The activities related to the project MORE included a presentation in an NKS initiated seminar on decommissioning projects in Nordic countries (Roskilde, Denmark, September 13-15, 2005), a paper presentation and demonstration of the prototype during SAFECOMP 2005 conference (Fredrikstad, Norway, September 28-30, 2005), a paper presentation and demonstration during the EHPG 2005 (Lillehammer, Norway, October 17-21, 2005), a project meeting (October 18, 2005), and a paper presentation and demonstration during an IAEA special meeting (Espoo, Finland, November 22-24, 2005). The participation during the two latter events also included meetings and discussions with the staff members of FORTUM who are involved in modernisation projects at Loviisa NPP.

1. Introduction

Experiences from modernisation projects at NPPs, particularly in Sweden and Finland, indicate the importance of adequate structure and modularisation of the requirements. It is important to handle the evolution of the requirements and the completeness with respect to the requirement sources, supported by some formalism for structuring the requirements. A particular issue is how to make an evolutionary, iterative systems engineering process that reflects the evolving nature of the requirements and their understanding, and at the same time meets the requirements set by the licensing authorities (e.g., with respect to quality assurance and documentation). An important part of such a process is traceability features making it possible to trace the requirements back to their origins and forward to their final (actual) specifications.

The overall objective of the project MORE is to improve the means for managing the large amounts of evolving requirements in Nordic NPP modernisation projects. In accordance to this objective, the activity will facilitate the industrial utilisation of the research results from the project TACO. On the basis of experiences in the Nordic countries, the overall aim of the TACO project has been to identify the best practices and most important criteria for ensuring effective communication in relation to requirements elicitation and analysis, understandability of requirements to all parties, and traceability of requirements. The project resulted in the development of a traceability model for handling requirements from their origins and through their final shapes. The traceability model is in terms of a *requirement change history tree* built up by linking the different requirements together through the definition of a simplest syntactical form for a requirement being a *paragraph*, through a complementary set of basic requirement *change types*, and through mechanisms for requirement *categorisation* [1][2][3][4].

The purpose of the present report is to document the research work and related activities to the project MORE: Management of Requirements in NPP Modernisation Projects (NKS-R project number NKS_R_2005_47, started on July 1, 2005), and carried out in the period July 1 – December 31, 2005. Particular emphasis in 2005 was put on utilising a prototype of a tool intended to support a more broad perception of requirements engineering, hence suitable for modelling and handling large amounts of requirements related to all stages of the systems development process and not only those traditionally including requirements at high-level stages. Relying on more clear and sound traceability mechanisms is one important feature amongst the intended properties of the tool. Providing tool support for the main elements of the traceability model suggested in the project TACO was also among the important issues raised by the advisory group behind the project TACO (formed through the industrial seminars arranged by the project).

Chapter 2 features a brief description of the traceability model, covering the main elements of the model. Chapter 3 describes an approach for dependable requirements engineering adopted in the project MORE. Chapter 4 covers the most important components of the prototype of a tool for supporting the approach and equally adopted in the project. Chapter 5 presents the references used to compose the report.

Appendix A features the project activity plan and organisation. Appendix B describes the main elements of the prototype, in terms of requirements, design and implementation specifications. Appendix C describes the dissemination activities in 2005.

2. The TACO Traceability Model

As stated previously, the overall aim of the TACO project has been to identify the best practices and most important criteria for ensuring effective communication in relation to requirements elicitation and analysis, understandability of requirements to all parties, and traceability of requirements. The project resulted in the development of a traceability model for handling requirements from their origins and through their final shapes. The traceability model is in terms of a *requirement change history tree* built up by linking the different requirements together through the definition of a simplest syntactical form for a requirement being a *paragraph*, through a complementary set of basic requirement *change types*, and through mechanisms for requirement *categorisation*.

2.1 The Motivation and Rationale

The three main aspects in the objectives of the project TACO were:

- Communication in relation to requirements elicitation and analysis
- Understandability of requirements to all parties
- Traceability of requirements from origin to final stages

Communication: Effective communication in relation to requirements elicitation and analysis relates to the common understanding of the requirements, which again requires adequate communication. The traceability model developed in the project TACO supports communication by providing a common basis, in terms of the *requirements change history tree* built up by linking the different requirements together through the definition of a simplest syntactical form for a requirement being a *paragraph*, through a complementary set of basic requirement *change types*, and through mechanisms for requirement *categorisation*. Thus, the model can be used for generating subsets of the change history showing the backwards or forwards traceability of given requirements. The project TACO provided guidelines for how to utilise these possibilities in practical work.

Understandability: In almost all cases of systems development, understanding the requirements continues to evolve as the development proceeds. For many large systems, the requirements are never perfectly understood or perfectly specified. By relating the development of the requirements through the traceability model, the different agents involved in the development can at any time relate a given version of a requirement to its origins, its different development stages, and its relationship to other requirements. Understandability can be further improved by providing traceability between different perspectives and forms of a requirement.

Traceability: It is implied by the foregoing discussion that traceability plays a central role in the field of requirements engineering, which aims at developing standard and systematic ways to elicit, document, classify, and analyse requirements. Since these are among the most critical activities in an engineering process, the effectiveness of traceability management may have a significant effect on the quality of this process, and thereby on the success of the application. In turn, traceability management depends on manners in managing requirement changes. Management of changes is closely related to the maintainability of a system. Typically, the requirements for a given system undergo many changes before the development is completed. These changes may be due to changes in the prospected operation environment,

but may also happen simply as a result of improved insight during the development. Work on requirements traceability can to a certain extent be seen as a response to the need for keeping track of these changes. One benefit of traceability is the identification and localisation of the side effects of a modification, and of the relationships that must be reconfirmed, thereby increasing the assurance that when changes are necessary they will be complete and consistent.

The traceability model adopted aims at forming the logic needed for formalising the activities related to change management and hence their further automation. By complementing the model with appropriate terminology, data structures and guidelines for use, the model can be adapted to different needs related to management of changes in computer-based systems, including safety-critical and security-critical systems.

2.2 The Elements of the Model

The traceability model proposed by the project TACO is in terms of a requirement change history tree built up by linking the different requirements together through the definition of a simplest syntactical form for a requirement being a paragraph, through a complementary set of basic requirement change types, and through mechanisms for requirement categorisation.

The change history tree is syntactically built up by composition of instances of eight different change types. The change types correspond to the following generic actions performed on requirements, or more generally, paragraphs [1][2]:

- Creating a new paragraph with no prior history
- Deleting an existing paragraph
- Un-deleting a deleted paragraph
- Splitting an existing paragraph, thereby creating a number of new paragraphs
- Combining existing paragraphs by a new paragraph
- Replacing existing paragraphs by a new paragraph
- Deriving a new paragraph from existing paragraphs
- Modifying a paragraph without changing its meaning

The paragraphs constitute the nodes of the tree, whereas the changes constitute the links between the nodes. On this merely syntactical basis, one can extract various kinds of information, including the following:

- All initial paragraphs
- All deleted paragraphs
- All applicable paragraphs
- The complete history of a paragraph
- The complete backwards traceability from a set of paragraphs
- The complete forwards traceability from a set of paragraphs
- The legality of a proposed requirements change

The possibility to find the backwards or forwards traceability from a set of requirements facilitates backwards and forwards branch isolation and analysis of the change history. The versatility of the representation can be further improved by extending the representation of the paragraphs to include different *parameters* that classify the requirements and provide additional information about the requirement.

When it comes to the representation of the actual *parameters*, it is important to distinguish between (1) the information that is essential to identify the paragraph, and (2) the various information associated to this parameter. Conceptually, and from a perspective of modularity, it is useful to let the nodes in the change history tree represent the necessary and sufficient information related to the identity of a paragraph. In the TACO Traceability Model, a paragraph is represented by the combination of a unique identifier for this paragraph and a version number to distinguish several versions of the same paragraph. The tree and the paragraphs can then be used and interpreted depending on the application in focus. For example, in one application, only the latest version of a paragraph can be an *applicable* paragraph (that is, a new version of a paragraph is introduced only if this replaces old versions.), whereas in another application, several versions of a paragraph can be applicable. In any case, it is possible to make duplicates of a paragraph when these are treated as different paragraphs that in reality are different only with regard to a certain parameter. This can be, e.g., the "application conditions" attached as guidelines to every single variant of the paragraph. Each variant will, however, be represented with a separate paragraph having its own identifier and version number.

It is important to note that concepts similar to those described above for the traceability model proposed in the project TACO can be found in commercial tools for version control and configuration management. Although the change types might have other names, they typically resemble those defined here. In general, however, these tools do not offer an identifiable, formally defined traceability model at the paragraph level, as almost all of these tools are file-based and not paragraph-based. Also, as the traceability model is fully generic, it can be formed according to the approach chosen for requirements engineering. Thus, an advanced approach can basically be mapped into the model.

Figure 1 shows a change history tree that is consisting of nodes represented by a pair of a paragraph identifier and a version number.

The development of the requirements in Figure 1 starts with the introduction of the paragraphs p1, p2, and p3. At later stages, another two new paragraphs are introduced; p5 and p11. All the other paragraphs are developed on the basis of these five paragraphs. Paragraphs p1 and p2 are first modified and then combined into a new paragraph p4. After a modification, this paragraph is split into four separate paragraphs p7 through p10. The latter of these paragraphs is modified and then combined with p6, originally derived from paragraphs p3 and p5, giving paragraph p12. It is certainly possible to represent the change history tree textually in such a way that the temporal relationships between the different changes are maintained.

Let us now consider other kinds of information attached to a paragraph, in terms of parameters (or attributes). The purpose of the tree is to give a complete representation of the changes and how they are related to each other. As has been argued in the foregoing, it is not necessary to represent all other types of information explicitly in the change history tree. For an implicit modelling, we can think of these relations in terms of some basic mathematical concepts:

- *Sets*: These are finite collections of objects of some type, and can be used for representing subsets of the paragraphs. By way of example, the classification of paragraphs with respect to Business plan, Requirements document, Design specification, etc, can be represented by means of separate, maybe overlapping sets corresponding to the different classification terms. Finding, say, all Business plan related requirements is then

trivial, since they are given by the corresponding set (of course, the actual identification and specification of these requirements, resulting in the establishment of the corresponding set is another matter). Checking whether a requirement belongs to the Business plan can equally be done by checking whether the given paragraph is a member of the corresponding set. On the other hand, finding the class of a given paragraph cannot be done by simple look-up but involves checking all the different sets for membership.

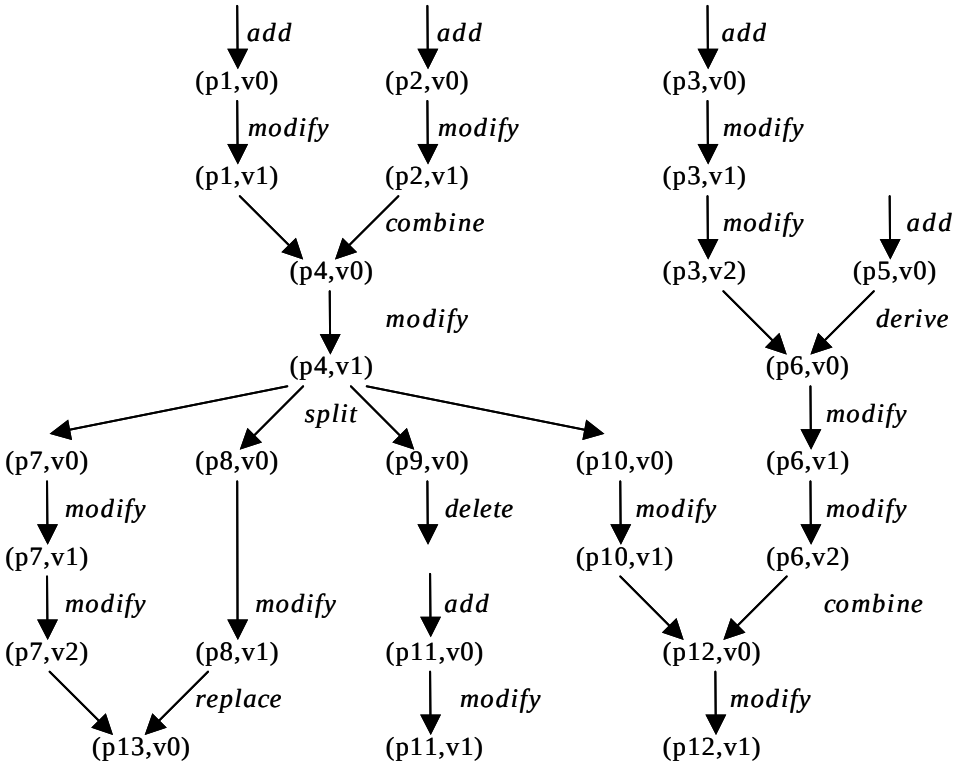


Figure 1. The example of a change history tree.

- *Mappings*: These are functions from a source set to a target set, and can be used for assigning information to the paragraphs in a simple look-up fashion. With this solution, e.g. the classification of paragraphs can be represented by mappings from the paragraphs to their classification. Finding the classification of a requirement is then simple, since it reduces to looking up the classification of that requirement. Finding all requirements is possible, but less trivial than for sets, as it involves selecting all requirements that are mapped to a certain term. On the other hand, the concept of relation is more convenient if there may be more than one class for a requirement.

- *Relations*: These are more general than mappings, since they allow an element in the source set to be associated to more than one element in the target set. With this solution, finding the classification of a requirement involves finding all elements in the target set (the classes) that are related to the given requirement. Finding all requirements related to a certain class can alternatively be understood as the inverse relation.

Sets can be considered as being implemented as simple lists. Mappings and relations can be considered as being implemented as tables. These representation concepts will suffice for rep-

resenting all information associated to the requirements. It is, however, possible to represent the same information in other ways as well, as long as consistency is maintained.

A basic piece of information related to a requirement is certainly the statement (phrasing) of the requirement. Assuming that (at most) one statement is associated to each requirement, we may think of this information as being available by means of a mapping from versioned requirements to their statements, see Table 1.

Requirement	Statement
(p1,v0)	<Statement of version v0 of paragraph p1>
(p1,v1)	<Statement of version v1 of paragraph p1>
(p2,v0)	<Statement of version v0 of paragraph p2>
...	...
(p13,v0)	<Statement of version v0 of paragraph p13>

Table 1. Mapping from requirements to their statements.

As is evident from Table 1 that the statement of a given requirement can be found by simple look-up in the table implementing the mapping. The table can be utilised in different ways. By way of example, finding all applicable requirements can be found by filtering the mapping to find the subset of the mapping that relates to applicable paragraphs only. Filling in the relevant information is an obvious task of an information system designed to support the use of the model.

3. An Approach for Dependable Requirements Engineering

This chapter describes a practical approach for dependable requirements engineering of computerised systems. The approach is the joint result of research within requirements engineering, systems modelling (mainly based on object-oriented, semi-formal and agent-oriented modelling methodologies), and model-based risk analysis and assessment [5][6][7]. The following provides some background and covers the main aspects of the approach.

3.1 The Background

Especially within information and communication technologies (ICT) and their applications in different branches, several approaches have been proposed towards a better system development process. Among the most applied is the Rational Unified Process (RUP) that provides a matrix-oriented lifecycle model highly supporting the time aspect of the lifecycle. Here, the road map is formed by two main activity categories: disciplines followed to develop the system and phases related to its life-path. The workload in each phase is decided by the actual discipline in focus: More elaboration phase is required during the design discipline, whereas more construction is needed during the implementation. Figure 2 illustrates another extended version of the RUP model, called the Enterprise Unified Process (EUP).

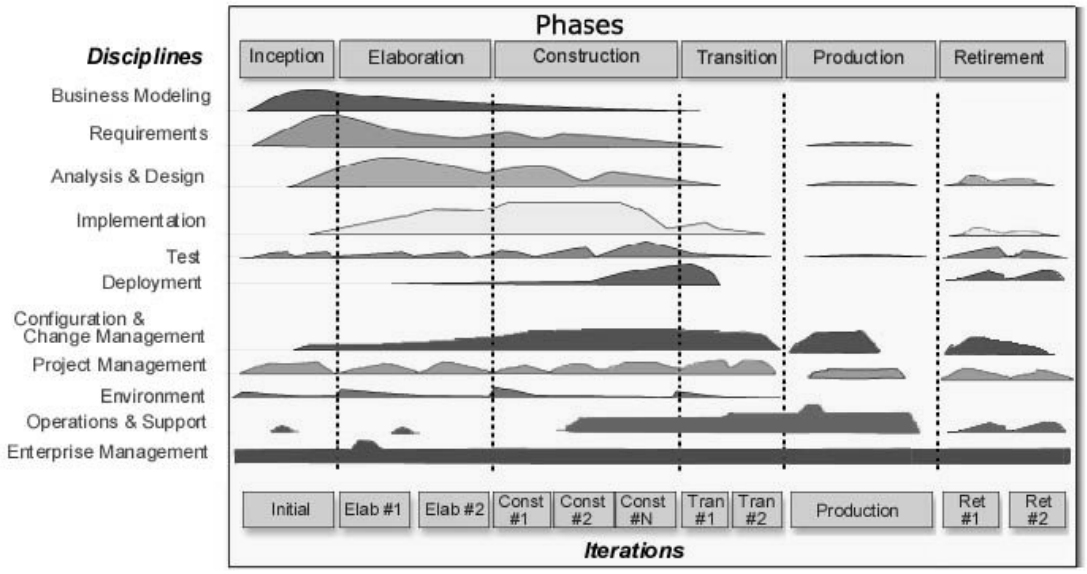


Figure 2. The Enterprise Unified Process (EUP).

Nevertheless, despite the availability of detailed guidelines for sub-activities in each discipline and for the number of iterations in each phase, neither RUP nor any other lifecycle models provide guidelines on how to achieve traceability among phases and disciplines. Also, if system properties are addressed at all, the implied concern is almost entirely on functional and operational factors, and not other dependability factors such as safety, security, reliability, flexibility and maintainability. To exemplify, there exist no instructions on how the security issues associated with the specific system architecture or application domain can influence the length of a certain phase, or the amount of certain sub-activities during the iterations [6]. The lack of addressing dependability factors in available life cycle models explains also why the concept of risk and risk analysis has not been an issue to take into account for these models.

As already mentioned, change management is closely related to the maintainability of the system development process and the result (product) of this process, the operational and applied system itself. In reality, clear and sound change management mechanisms are necessary to ensure the dependability of the task of requirements engineering. Typically, the requirements at each stage of the development process of a system undergo many changes before the development is completed. These changes may be due to changes in the prospected operation environment, but may also happen simply as a result of improved insight during the development or a desire to incorporate technological advances into the development stages (use of new methods, procedures, tools, etc.). Thus, it appears that change management mechanisms themselves depend highly on whether they utilise requirements traceability mechanisms.

3.2 The Four Pillars of the Approach

The approach for dependable requirements engineering is different from the traditional manner of understanding requirements engineering, as the approach advocates a perception of a requirement to be applicable for *all stages* of the system development process (or system lifecycle) and not only the high-level stages. Based on this perception, the requirements should

be identified, specified, validated and verified, and finally implemented for all stages of the system development process. Referring to the disciplines in the RUP/EUP model shown in Figure 2, this means that requirements should be defined and specified in an inter-disciplinary fashion.

Furthermore, the approach aims at making a computerised system and its lifecycle analysable with regard to several *dependability factors* such as safety, security, reliability, flexibility and maintainability [5]. This means that dependability factors are integrated into the lifecycle, thus also integrated into the very definition of dependability-critical requirements. Additionally, the approach recognises the relationship between how a requirement can be met and how it can be opposed to, due to unexpected or unwanted events. Thus, the requirements expressed in this approach are also *risk-informed* [5][7]. Finally, the approach acknowledges the importance of well-defined *traceability mechanisms* to provide links between the requirements belonging to a particular stage or different stages of the lifecycle.

In order to validate and verify the requirements and their changes in a dependable manner, different analyses are needed as an integrated part of carrying out each stage of the development process. The most important analysis is that of thorough risk analysis with focus on one or several dependability factors that need to be analysed and assessed, before introducing any progress or any change. There is a need for traceability of the requirements related to a specific risk analysis method or process, in accordance with the requirements of system development process and its product a risk analyst is supposed to analyse.

From the above, the four main aspects of the approach are:

1. Requirements engineering for all stages of the system development process
2. Integrating dependability factors into the system development process, hence into very definition of the requirements
3. Integrating risk analysis and assessment into the system development process and thus requirements engineering, so that risks are associated with the dependability-critical requirements
4. Utilising traceability mechanisms for providing well-defined links amongst the requirements within a stage and across the stages

The next chapter explains the main elements of a tool that aims to support the above approach by also utilising the core elements of the traceability model developed in the project TACO. This tool is called *TRACE: Traceability of Requirements for Analysable Computerised Environments* [8].

4. TRACE: A Tool for Traceability of Requirements for Analysable Computerised Environments

As explained earlier, providing tool support for the main elements of the traceability model suggested in the project TACO was also among the important issues raised by the advisory group behind the project TACO (formed through the industrial seminars arranged by the pro-

ject). To provide tool support for not only the traceability model but in higher degrees for the approach described in Chapter 3, the first prototype of the tool TRACE was developed. The work started in the beginning of 2005 and in parallel with the finalisation activities related to the project TACO.

The ideas behind the features of the tool were all concentrated on the four main components of the approach for dependable requirements engineering. Furthermore, it has been considered as a very important feature that the tool can be expanded as well as tailor-made (specialised), as response to different needs and applications.

This chapter describes the basic elements of TRACE that in combination can be used to achieve the objectives behind the approach proposed in an efficient and practical manner. The following summarises therefore the main possibilities in TRACE:

- Traceability between the requirements at a particular stage of the system lifecycle
- Traceability between the requirements defined for different stages of the system life-cycle
- Traceability of changing or changed requirements throughout the system lifecycle for better change management
- Traceability of dependability-related requirements throughout the system lifecycle for better dependability analysis
- Traceability of risk factors with respect to a certain dependability factors, and thus traceability of all risk-informed requirements related to these risk factors

The basic elements of TRACE are *Paragraphs*, *Changes*, *Change Types*, *Links*, *History Trees*, and *Sets*. The following focuses on their description and their applications.

4.1 The Main Elements of TRACE

4.1.1 Paragraphs

The traceability approach and associated tool focuses on the concept of *Paragraphs*, which are objects containing the text describing a specific requirement. Paragraphs are associated with the following list of attributes:

id	Automatically generated unique identifier.
label	Textual short label.
version	Version number. A Paragraph can be subject to a number of different Changes, where some will cause the creation of Paragraphs with a new label, and other the creation of Paragraphs with the same label but incremented version number (see description of Change class below).
time	Time of creation.
status	Status attribute (see table below for possible values).

description	Paragraph content, which for e.g. software development will be the textual description of a requirement. The purpose of the traceability approach is to keep a track of all changes to this attribute across different Paragraph versions and across all development phases.
change_in	The change that caused the creation of the Paragraph.
changes_out	List of changes performed on the Paragraph causing the creation of other Paragraphs.
origins	List of paragraph origins. See description of Link (which is the class implementing the concept of origin) below.

The *status* attribute of a Paragraph or a Change can take the following values:

None	Default Paragraph/Change status.
Created	Indicates that the Paragraph is the first in a list of Paragraphs with the same label, but different version numbers. The Paragraph is the result of either a *create* Change or a Change performed on another Paragraph which creates one or more new Paragraph(s) (*derive, split, combine*...).
Trace	The Paragraph/Change is part of a trace result, e.g. a backward trace. The Paragraph/Change will be highlighted in the history tree display.
Highlight	The Paragraph/Change is highlighted in the history tree display.
Deleted	The Paragraph has been explicitly deleted (having been subject to the *delete* Change).

4.1.2 Changes

The *Change* class contains the properties of a single Change from one or more Paragraphs into one or more Paragraphs. Changes are associated with the following list of attributes:

id	Automatically generated unique identifier.
type	Type of Change (see description of *ChangeType* class below).
sources	List of input Paragraphs to this Change.
targets	List of output Paragraphs from this Change.
status	Status attribute (see table above).
user_id	The identifier of the user responsible for introducing the Change.
time	Time of Change introduction.
reason	Textual description of the reason for introducing the Change.
basis	The basis for introducing the Change (see table below).

The *basis* parameter is used to provide some description of the basis for applying the Change to one or more Paragraphs:

Method	The Change has been introduced due to the outcome of some analysis method, e.g. a HazOp analysis, which has suggested that the Paragraph(s) must be up-

	dated due to some shortcoming.
Expert	The Change has been introduced due to input from some expert (expert judgement).
None	No special basis is given for the Change.

4.1.3 Change Types

The *ChangeType* class is used to define different types of Changes. The ChangeType class is associated with the following list of attributes:

label	Unique label.
para_in	The number of input Paragraphs (possible values are “0”, “1”, “1 or more” and “2 or more”).
para_out	The number of output Paragraphs (same as above).
description	Textual description of change type.
result_status	Status of output Paragraph(s) (see table above).
update	How to update the output Paragraphs label and version (see below).

The *update* value defines how the Paragraph label and version number are determined for a Paragraph resulting from a Change:

No update	The output Paragraph has the same label and version number as the input Paragraph.
New label	The output Paragraph is given a new label.
Increment version number	The version number of the output Paragraph is incremented relative to the input Paragraph.

For use in software development, the default Change types include:

- create
- modify
- combine
- replace
- split
- derive
- delete
- un-delete

An example of a change type is “modify”, where the attribute values are given in the following table:

label	“modify”
para_in	1
para_out	1
description	“This change denotes a modification of the paragraph”
result_status	None
update	Increment version number

Only one Paragraph at a time can be subject to a *modify* Change, and the result is a single Paragraph where the label remains the same, while the version number is incremented.

4.1.4 Links

In many cases it can be useful to include information regarding the reason for introducing a Paragraph. Examples of this information can be:

- a textual reference from a brainstorming meeting
- an IAEA safety standard, suggesting the introduction of a specific safety function
- a web-page with statistical data showing the potential improvements in system reliability by developing in accordance with certain object-oriented metrics
- a link between a Paragraph in the *implementation* phase and a Paragraph in the *design* phase, indicating that the former fulfils the requirements of the latter

The *origin* attribute of a Paragraph is used to provide information regarding where the *idea* of the Paragraph originated, and it can be a combination of textual descriptions, files, hypertext links, and other Paragraphs. The Link type implements the concept of the origin attribute, and the attributes associated with the Link type are:

type	Type of link
string	Textual information

Examples of Links are given in the following table:

A textual link

```
object Link
      type: TEXT
      string: “This Paragraph was included due to a discussion at project meeting
      in Halden on 2005-04-08”
end
```

A file link

```
object Link
      type: FILE
      string: “c:\projects\more\p08-basis.doc”
end
```

A hypertext link

```
object Link
      type: HYPERTEXT
      string: “http://standards.ieee.org/catalog/olis/index.html”
end
```

A Paragraph link

```
object Link
      type: PARAGRAPH
      string: “PA_002389” (the ID of a particular Paragraph)
end
```

4.1.5 History Trees

The *HistoryTree* class is used to hold all required information about one history tree, including all Paragraphs and Changes. An example of a history tree is shown in Figure 3. History trees will show the development of a number of Paragraphs as they are subject to Changes, and for software development projects a typical use is to create one history tree for each development phase.

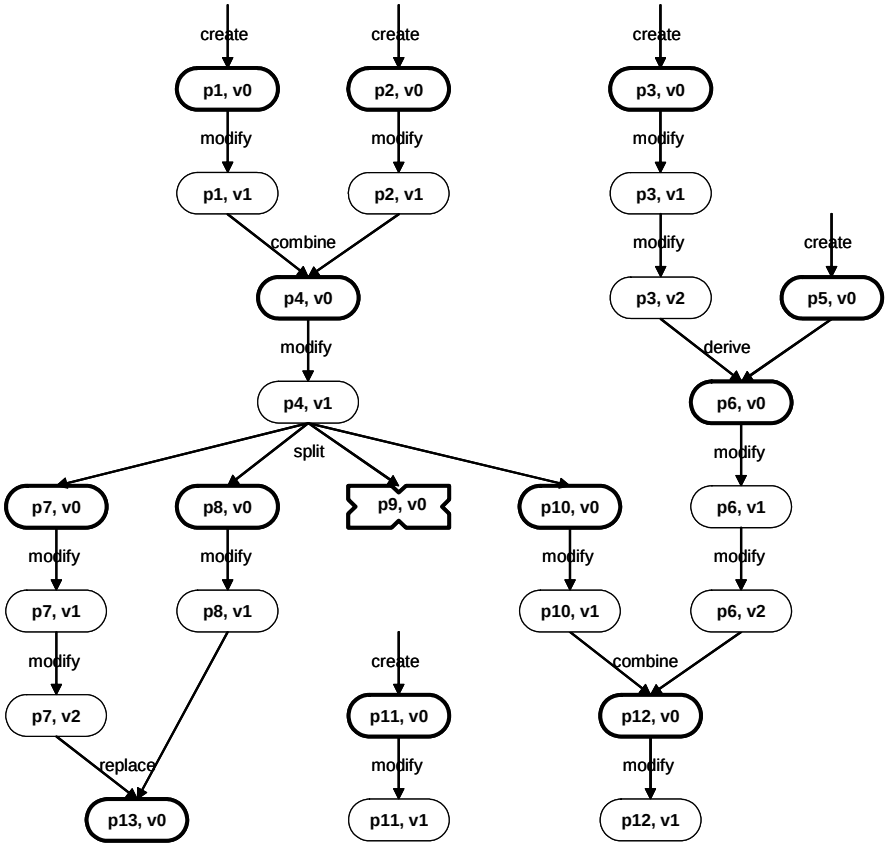


Figure 3. Example history tree.

The list of attributes associated with a HistoryTree is:

id	Automatically generated unique identifier.
label	Textual label provided by user.
paragraphs	List of Paragraphs.
changes	List of Changes.
create_time	Creation time.
last_change_time	Last time history tree was changed.

4.1.6 Sets

The *Set* class extends the HistoryTree class to include a list of subsets, links to parent and child sets, and information about opening and closing times and status. This allows a Set to contain any number of Paragraph objects, as well as any number of Set objects, and to maintain a derivative relationship between Sets.

The list of attributes associated with a Set (in addition to those inherited from the HistoryTree class) is:

sets	List of subsets.
parent	Parent set.

child	Child set.
open	Indicates whether Set is open or closed.
close_time	Time the Set was closed.

One typical use of the Set could e.g. be to group all *security-related* requirements into a separate Set, facilitating a subsequent *security analysis* and its associated *risk analysis*.

A Set will be able to compare its content (specifically its list of Paragraphs) to the content of another Set, i.e. which Paragraphs are common to both Sets, and which Paragraphs are unique. This ability is particularly relevant in *change management*, where the difference between two versions of the same software with regard to which Paragraph versions they implement is readily apparent.

An open Set can have its content (i.e. list of paragraphs, history trees and subsets) changed, while a closed set is not editable. In software development this will typically correspond to a version of the software where the feature set has been frozen.

4.2 Basic analyses

Using the features of the classes described in Section 4.1, the tool can perform a number of analyses relevant to software development and change management:

Created Paragraphs	Whenever a new Paragraph is created, either “from scratch” or by certain Changes to other Paragraphs (e.g. derive, split, combine...), the Paragraph is marked as “Created”.
Current Paragraphs	The current or most recently updated version of a Paragraph is found by iterating through the list of Paragraphs and for each Paragraph label find the Paragraph with the highest version number. (Paragraphs that have been explicitly deleted are not included in this search)
Deleted Paragraphs	Whenever a Paragraph is deleted, it is marked as “Deleted”.
Paragraph History (forward/ backward)	The Paragraph history for any Paragraph can be determined by finding all versions of the selected Paragraph, all Changes affecting these versions, as well as the relevant version of all Paragraphs included in these Changes. This is straightforward, as all *Paragraph objects* contain lists of “incoming” and “outgoing” Changes, and all *Change objects* contain lists of “input” and “output” Paragraphs.
Paragraph Trace (forward/ backward)	**Forward:** Forward traceability relates to the development of Paragraphs starting with a selected Paragraph. The result will include all Paragraphs affected by the selected Paragraph (see Figure 4). The trace is performed by a recursive search through all output Changes starting with the selected Paragraph. The search through a sub-tree is halted once a Paragraph without any output Changes is reached. **Backward:** Given a Paragraph, we want to find the development of Paragraphs that leads to this Paragraph, i.e. the minimum fragment of the Change history that has influenced the development of the given Paragraph

(see Figure 5).

The trace is performed by a recursive search through all input Changes starting with the selected Paragraph. The search through a sub-tree is halted once a Paragraph whose input is a “create” Change is reached.

Origin Trace

The *origin* parameter in the Paragraph class provides links to information used when creating a Paragraph. This information could e.g. be a textual description of why the Paragraph should be included, a shortcut to a file, a hypertext link to an IEEE standard used as basis for the Paragraph, or a link to another Paragraph in a different history tree. A typical use of the origin parameter could be during a software development project, where a separate history tree is created for each development phase (requirement, design, implementation, test...). Here, each Paragraph would represent a specific version of a specification, and often a specification in the design phase would be based on a specification in the requirement phase (see Figure 6).

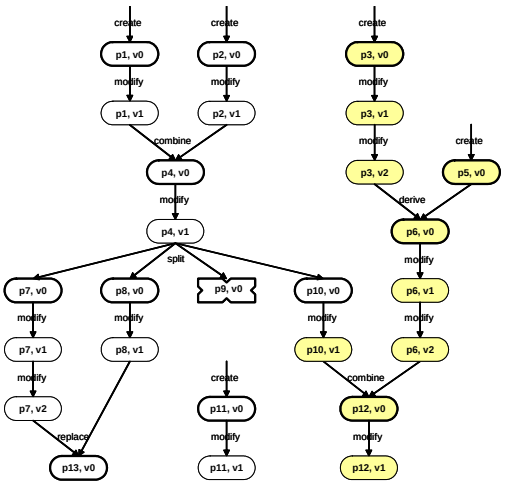


Figure 4. Forward trace from (p3, v0).

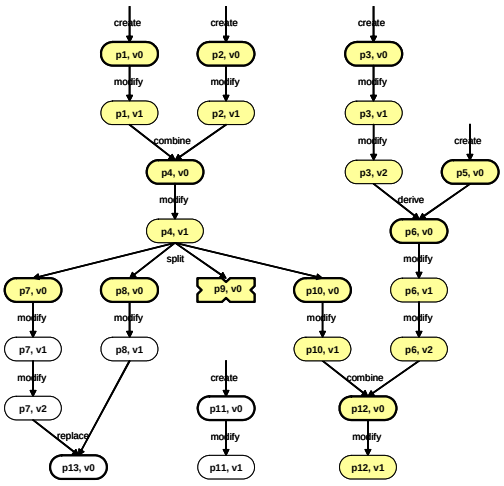


Figure 5. Backward trace from (p12, v1).

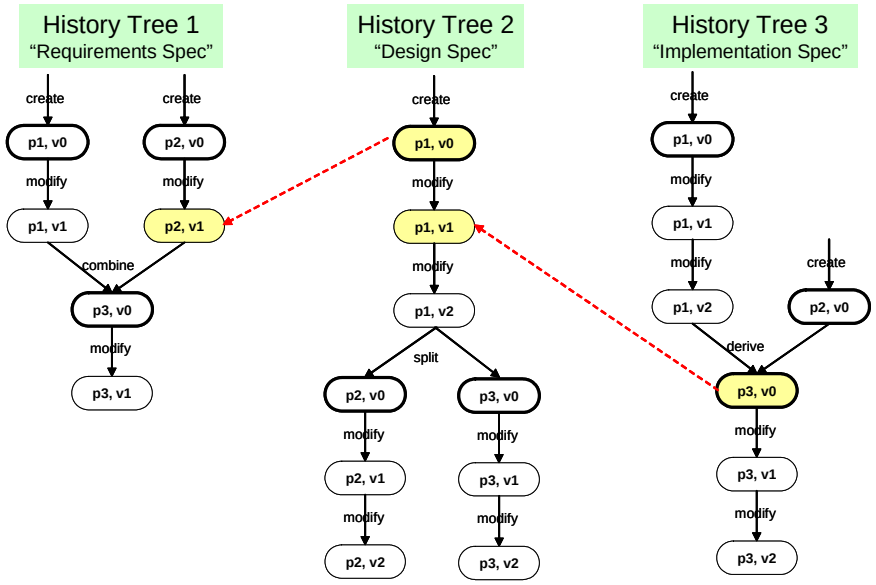


Figure 6. Origin trace. The dotted lines are links from Paragraphs in one development phase to a previous phase.

5. References

[1] T. Sivertsen et. al., "Traceability and communication of requirements in digital I&C systems development" – Project report 2004, NKS-103, April 2005.

[2] T. Sivertsen et. al., "Traceability and communication of requirements in digital I&C systems development" – Final report 2005, NKS-115, October 2005.

[3] T. Sivertsen et. al., "The TACO approach for traceability and communication of requirements", Proceedings of SAFECOMP 2005, September 2005.

[4] T. Sivertsen et. al., "Traceability and communication of requirements in digital I&C systems development" – The TACO project, in: Proceedings of the Man-Technology-Organisation Sessions, Enlarged Halden Programme Group Meeting, Lillehammer, Norway, October 2005.

[5] A. P-J Thunem, "Modelling of Knowledge Intensive Computerised Systems Based on Capability-Oriented Agent Theory (COAT)", International IEEE Conference on Integration of Knowledge Intensive Multi-Agent Systems, IEEE-KIMAS'03 (58-63), September 2003, Cambridge (MA), USA.

[6] A. P-J Thunem, "A Framework for Dependable Development Process of Complex Computerised Systems", the joint European Safety and Reliability 2004 (ESREL04) and the 7th International Probabilistic Safety Assessment and Management (PSAM7) conference (902-907), June 2004, Berlin, Germany.

[7] A. P-J Thunem, "An Approach for Dependable Requirements Engineering and Change Management of Dependability-Critical Computerised Systems", To be published and presented in the 8th International Probabilistic Safety Assessment and Management (PSAM8) conference, May 2006, New Orleans (LA), USA.

[8] A. P-J, Thunem, H. P-J Thunem, "TRACE: Traceability of Requirements for Analysable Computerised Environments", IAEA Technical Meeting on Implementing and Licensing Digital I&C Systems and Equipment in Nuclear Power Plants, November 2005, Espoo, Finland.

6. Appendix A: Project Organisation and Activities

6.1 Project Organisation

The project is led by Atoosa P-J Thunem (IFE), and comprises the following organisations and persons:

Organization	Address	Project participants
IFE	Institute for energy technology P.O. Box 173 NO-1751 Halden Norway	Atoosa P-J Thunem +47 69 212322 (atoosa.p-j.thunem@hrp.no) Rune Fredriksen +47 69 212430 (rune.fredriksen@hrp.no) Harald P-J Thunem +47 69 212278 (harald.p-j.thunem@hrp.no)
VTT	VTT Industrial Systems P.O. Box 1000 FIN-02044 VTT Finland	Olli Ventä +358 20 722 456 6556 (Olli.Venta@vtt.fi) Janne Valkonen +358 20 722 6469 (Janne.Valkonen@vtt.fi) Jan-Erik Holmberg +358 20 722 6450 (Jan-Erik.Holmberg@vtt.fi)

The activity organisation is subject for extension by involvement of additional industrial partners. In addition, the network represented by the activity organisation is extended though the arrangement of the industrial seminars.

The project leader is responsible for organising the work within the project and for directing it towards its objectives. This includes:

- Project planning and tracking
- Establishment and maintenance of the project archive
- Establishment of good communication and cooperation within the project
- Reporting to NKS
- Coordination of activities, in particular the production of the project deliverables
- Follow up of meetings and decisions

- Securing of proper quality control, including review and approval of documents included in the project archive
- Reporting of deviations and implementation of agreed corrections

All the individual participants represent important parts of the technical competence within the project, and are responsible for contributing to the activities in such a way that the project can meet its objectives.

The funds received from NKS for the work in 2006 are estimated to cover 50% of the overall costs. The remaining 50% will be covered through the individual costs and efforts of each participating organisation. Each organisation will be responsible for ensuring that their contribution is sufficient to satisfy their fraction of the overall budget. In order to facilitate roughly the same amount of effort from IFE and VTT to the technical part of the project, an estimated 20% of the funds will be allocated for project coordination (IFE). The remaining 80% will be split equally between IFE and VTT. This gives the following split of funds:

IFE	60% (= 20% + 40%)
VTT	40%

Possible common costs related to the arrangement of project meetings and seminars will be split equally between IFE and VTT. The approximate division of costs between work, travel, and equipment is given in the Proposal Summary 2006 submitted to NKS in September 2005.

6.2 Project Activities

The activity will be carried out through a three-year period, as a strategic follow-up activity to the project TACO. The activity started on July 1, 2005 and will terminate on June 30, 2008. The project is planned to deliver two industrial seminars or international workshops closely related to the background, objectives and activities of the project, at least two organised visits to selected NPPs undertaking modernisation activities, three annual project reports, and one final report.

The activities in 2005 have been with focus on the following:

- Establishing a strategy and implementation plan for the improvement and industrial take-up and utilisation of the research results from the project TACO. This was done, amongst others, through adopting an approach for dependable requirement engineering and its supporting tool into the project MORE, as described in this report.
- Compiling experiences on the problem of handling large amounts of information in relation to Nordic modernisation projects. This has been an ongoing activity, amongst others, through communication with Nordic NPPs and in particular Loviisa NPP (FORTUM), and through dissemination and representation activities.
- Extending the industrial network from the project TACO. This has been an ongoing and very successful activity.
- Preparing and arranging the first organised visit to an NPP that currently undertakes or plans to undertake one or several modernisation activities. Through several contacts with several staff members at FORTUM involved in modernisation projects at Loviisa NPP, the project anticipates a visit to Loviisa NPP during Spring 2006.

The activities in 2006 and 2007 will carry out the implementation plan in cooperation with an extended network of industrial partners. The network established through the activity organisation and the TACO industrial seminars will be further extended and consolidated through the arrangement of industrial seminars or international workshops.

The experiences and lessons learned from the research will be reported in the annual project reports, and summarised in a final report to be produced in the first half of 2008.

The activities in 2006 will include the following:

- Establishing a strategy for continuous improvements of the results from the project, on the basis of the received feedback and gained knowledge
- Carrying out the implementation plan for the industrial take-up and utilisation of the research results
- Continuing to compile experiences on the problem of handling large amounts of information in relation to Nordic modernisation projects, amongst others, through organised visits to selected plants
- Extending the industrial network, also through disseminations and presentation of the results in Nordic and NKS related events such as seminars and workshops
- Preparing and arranging an international workshop on dependable requirements engineering (November - December 2006)

Reporting plan
The overall documentation schedule is as follows:

- January 2006: Activity report for 2005
- Spring 2006: Presentations, materials and results from an organised visit to Loviisa NPP with the aim of being granted access to documentation available on some modernisation activities
- December 2006: Presentations and materials from the international workshop
- January 2007: Activity report for 2006
- December 2007: Presentations and materials from the industrial seminar or international workshop planned for 2007
- January 2008: Activity report for 2007
- June 30, 2008: Final report

The discussions from the project meetings and industrial seminars / international workshops, and the progress of the project will be reported by means of detailed minutes.

7. Appendix B: The Complete Documentation for TRACE

The following covers the documentation for TRACE, in terms of requirements specifications, design specifications and implementation specifications.

7.1 Requirements Specifications

This section describes the requirements specification for the prototype of the tool TRACE. The requirements for the tool reflect the purposes of the tool, which are to support the approach adopted for dependable requirements engineering. The tool utilises the main elements of the traceability model proposed in the project TACO, and further development and enhancement of the model, also formed during internal meeting discussions. The following covers the overall requirements for the tool:

- The tool must be able to visualise history trees and sub-trees.
- The tool must support both graphical and textual presentation of history trees, both for whole trees and sub-trees.
- The tool should support traceability of requirements related to e.g. a given standard. Several versions of the tool may be developed, depending on customer needs.
- Data input: primary concern is to avoid duplication of work. Today, requirements are mostly written as pure text in Word.
- The tool must be able to create sets to hold paragraphs, history trees and other sets. Each set is defined by the user and labelled either as closed or opened. A closed set is never to be opened again (remain unchangeable for all future). An open set may be changed by the users.
- The sets may be related to each other by a ‘derivation’ relationship meaning that one set is used to derive the other one. These two sets are called subsequent sets.
- The sets must have information on the opening and closing dates.
- The tool must keep all historic changes to a set in an incremental manner, i.e. one needs to know the differences between any two subsequent versions. Nice to have: To know the differences between any two versions of sets.
- The tool must be able to associate explanations for all changes between subsequent set. These changes must have the possibility to associate to any number of element (e.g. procedure element, such as instructions, steps, component manipulations etc.), not only two of them. The changes thus associated can be several, referring to information elements both inside and outside the tool (e.g. minutes from revision meetings).
- The tool must support both generic and specific change types. The latter will belong to one or several groups of change types, e.g., gathered towards specific “projects”.
- The tool must provide both static and dynamic information about any change in the history tree, including the basis for a change (methodical, expert judgement or logical) and the arguments for applying the change on a paragraph.
- The tool must support distinct textual description of any change in the history tree.

7.1.1 Paragraph specifications

01-01: Paragraph attributes

The tool must associate each paragraph with (at least) these attributes:

Attribute	**Description**
Label	String
Version	Integer number
Time	Creation time
Open	True, False
Status	Created, Deleted, Other
Description	Format: rich text, html, separate XML standard, etc.
Origin(s)	Can be document references and/or other paragraphs in other phases
OriginOf (**)	Other paragraph(s) for which this paragraph is an origin
ChangeIn	Change performed on other paragraph(s) to produce this paragraph.
ChangeOut	Change performed on this paragraph to produce other paragraph(s).

01-02: Paragraph rules

1. A paragraph within a single history tree can be subject to several changes, unless the paragraph has already been deleted as a result of a change.
2. A deleted paragraph can be un-deleted using the Un-delete change type.
3. A paragraph should initially be Open, but should be set to Closed after a change.
4. It should not be possible to create a new Paragraph with the same label as an existing Paragraph in the same history tree.

7.1.2 Change Types specifications

The tool must support a number of change types, and not allow illegal paragraph changes. A change type should include at least the following attributes:

Attribute	**Description**
Label	String
ParagraphsIn	Integer number, determining allowed number of input paragraphs (typically either “0”, “1”, “1 or more” or “2 or more”)
ParagraphsOut	Integer number, determining allowed number of output paragraphs (typically either“0”, “1”, “1 or more” or “2 or more”)
Description	Textual description of change type
ResultStatus	Status of resulting paragraph(s) after change

Identification Update	Result paragraph should have either new label or incremented version number

To satisfy the requirements, a group of change types could include e.g.:

- Creation
- Modification
- Combination
- Replacing
- Splitting
- Derivation
- Deletion
- Un-deletion

02-01: Paragraph creation

The tool must support paragraph creation, and ensure legality of creation. The change type definition for “create” is:

Attribute	**Description**
Label	“create”
ParagraphsIn	0
ParagraphsOut	1
Description	“This change denotes the creation of a paragraph”
ResultStatus	Created
Identification Update	New label

02-02: Paragraph modification

The tool must support paragraph modification, and ensure legality of modification. The change type definition for “modify” is:

Attribute	**Description**
Label	“modify”
ParagraphsIn	1
ParagraphsOut	1
Description	“This change denotes a modification of the paragraph”
ResultStatus	Other
Identification Update	Increment version number

02-03: Paragraph combination

The tool must support paragraph combination, and ensure legality of combination. The change type definition for “combine” is:

Attribute	**Description**
Label	“combine”
ParagraphsIn	2 or more
ParagraphsOut	1
Description	“This change denotes a combination of 2 or more paragraphs”
ResultStatus	Created
Identification Update	New label

02-04: Paragraph replacing

The tool must support paragraph replacing, and ensure legality of replacing. The change type definition for “replace” is:

Attribute	**Description**
Label	“replace”
ParagraphsIn	1 or more
ParagraphsOut	1
Description	“This change denotes a replacement of one or more paragraphs”
ResultStatus	Created
Identification Update	New label

02-05: Paragraph splitting

The tool must support paragraph splitting, and ensure legality of splitting. The change type definition for “split” is:

Attribute	**Description**
Label	“split”
ParagraphsIn	1
ParagraphsOut	2 or more
Description	“This change denotes a split of one paragraph into 2 or more paragraphs”
ResultStatus	Created
Identification Update	New labels

02-06: Paragraph derivation

The tool must support paragraph derivation, and ensure legality of derivation. The change type definition for “derive” is:

Attribute	**Description**
Label	“derive”
ParagraphsIn	1 or more
ParagraphsOut	1
Description	“This change denotes a derivation of 1 or more paragraphs into 1 paragraph”
ResultStatus	Created
Identification Update	New label

02-07: Paragraph deletion

The tool must support paragraph deletion, and ensure legality of deletion. The change type definition for “delete” is:

Attribute	**Description**
Label	“delete”
ParagraphsIn	1
ParagraphsOut	0
Description	“This change denotes a deletion of 1 paragraph”
ResultStatus	Deleted
Identification Update	None

02-08: Paragraph un-deletion

The tool must support paragraph un-deletion, and ensure legality of un-deletion. The change type definition for “undelete” is:

Attribute	**Description**
Label	“undelete”
ParagraphsIn	1
ParagraphsOut	1
Description	“This change denotes an un-deletion of a paragraph”
ResultStatus	Other
Identification Update	None

7.1.3 Set specifications

03-01: Set creation

The tool must support the creation of different sets, such as e.g. phase sets and dependability factor sets. Each set may contain any number of paragraphs, history trees or other sets (subsets), e.g.:

Set: PHASE
Subsets: REQUIREMENTS, DESIGN, IMPLEMENTATION, TEST

Set: DEPENDABILITY
Subsets: RELIABILITY, SECURITY, SAFETY, ROBUSTNESS, MAINTAINABILITY, USABILITY, REUSABILITY, FLEXIBILITY, EFFICIENCY, PORTABILITY

03-02: Set support

The tool must support associating a paragraph, a history tree or another set to one or more sets, so that a paragraph may belong to e.g. the [PHASE] DESIGN and [DEPENDABILITY] USABILITY/FLEXIBILITY sets.

03-03: Set comparisons

The tool must be able to display differences between sets, i.e. which paragraphs, history trees and other sets are members of the selected sets and which are unique to each set.

03-04: Set status

A set can be either open (editable) or closed (non-editable). When a set is created it is initially open, and the tool shall record the time it was created. The set can be closed by the user, and the tool shall record the time it was closed. When the set is closed it cannot be altered (edited), and it cannot be re-opened.

03-05: Set relation

A set can be derived from another set, i.e. there can be a parent/child relationship. A set can have only one parent set and one child set.

03-05: Set rules

1. It should not be possible to create a new Set with the same name as an existing Set.
2. A Set should not be allowed to contain itself. This also means that a Set A should not be allowed to contain Set B, if Set B already contains Set A at some place in its Set hierarchy.

7.1.4 Display specifications

04-01: Change history list

The tool must be able to generate a textual list of the change history.

04-02: History tree

The tool must be able to display a history tree for all paragraphs within a selected phase. The tree should include all paragraphs with associated labels and versions, and all paragraph changes.

04-03: Set display in history tree

The tool must be able to highlight all paragraphs (in the history tree containing the paragraphs) belonging to a selected set, and to display this list textually.

04-04: Related sets display

The tool must be able to display all related sets graphically, i.e. visualise the parent/child relationships.

04-05: Tree animation

The tool should be able to animate a history tree, i.e. graphically and textually show the temporal development of the tree.

7.1.5 Analysis specifications

05-01: Created paragraphs list

The tool must be able to identify all created paragraphs within a selected phase or within a selected project, and display these paragraphs, both in textual form and as high-light in a history tree.

05-02: Current paragraphs list

The tool must be able to identify all current paragraphs within a selected phase or within a selected project, and display these paragraphs, both in textual form and as high-light in a history tree.

05-03: Deleted paragraphs list

The tool must be able to identify all deleted paragraphs within a selected phase or within a selected project, and display these paragraphs, both in textual form and as high-light in a history tree.

05-04: Paragraph history

The tool must provide the complete history of a paragraph in textual form, as high-light in a history tree, and as a separate history subtree. The history of a paragraph includes all versions of the paragraph, the paragraph(s) used to create it, and the paragraph(s) created by using it.

05-05: Paragraph backwards traceability

The tool must provide the complete backwards traceability from a set of paragraphs in textual form, as high-light in a history tree, and as a separate history subtree. Two modes of traceability should be supported:

- backward traceability through the same history tree as the selected paragraph
- backward traceability through all history trees in the current project by using the "origin" parameter

05-06: Paragraph forwards traceability

The tool must provide the complete forwards traceability from a set of paragraphs in textual form, as high-light in a history tree, and as a separate history subtree. Two modes of traceability should be supported:

- forward traceability through the same history tree as the selected paragraph
- forward traceability through all history trees in the current project by using the "origin" parameter

7.1.6 Documentation specifications

06-01: Document generation

The tool shall provide functionality to generate documents based on current paragraphs.

06-02: Traceability table and list generation

The tool shall provide functionality to generate standard traceability tables and lists showing relationships between paragraphs.

7.1.7 Authentication specifications

07-01: Tool access

Initially, the tool will provide some security in the form of a password login procedure. The tool will be a stand-alone application with direct access to project files.

In the future, a server / client architecture should be implemented where the client can have limited access to project data after completing a login procedure.

7.1.8 Project specifications

08-01: Templates

The tool shall provide different templates for different project types, e.g. for software development a template should include development phase sets and dependability factor sets, while for procedure maintenance it should include procedure sets.

08-02: Project rules

1. It should not be possible to create a new Project with the same name as an existing Project.

7.1.9 History Tree specifications

09-01: History Tree creation

The tool must support the creation of different history trees. Each history tree may contain any number of paragraphs and their associated changes.

09-02: History Tree rules

1. It should not be possible to create a new History Tree with the same name as an existing History Tree.

7.2 Design Specifications

This section describes the design specification for the prototype of the tool TRACE.

7.2.1 Class specifications

01-01: Paragraph class

The Paragraph class contains the definition of a paragraph, and is uniquely identified by a label and version number (internally an ID may be used). The Paragraph may be the result of *one* Change, and may be subject to *one or more* Changes.

Paragraph	
Parameter	Type
ID	String
label	String
version	int
time	long
status	int
open	boolean
description	String
changeIn	Change
changesOut	array of Change
origin	array of Link
originOf	array of Paragraph

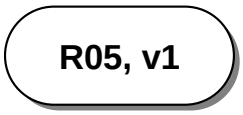
The *status* variable is used to set various status values for a Paragraph:

None Default Paragraph status.

Created	Indicates that the Paragraph is the first in a list of Paragraphs with the same label, but different version numbers. The Paragraph is the result of either a *create* change or a change performed on another Paragraph which creates one or more new Paragraph(s) (*derive, split, combine*...).
Trace	The Paragraph is part of a trace result, e.g. a backward trace. The Paragraph will be highlighted in the history tree display.
Highlight	The Paragraph is highlighted in the history tree display.
Deleted	The Paragraph has been explicitly deleted (having been subject to the *delete* change).

A newly created Paragraph could be displayed like this (with label and version):

A Paragraph that has been subject to some change could be displayed like this:



A Paragraph that has been subject to a “delete” change could be displayed like this:

Right-clicking on a Paragraph should display a context-sensitive menu with options, e.g. which change types are available, properties dialog...

After being subject to a Change, a Paragraph should be *locked*, i.e. no more changes should be made to it (the *open* attribute is set to *false*). It should be possible to introduce *minor changes* to the description text of a Paragraph without introducing a new paragraph (by using a Change). In this case, a user should be able to manually *un-lock* a Paragraph, make the necessary changes, and then lock the Paragraph. Any outgoing Changes from the Paragraph should be marked as *not validated* (see description of Changes below).

01-02: ChangeType class

The ChangeType class is used to define different types of Changes. Each ChangeType is identified by a unique label.

ChangeType	
Parameter	Type
label	String
paragraphsIn	int
paragraphsOut	int

description	String
resultStatus	int
update	int

The *paragraphsIn* and *paragraphsOut* parameters limits the number of Paragraphs used as input and output to the Change, and each should have one of the following values:

- 0
- 1
- 1 or more
- 2 or more

The resultStatus determines the status of the Paragraph resulting from the Change, and should have one of the following values:

None	This applies to most Changes.
Created	For the *create* Change, when a Paragraph is created from scratch, or for Changes performed on other Paragraph(s) which creates one or more new Paragraph(s) (*derive, split, combine*...).
Deleted	For the *delete* Change.

The *update* values defines how the Paragraph label and version number are determined for a Paragraph resulting from a Change:

No update	The output Paragraph has the same label and version number as the input Paragraph.
New label	The output Paragraph is given a new label.
Increment version number	The version number of the output Paragraph is incremented relative to the input Paragraph.

For use in software development, the Change types may include, but not be limited to:

- create
- modify
- combine
- replace
- split
- derive
- delete
- un-delete

A separate menu item will allow the user to introduce new Change types.

01-03: Change class

The Change class contains the properties of a single Change from one or more Paragraphs into one or more Paragraphs.

Change	
Parameter	Type
ID	String
type	ChangeType
sources	array of Paragraph
targets	array of Paragraph
status	int
userID	String
time	long
reason	String
basis	int
validated	boolean
validateUserID	String

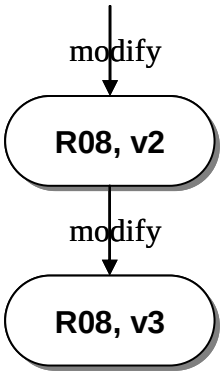
The *status* parameter is used to mark the Change as a result from various analyses, e.g. a backward trace.

Trace	The Change is part of a trace result, e.g. a backward trace. The Change will be highlighted in the history tree display.
Highlight	The Change is highlighted in the history tree display.

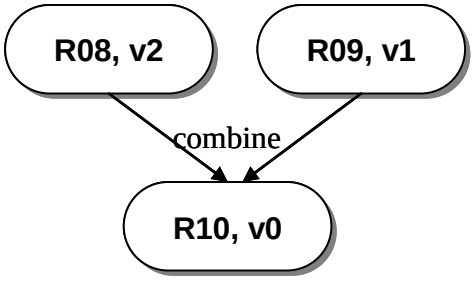
The *basis* parameter is used to provide some description of the basis for applying the Change to one or more Paragraphs.

None	No special basis is given for the Change.
Method	The Change has been applied due to the outcome of some analysis method, e.g. a HazOp analysis, which has suggested that the Paragraph(s) must be updated due to some shortcoming.
Expert	The Change has been applied due to input from some expert (expert judgement).

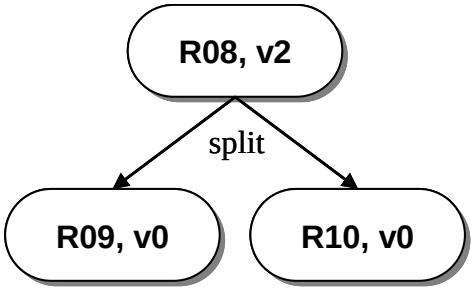
A Change (e.g.”modify”) of a single Paragraph into another Paragraph could be displayed like this:



A Change (e.g.”combine”) of two Paragraphs into another Paragraph could be displayed like this:



A Change (e.g.”split”) of a single Paragraph into two other Paragraphs could be displayed like this:



A list of all Changes in a history tree should be displayed next to the history tree, so the user can choose and edit the Change properties.

A Change can be manually validated by a user, whereupon the *validate* attribute is set to true, and the *validateUserID* holds the ID of the user. A Change should be automatically marked as *not validated* when minor changes are made to the source Paragraph(s) (see above).

01-04: HistoryTree class

The HistoryTree class is used to hold all required information about one history tree, including all Paragraphs and Changes. An example of a history tree is shown in Figure 7.

HistoryTree	
Parameter	Type
ID	String
label	String
paragraphs	array of Paragraph
changes	array of Change
createTime	long
lastChangeTime	long

01-05: Set class

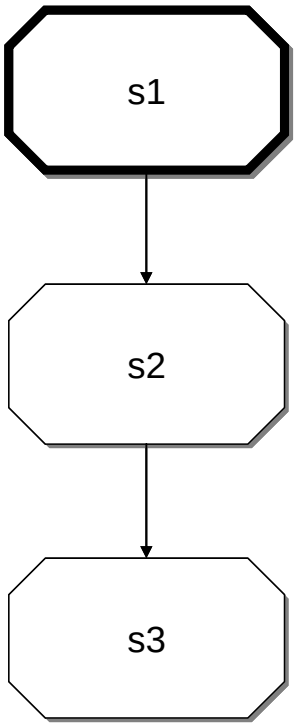
The Set class extends the HistoryTree class to include a list of Sets, links to parent and child sets, and information about opening and closing times and status. This allows a Set to contain

any number of Paragraph objects, as well as any number of Set objects, and to maintain a derivative relationship between Sets.

A Set will be able to compare its content (list of Paragraphs) to the content of another Set, i.e. which Paragraphs are common to both Sets, and which Paragraphs are unique. This will be achieved by an iteration through the list of Paragraphs in each Set and compare Paragraph IDs.

Set extends HistoryTree	
Parameter	Type
trees	array of HistoryTree
sets	array of Set
parentSet	Set
childSet	Set
open	boolean
closeTime	long

A closed Set and its open derivatives could be displayed like this:



01-06: Link class

A Link is used to provide different sources of information to the *origin* parameter of a Paragraph (there may be other uses as well). The origin of a Paragraph could be a combination of textual descriptions, files, hypertext links, other Paragraphs or Sets.

Link	
Parameter	Type

id	String
type	int
string	String

The *type* values may include:

- TEXT = 0
- FILE = 1
- HYPERTEXT = 2
- PARAGRAPH = 3
- SET = 4

Examples of link types:

A textual link

`object Link`
`  type: TEXT`
`  string: "This Paragraph was included due to a discussion at project meeting 2005-04-08"`
`end`

A file link

`object Link`
`  type: FILE`
`  string: "p08-basis.doc"`
`end`

A hypertext link

`object Link`
`  type: HYPERTEXT`
`  string: "http://standards.ieee.org/catalog/olis/index.html"`
`end`

A Paragraph link

`object Link`
`  type: PARAGRAPH`
`  string: "P738961763" (the ID of a particular Paragraph)`
`end`

A Set link

`object Link`
`  type: SET`
`  string: "S168826738" (the ID of a particular Set)`
`end`

01-07: TimeStamp class

The TimeStamp class is a helper class to store the time of various events with an accuracy of one second.

TimeStamp	
Parameter	Type
year	integer
month	integer
day	integer
hour	integer
minute	integer
second	integer

PS! This class will probably not be needed, as all time stamps (with an accuracy of 1 milli-second) can be stored a long integers, and dedicated routines are available to extract year, month, day, hour, minute and second values.

01-08: Project class

The project class extends the Set class and contains information for an entire project, i.e. a list of history trees and sets.

Project	
Parameter	Type
name	String
description	String
links	array of Link

7.2.2 Analysis specifications

This chapter will describe the analyses available, and descriptions of how they can be performed using the class definitions above.

The example history tree used is given in Figure 7 below.

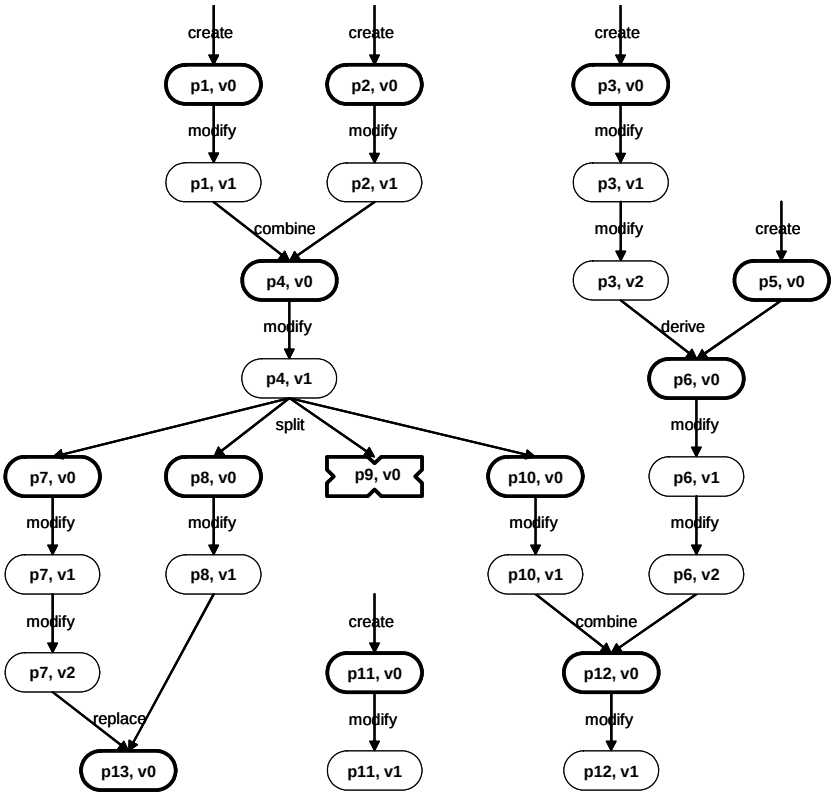


Figure 7. ***Example history tree.***

The temporal development of the history tree can be given in the form of a change table:

Change	Source paragraph(s)	Target paragraph(s)
create		(p1, v0)
create		(p2, v0)
create		(p3, v0)
modify	(p1, v0)	(p1, v1)
modify	(p2, v0)	(p2, v1)
modify	(p3, v0)	(p3, v1)
combine	(p1, v1), (p2, v1)	(p4, v0)
modify	(p3, v1)	(p3, v2)
create		(p5, v0)
modify	(p4, v0)	(p4, v1)
derive	(p3, v2), (p5, v0)	(p6, v0)
split	(p4, v1)	(p7, v0), (p8, v0), (p9, v0), (p10, v0)
modify	(p6, v0)	(p6, v1)
modify	(p7, v0)	(p7, v1)
delete	(p9, v0)	
modify	(p10, v0)	(p10, v1)
modify	(p6, v1)	(p6, v2)
modify	(p7, v1)	(p7, v2)
modify	(p8, v0)	(p8, v1)
create		(p11, v0)
combine	(p10, v1), (p6, v2)	(p12, v0)
replace	(p7, v2), (p8, v1)	(p13, v0)
modify	(p11, v0)	(p11, v1)
modify	(p12, v0)	(p12, v1)

02-01: Created paragraphs list

Ref.: REQ-SPEC 05-01

Whenever a new paragraph is created, either “from scratch” or by certain changes to other paragraphs (e.g. derive, split, combine...), the STATUS_CREATED bit in the paragraphs status variable is set to 1 (by default this bit is 0).

To generate a list of created paragraphs, the tool will iterate through the list of paragraphs within current history tree and display (textually or graphically) the paragraphs with the STATUS_CREATED bit set.

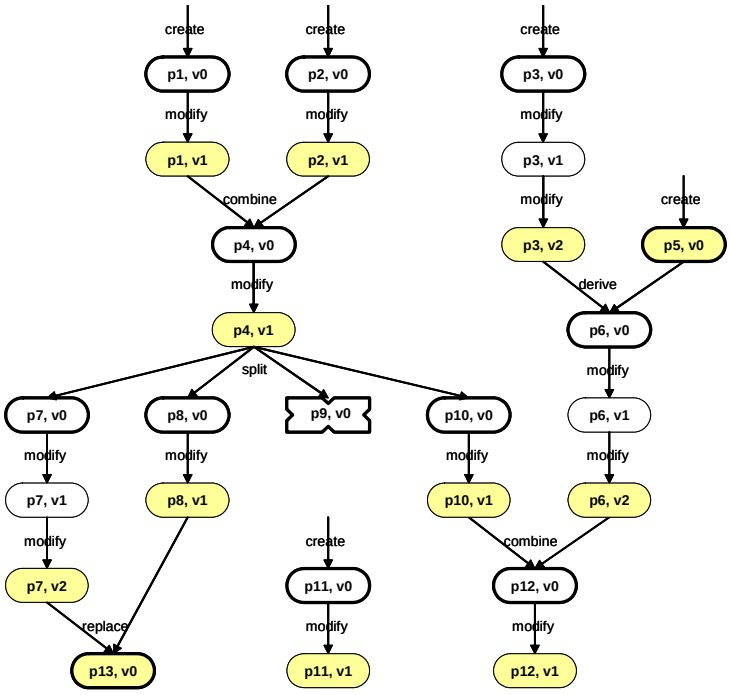


Figure 8. ***Current paragraphs.***

In the example the created paragraphs have bold outline.

02-02: Current paragraphs list

Ref.: REQ-SPEC 05-02

The current, or most recently updated version of a paragraph, is found by iterating through the list of paragraphs and for each paragraph label find the paragraph with the highest version number. Paragraphs that have been explicitly deleted are not included in this search.

In the example case, the current paragraphs would be: (p1,v1), (p2,v1), (p3,v2), (p4,v1), (p5,v0), (p6,v2), (p7,v2), (p8,v1), (p10,v1), (p11,v1), (p12,v1) and (p13,v0).

02-03: Deleted paragraphs list

Ref.: REQ-SPEC 05-03

Whenever a paragraph is deleted, the STATUS_DELETED bit in the paragraphs status variable is set to 1 (by default this bit is 0).

To generate a list of deleted paragraphs, the tool will iterate through the list of paragraphs within current history tree and display (textually or graphically) the paragraphs with the STATUS_DELETED bit set.

In the example case, the paragraph (p9, v0) has been deleted.

02-04: Paragraph history

Ref.: REQ-SPEC 05-04

The paragraph history for any paragraph can be determined by find all versions of the selected paragraph, all changes affecting these versions, as well as the relevant version of all paragraphs included in these changes. This is straightforward, as all *paragraph objects* contain lists of “incoming” and “outgoing” changes, and all *change objects* contain lists of “input” and “output” paragraphs.

In the example case, the backward search would be:

- The incoming change to (p6, v1) is a *modify* change, whose input paragraph is (p6, v0).
- The incoming change to (p6, v0) is a *derive* change, whose input paragraphs are (p5, v0) and (p3, v2).

At this point the backward search is halted, since (p6, v0) is the first version of paragraph p6.

The forward search would be:

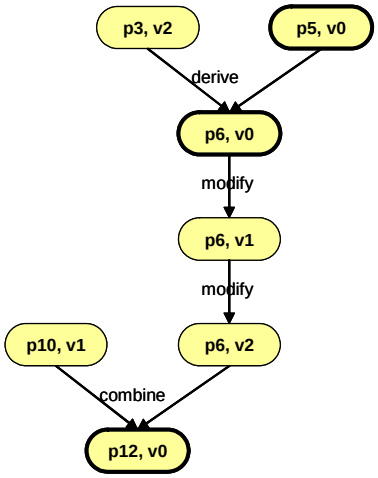


Figure 9. ***Paragraph history of (p6, v1).***

- The outgoing change from (p6, v1) is a *modify* change, whose output paragraph is (p6, v2).
- The outgoing change from (p6, v2) is a *combine* change, whose output paragraph is (p12, v0) and other input paragraph is (p10, v1).

At this point the forward search is halted, since (p6, v2) is the last version of paragraph p6.

The paragraph history for paragraph (p6, v1) would thus include (p3, v2), (p5, v0), (p6, v0), (p6, v1), (p6, v2), (p10, v1) and (p12, v0) (see Figure 9). The changes involved are given in the following table:

Change	Source paragraph(s)	Target paragraph(s)
derive	(p3, v2), (p5, v0)	(p6, v0)
modify	(p6, v0)	(p6, v1)
modify	(p6, v1)	(p6, v2)
combine	(p10, v1), (p6, v2)	(p12, v0)

02-05: Paragraph backward traceability

Ref.: REQ-SPEC 05-05

Given a paragraph, we want to find the development of paragraphs that leads to this paragraph, i.e. the minimum fragment of the change history that has influenced the development of the given paragraph.

The trace can be performed by a recursive search through all input changes starting with the selected paragraph. The search through a sub-tree is halted once a paragraph whose input is a "create" change is reached.

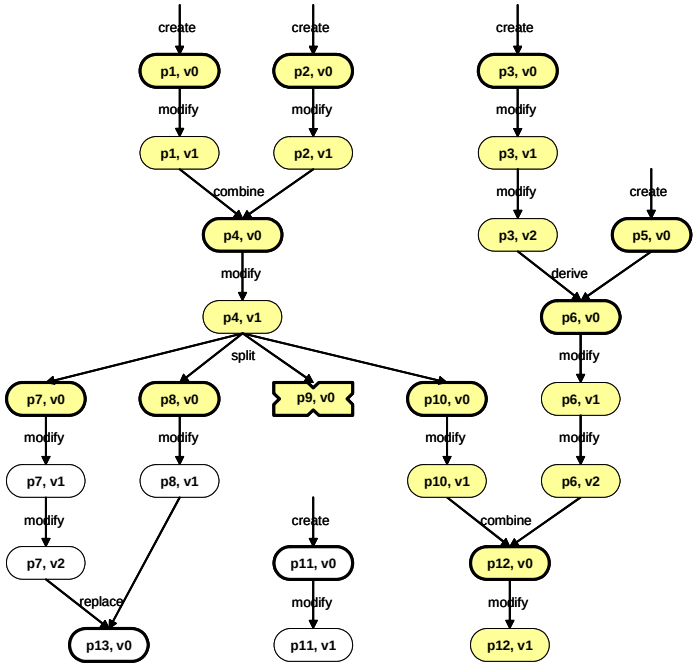


Figure 10. ***Backward traceability from (p12, v1)***

Figure 10 shows the results of a backward trace on paragraph (p12, v1). In this case there is a *modify* change as input, whose source is (p12, v0). This paragraph is the result of a combine change with (p6, v2) and (p10, v1) as inputs. Continuing the search through several sub-trees, we finally reach the paragraphs (p1, v0), (p2, v0), (p3, v0) and (p5, v0), all results of *create* changes.

Note that the paragraphs (p7, v0), (p8, v0) and (p9, v0) are included in the trace result, as they are involved in the *split* change of (p4, v1).

The changes involved are given in the following table:

Change	Source paragraph(s)	Target paragraph(s)
create		(p1, v0)
create		(p2, v0)
create		(p3, v0)
modify	(p1, v0)	(p1, v1)
modify	(p2, v0)	(p2, v1)
modify	(p3, v0)	(p3, v1)
combine	(p1, v1), (p2, v1)	(p4, v0)
modify	(p3, v1)	(p3, v2)
create		(p5, v0)
modify	(p4, v0)	(p4, v1)
derive	(p3, v2), (p5, v0)	(p6, v0)
split	(p4, v1)	(p7, v0), (p8, v0), (p9, v0), (p10, v0)
modify	(p6, v0)	(p6, v1)
modify	(p10, v0)	(p10, v1)
modify	(p6, v1)	(p6, v2)
combine	(p10, v1), (p6, v2)	(p12, v0)
Modify	(p12, v0)	(p12, v1)

02-06: Paragraph forward traceability

Ref.: REQ-SPEC 05-06

Forwards traceability relates to the development of paragraphs starting with a selected paragraph. The result will include all paragraphs affected by the selected paragraph.

The trace can be performed by a recursive search through all output changes starting with the selected paragraph. The search through a sub-tree is halted once a paragraph without any output changes is reached.

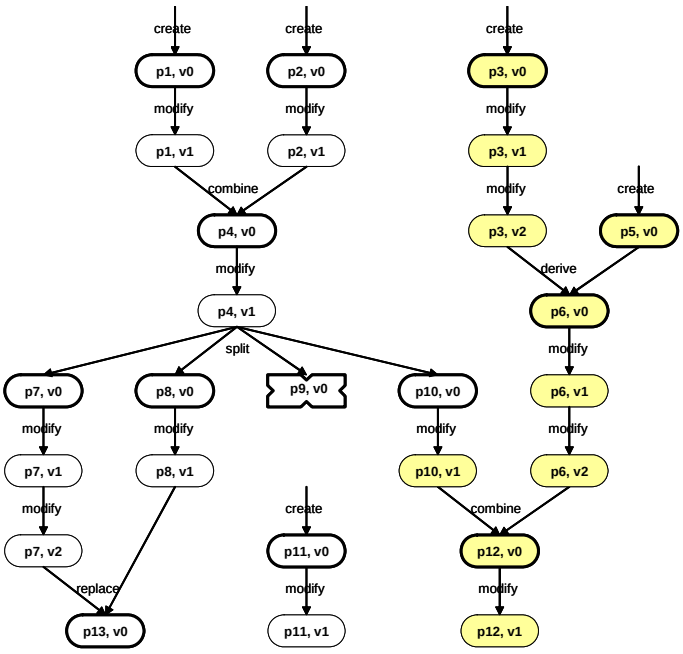


Figure 11. ***Forward traceability from (p3, v0)***

Figure 11 shows the results of a forward trace on paragraph (p3, v0). In this case there is a *modify* change as output, whose target is (p3, v1). Continuing the search, we finally reach the paragraphs (p12, v1), which has no output changes.

Note that the paragraphs (p5, v0) and (p10, v0) are included in the trace result, as they are involved in the *derive* and *combine* changes of (p3, v2) and (p6, v2), respectively.

The changes involved are given in the following table:

Change	Source paragraph(s)	Target paragraph(s)
create		(p3, v0)
modify	(p3, v0)	(p3, v1)
modify	(p3, v1)	(p3, v2)
create		(p5, v0)
derive	(p3, v2), (p5, v0)	(p6, v0)
modify	(p6, v0)	(p6, v1)
modify	(p6, v1)	(p6, v2)
combine	(p10, v1), (p6, v2)	(p12, v0)
modify	(p12, v0)	(p12, v1)

02-07: Origin traceability

Ref.: REQ-SPEC 05-05 + 05-06

The *origin* parameter in the Paragraph class provides links to information used when creating a paragraph. This information could be a textual description of why the paragraph should be included, a shortcut to a file, a hypertext link to an IEEE standard used as basis for the paragraph, or a link to another paragraph in a different history tree. A typical use of the origin parameter could be during a software development project, where a separate history tree is created for each development phase (requirement, design, implementation, test...). Here, each paragraph would represent a specific version of a specification, and often a specification in the design phase would be based on a specification in the requirement phase.

This situation is shown in Figure 12 below. Here we have included 3 developments phases: requirement, design and implementation. The origin of paragraph (or specification) (p1, v0) in the design phase is paragraph (p2, v1) in the requirement phase. The origin of paragraph (p3, v0) in the implementation phase is paragraph (p1, v1) in the design phase.

Right-clicking on a selected Paragraph will provide a list of origins for the Paragraph, and the possibility to go to a selected origin. If the origin is

- another Paragraph in another history tree, the history tree will be displayed and the Paragraph highlighted
- a file link, the file will be opened
- a hypertext link, a webpage based on the link will be opened

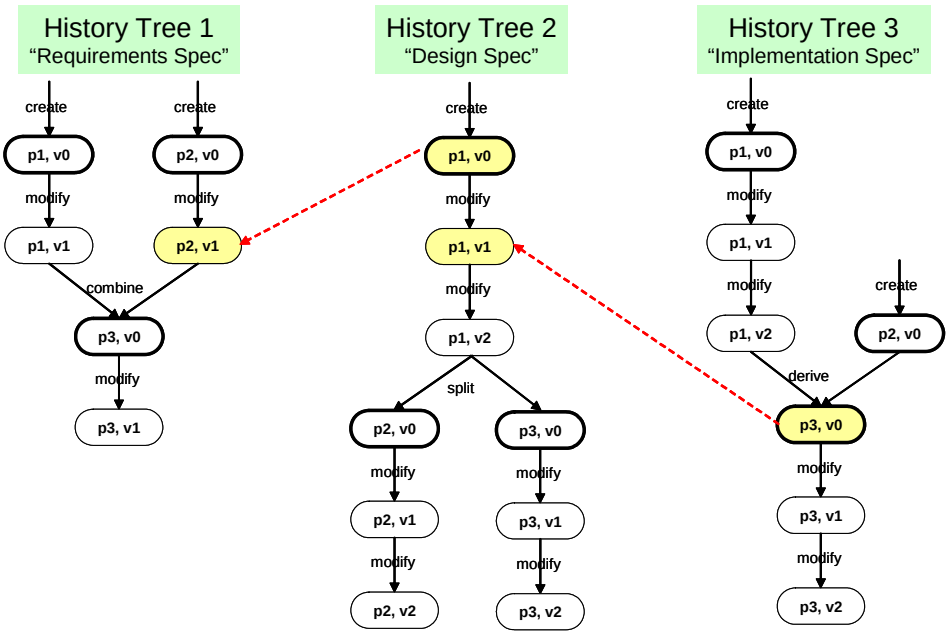


Figure 12. ***Origin trace***

03-01: Authentication

Ref.: REQ-SPEC 07-01

The tool will provide a login procedure (see Figure 13). Passwords will be encrypted and stored in a separate file.

Figure 13. ***Login dialog box.***

7.3 Implementation Specifications

This section describes the implementation specification for the prototype of the tool TRACE.

7.3.1 Language specifications

01-01: Programming language

The prototype will be written in Java 1.5.

01-02: Data storage

Application data and configuration information will be stored in XML formats specified below.

7.3.2 Class specifications

02-01: Paragraph class

Paragraph		
Parameter	**Type**	**Description**
ID	String	Unique ID
label	String	Textual label
version	int	Paragraph version
time	long	Creation time (given as milliseconds since January 1, 1970)
status	int	Paragraph status (see table below)
open	boolean	*True* if the Paragraph can be modified, otherwise *false*
description	String	Textual description of Paragraph, which for e.g. SW development would contain the requirements text
changeIn	Change	Change which created the Paragraph
changesOut	Vector<**Change**>	List of output Changes from the Paragraph
origins	Vector<**Link**>	List of origins for Paragraph
originOf	Vector<**Paragraph**>	List of Paragraphs for which this Paragraph is the origin
ownerTree	HistoryTree	The history tree to which this Paragraph belongs
posX	int	Horizontal position in history tree display
posY	int	Vertical position in history tree display

selected	boolean		*True* if Paragraph is selected in history tree view, otherwise *false*
Method	**Input**	**Return**	**Description**
getID		String	Returns the *ID* parameter.
setID	String		Sets the *ID* parameter.
getLabel		String	Returns the *label* parameter.
setLabel	String		Sets the *label* parameter.
getVersion		int	Returns the *version* parameter.
setVersion	int		Sets the *version* parameter.
incVersion			Increases the *version* parameter.
getTime		long	Returns the *time* parameter.
setTime	long		Sets the *time* parameter.
getStatus		int	Returns the *status* parameter.
setStatus	int		Sets the *status* parameter.
isStatusBitSet	boolean	int	Returns *true* if given status bit is set (see status constants in table below)
setStatusBit		int	Sets a bit in the status parameter
clearStatusBit	int		Clears a bit in the status parameter
close			Sets the *open* attribute to *false*
isOpen		boolean	Returns the *open* attribute
getOpen		boolean	Returns the *open* attribute
setOpen	boolean		Sets the *open* attribute to the input value
setOpen	String		Sets the *open* attribute to the input String value, given as “open” or “false”
getDescription		String	Returns *description* parameter
setDescription	String		Sets *description* parameter
getChangeIn		**Change**	Returns the input Change
setChangeIn	**Change**		Sets the input Change
getChangesOut		Vector<**Change**>	Returns the list of output Changes
setChangesOut	Vector<**Change**>		Sets the list of output Changes
addChangeOut	**Change**		Adds an output Change
getOrigins		Vector<**Link**>	Returns the list of origins
setOrigins	Vector<**Link**>		Sets the list of origins
addOrigin	**Link**		Adds an origin
getOriginOf		Vector<**Paragraph**>	Returns the list of Paragraphs for which this Paragraph is the origin
setOriginOf	Vector<**Paragraph**>		Sets the list of Paragraphs for which this Paragraph is the origin
addOriginOf	**Paragraph**		Adds to the list of Paragraphs for which this Paragraph is the origin
getOwnerTree		**HistoryTree**	Returns the history tree to which this Paragraph belongs
setOwnerTree	**HistoryTree**		Sets the history tree to which this Paragraph belongs
getPosX		int	Returns X position in history tree view

setPosX	int		Sets X position in history tree view
getPosY		int	Returns Y position in history tree view
setPosY	int		Sets Y position in history tree view
setXY	int, int		Sets X and Y position in history tree view
isSelected		boolean	Returns *true* if Paragraph is selected in display, otherwise *false*
getSelected		boolean	Same as isSelected
setSelected	boolean		Sets the *selected* parameter

The *status* parameter may be assigned any combination of the following constant values:

Constant	Value	Description
STATUS_NONE	0x0000	Default status value of a Paragraph, meaning no bit field has been set.
STATUS_CREATED	0x0001	When this bit is set, it indicates that the Paragraph is the first in a list of Paragraphs with the same label, but different version numbers. The Paragraph is the result of either a *create* change or a change performed on another Paragraph which creates one or more new Paragraph(s) (*derive, split, combine*...).
STATUS_TRACE	0x0002	This status bit used to mark a Paragraph as part of a trace result, e.g. a backward trace. When this bit is set, the Paragraph will be highlighted in the history tree display.
STATUS_HIGHLIGHT	0x0004	This status bit is used highlight the Paragraph in the history tree display.
STATUS_DELETED	0x0008	This status bit is used to indicate that the Paragraph has been explicitly deleted (having been subject to the *delete* change).

02-02: ChangeType class

ChangeType			
Parameter	**Type**		**Description**
label	String		Textual label
paraIn	int		Number of input Paragraphs
paraOut	int		Number of output Paragraphs
description	String		Textual description of Change, including reason for introducing it
resultStatus	int		The status of the Paragraph created due to this Change
update	int		Indicates how to update the label and version number of a new Paragraph (see below)
locked	boolean		*True* if the ChangeType is locked for editing (modification), *false* otherwise
Method	**Input**	**Return**	**Description**
getLabel		String	Returns the *label* parameter.

setLabel	String		Sets the *label* parameter.
getParaIn		int	Returns the *paraIn* parameter.
setParaIn	int		Sets the *paraIn* parameter.
getParaOut		int	Returns the *paraOut* parameter.
setParaOut	int		Sets the *paraOut* parameter.
getDescription		String	Returns the *description* parameter.
setDescription	String		Sets the *description* parameter.
getResultStatus		int	Returns the *resultStatus* parameter.
setResultStatus	int		Sets the *resultStatus* parameter.
getUpdate		int	Returns the *update* parameter.
setUpdate	int		Sets the *update* parameter.
isLocked	boolean		Returns the *locked* parameter, which determines whether ChangeType is editable.
getLocked	boolean		Same as isLocked.
setLocked		boolean	Sets the *locked* parameter, which determines whether ChangeType is editable.

The *paragraphsIn* and *paragraphsOut* parameters may be assigned one of the following constant values:

Constant	Value	Description
NUM_0	0	Zero
NUM_1	1	One
NUM_1PLUS	2	One or more
NUM_2PLUS	3	Two or more

The *resultStatus* parameter may be assigned one of the following constant values (see *status* parameter of Paragraph class):

Constant	Value	Description
STATUS_NONE	0x0000	For Changes that do not delete or create new Paragraphs
STATUS_CREATED	0x0001	For all Changes that create new Paragraphs, i.e. with new label
STATUS_DELETED	0x0008	For “delete” change

The *update* parameter may be assigned one of the following constant values:

Constant	Value	Description
UPDATE_NOCHANGE	0	The label and version number of the new Paragraph is the same as for the source Paragraph
UPDATE_NEWLABEL	1	The new Paragraph shall be given a new label, with version number 0
UPDATE_INCVERSION	2	The new Paragraph shall keep the same label as the source Paragraph, but with incremented version number

For use in software development, the change types may include, but not be limited to:

label	para...In	para...Out	description	resultStatus	update
“create”	NUM_0	NUM_1	“This change causes a creation of a	STATUS_CREATED	UPDATE_NEWLABEL

			paragraph”		
“modify”	NUM_1	NUM_1	“This change causes a modification of a paragraph”	STATUS_NONE	UPDATE_INCVERSION
“combine”	NUM_2PLUS	NUM_1	“This change causes a combination of 2 or more paragraphs”	STATUS_CREATED	UPDATE_NEWLABEL
“replace”	NUM_1PLUS	NUM_1	“This change causes a replacement of one or more paragraphs”	STATUS_CREATED	UPDATE_NEWLABEL
“split”	NUM_1	NUM_2PLUS	“This change causes a split of one paragraph into 2 or more paragraphs”	STATUS_CREATED	UPDATE_NEWLABEL
“derive”	NUM_1PLUS	NUM_1	“This change causes a derivation of 1 or more paragraphs into 1 paragraph”	STATUS_CREATED	UPDATE_NEWLABEL
“delete”	NUM_1	NUM_0	“This change causes a deletion of 1 paragraph”	STATUS_DELETED	UPDATE_NOCHANGE
“un-delete”	NUM_1	NUM_1	“This change causes an un-deletion of 1 paragraph”	STATUS_NONE	UPDATE_NOCHANGE

02-03: Change class

Change			
Parameter	**Type**		**Description**
ID	String		Unique ID of Change
type	**ChangeType**		The type of the Change (see ChangeType)
sources	Vector<**Paragraph**>		Input Paragraphs
targets	Vector<**Paragraph**>		Output Paragraphs resulting from the Change
status	int		See table below
userID	String		ID of user introducing the Change
time	long		Time the Change was introduced
reason	String		Textual description of the reason for introducing the Change
basis	int		The basis for introducing the Change (see table below)
validated	boolean		True if the Change has been manually validated by a user, false otherwise
validateUserID	String		ID of the user who has validated the Change
Method	**Input**	**Return**	**Description**
getID		String	Returns the *ID* parameter.
setID	String		Sets the *ID* parameter.
getType		**ChangeType**	Returns the *type* parameter.
setType	**ChangeType**		Sets the *type* parameter.
getSources		Vector<**Paragraph**>	Returns the list of source Paragraphs.
setSources	Vector<**Paragraph**>		Sets the list of source Paragraphs.

addSource	**Paragraph**		Adds a Paragraph to the list of source Paragraphs.
getTargets		Vector<**Paragraph**>	Returns the list of target Paragraphs.
setTargets	Vector<**Paragraph**>		Sets the list of target Paragraphs.
addTarget	**Paragraph**		Adds a Paragraph to the list of target Paragraphs.
getStatus		int	Returns the *status* parameter.
setStatus	int		Sets the *status* parameter.
isStatusBitSet	boolean	int	Returns *true* if given status bit is set (see status constants in table below)
setStatusBit		int	Sets a bit in the status parameter
clearStatusBit	int		Clears a bit in the status parameter
getUserID		String	Returns the user ID of the user who introduced the Change.
setUserID	String		Sets the user ID of the user who introduced the Change.
getTime		long	Returns the *time* parameter.
setTime	long		Sets the *time* parameter.
getReason		String	Returns the *reason* parameter.
setReason	String		Sets the *reason* parameter.
getBasis		int	Returns the *basis* parameter.
setBasis	int		Sets the *basis* parameter.
isValidated		boolean	Returns the *validate* parameter.
getValidated		boolean	Returns the *validate* parameter.
setValidated	boolean		Sets the *validate* parameter.
setValidated	String		Sets the *validate* parameter using the string “true” or “false”.
validate			Sets the *validate* parameter to *true*.
invalidate			Sets the *validate* parameter to *false*.
getValidateUserID		String	Returns the user ID of the user who validated the Change.
setValidateUserID	String		Sets the user ID of the user who validated the Change.

The *basis* parameter may be assigned the following constant values:

Constant	**Value**	
BASIS_NONE	0	No basis given for introducing Change
BASIS_METHOD	1	The Change was introduced due to some method output
BASIS_EXPERT	2	The Change was introduced due to expert judgement

The *status* parameter may be assigned one of the following constant values (see *status* parameter of Paragraph class):

Constant	Value	Description
STATUS_TRACE	0x0002	This status bit used to mark a Change as part of a trace result, e.g. a backward or forward trace. When this bit is set, the Change will be highlighted in the history tree display.
STATUS_HIGHLIGHT	0x0004	This status bit is used highlight the Change in the history tree display.

02-04: HistoryTree class

The HistoryTree class is used to hold all required information about one history tree, including all Paragraphs and Changes.

HistoryTree			
Parameter	**Type**		**Description**
ID	String		Unique ID
label	String		Textual label
paragraphs	Vector<**Paragraph**>		List of Paragraphs in HistoryTree
changes	Vector<**Change**>		List of Changes in HistoryTree
createTime	long		Creation time of HistoryTree given as milliseconds from Jan 1 1970.
lastChangeTime	long		Time of last change to HistoryTree.
Method	**Input**	**Return**	**Description**
getID		String	Returns the *ID* parameter.
setID	String		Sets the *ID* parameter.
getLabel		String	Returns the *label* parameter.
setLabel	String		Sets the *label* parameter.
getParagraphs		Vector<**Paragraph**>	Returns the list of Paragraphs.
setParagraphs	Vector<**Paragraph**>		Sets the list of Paragraphs.
addParagraph	**Paragraph**		Adds a Paragraph to the list of Paragraphs.
removeParagraph	**Paragraph**		Removes a Paragraph from the list of Paragraphs.
getChanges		Vector<**Change**>	Returns the list of Changes.
setChanges	Vector<**Change**>		Sets the list of Changes.
addChange	**Change**		Adds a Change to the list of Changes.
getCreateTime		long	Returns the *createTime* parameter.
setCreateTime	long		Sets the *createTime* parameter.
getLastChangeTime		long	Returns the *lastChangeTime*

			parameter.
setLastChangeTime	long		Sets the *lastChangeTime* parameter.
setLastChangeTime			Sets the *lastChangeTime* parameter to current millisecond.
paragraphExists	String		Returns *true* if a paragraph with same label exists in the tree.

02-05: Set class

The Set class extends the HistoryTree class to include a list of Sets, links to parent and child sets, and information about opening and closing times and status. This allows a Set to contain any number of Paragraph objects, as well as any number of Set objects, and to maintain a derivative relationship between Sets.

A Set will be able to compare its content (list of Paragraphs) to the content of another Set, i.e. which Paragraphs are common to both Sets, and which Paragraphs are unique. This will be achieved by an iteration through the list of Paragraphs in each Set and compare Paragraph labels and versions.

Set			
Parameter	**Type**		**Description**
trees	Vector<**HistoryTree**>		The list of HistoryTree objects contained by the Set.
sets	Vector<**Set**>		The list of Set objects contained by the Set.
parentSet	**Set**		The *parent* Set.
childSet	**Set**		The *child* Set.
open	boolean		*True* if the content of the Set can be modified, *false* otherwise.
closeTime	long		The time the Set was closed (0 if Set is still open)
posX	int		Horizontal position in set display
posY	int		Vertical position in set display
selected	boolean		*True* is Set is selected in Set view, otherwise *false*
Method	**Input**	**Return**	**Description**
getTrees		Vector<**HistoryTree**>	Returns the list of HistoryTrees in Set
setTrees	Vector<**HistoryTree**>		Sets the list of HistoryTrees in Set
addTree	**HistoryTree**		Adds a HistoryTree to the list of HistoryTrees
removeTree	**HistoryTree**		Removes a HistoryTree from

			the list of HistoryTrees
getSets		Vector<**Set**>	Returns the list of Sets in Set
setSets	Vector<**Set**>		Sets the list of Sets in Set
addSet	**Set**		Adds a Set to the list of Sets
removeSet	**Set**		Removes a Set from the list of Sets
getParentSet		**Set**	Returns the parent Set
setParentSet	**Set**		Sets the parent Set
getChildSet		**Set**	Returns the child Set
setChildSet	**Set**		Sets the child Set
getCloseTime		long	Returns the time the Set was closed
setCloseTime	long		Sets the time the Set was closed
close			Closes the Set, and sets the close time to the current milli-second
isOpen		boolean	Returns the *open* parameter
getOpen		boolean	Same as isOpen
setOpen	boolean		Sets the *open* parameter
setOpen	String		Sets the *open* parameter using String input "true" or "false"
getPosX		int	Returns X position in set view
setPosX	int		Sets X position in set view
getPosY		int	Returns Y position in set view
setPosY	int		Sets Y position in set view
setXY	int, int		Sets X and Y position in set view
isSelected		boolean	Returns the *selected* parameter
getSelected		boolean	Same as isSelected
setSelected	boolean		Sets the *selected* parameter
treeExists		boolean	Returns true if a tree with same label exists in set.
setExists		boolean	Returns true if a set with same label exists in set.

02-06: Link class

A Link is used to provide different sources of information to the *origin* parameter of a Paragraph (there may be other uses as well). The origin of a Paragraph could be a combination of textual descriptions, files, hypertext links, other Paragraphs or Sets.

Link			
Parameter	**Type**		**Description**
ID	String		Unique ID
type	int		Type of link (see table below)
string	String		Textual information on Link
Method	**Input**	**Return**	**Description**

getID		String	Returns ID
setID	String		Sets ID
getType		int	Returns type
setType	int		Sets type
getString		String	Returns string
setString	String		Sets string

The *linkType* parameter may be assigned the following constant values:

Constant	Value	Description
LINK_TEXT	0	The link is text
LINK_FILE	1	The link is the name (with path) of a file, e.g. “c:\code\test.java”
LINK_HYPERTEXT	2	The link is a hypertext string, e.g. “www.ife.no”
LINK_PARAGRAPH	3	The link is the ID of a Paragraph, e.g. “PA_003782”
LINK_SET	4	The link is the ID of a Set, e.g. “SE_000218”

02-07: Project class

Project			
Parameter	**Type**		**Description**
name	String		Project name
description	String		Project description
links	Vector<**Link**>		List of origin links in Project
Method	**Input**	**Return**	**Description**
getName		String	Returns name
setName	String		Sets name
getDescription		String	Returns description
setDescription	String		Sets description
getLinks		Vec-tor<**Link**>	Returns list of links
setLinks	Vec-tor<**Link**>		Sets list of links

7.3.3 Menu specifications

03-01: File menu

Menu item	Description
New Project...	Create a new project.
Open Project...	Open an existing project for editing.
Save Project	Save a project to file.
Project Properties...	Dialog box to set properties for current project.
Quit	Quit application.

03-02: Analysis menu

Menu item	Description
Paragraphs	
Created	Display all paragraphs with status Created.
Current	Display all paragraphs with status Current.
Deleted	Display all paragraphs with status Deleted.
History	Display history of selected Paragraph.
Trace	
Backward...	Perform backward trace of selected Paragraph.
Forward...	Perform forward trace of selected Paragraph.
Origin...	Perform origin trace of selected Paragraph.

03-03: Tools menu

Menu item	Description
Paragraph	
New...	Create a new Paragraph.
Edit...	Edit an existing Paragraph.
ChangeType	
New...	Create a new ChangeType.
Edit...	Edit an existing ChangeType.
Delete...	Delete a ChangeType.
History Trees	
New...	Create a new History Tree.
Edit...	Edit an existing History Tree.
Delete...	Delete a History Tree.
Sets	
New...	Create a new Set.
Edit...	Edit an existing Set.
Delete...	Delete a Set.
Add to...	Add selected Paragraphs to a Set.
Remove from...	Remove selected Paragraphs from a Set.
Templates	
New...	Create a new Template.
Edit...	Edit an existing Template.
Delete...	Delete a Template.

03-04: Context-sensitive menus

Right-clicking on a Paragraph should produce a pop-up menu with the following items:

- Change (subject Paragraph to a change, provide list of possible Changes based on number of selected Paragraphs)
- Select (allow forward and backward selection from this Paragraph, as well as selecting all Paragraphs in tree)
- Analysis (provide list of available analyses)
- Properties...
- Add to Set (provide list of Sets)

Right-clicking on a Change should produce a pop-up menu with the following items:

- Properties...

Right-clicking on a HistoryTree should produce a pop-up menu with the following items:

- Properties...

Right-clicking on a Set should produce a pop-up menu with the following items:

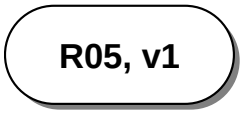
- Properties...

7.3.4 Display specifications

04-01: Paragraph display

A newly created Paragraph will be displayed like this (with label and version):

A Paragraph that has been subject to some change will be displayed like this:

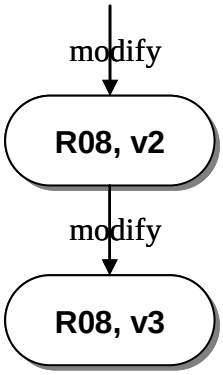


A selected Paragraph will be displayed like this:

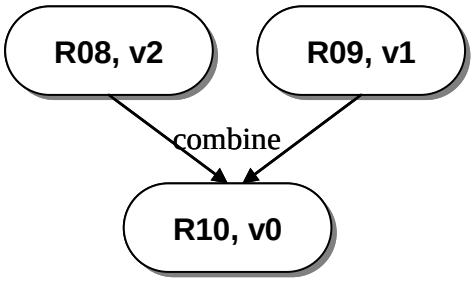
A Paragraph that has been subject to a “delete” change will be displayed like this:

04-02: Change display

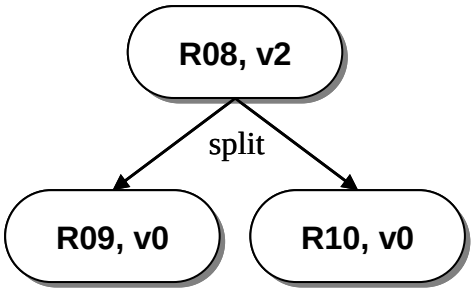
A Change (e.g.”modify”) of a single Paragraph into another Paragraph will be displayed like this:



A Change (e.g.”combine”) of two Paragraphs into another Paragraph will be displayed like this:

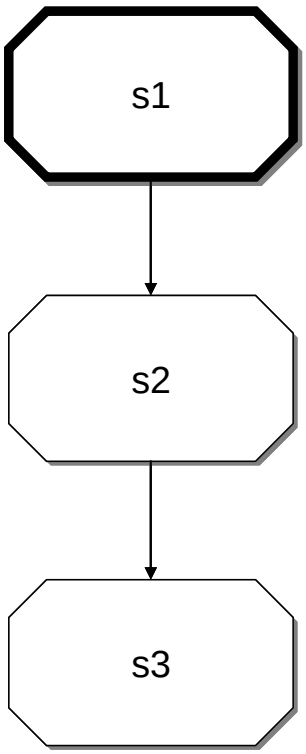


A Change (e.g.”split”) of a single Paragraph into two other Paragraphs will be displayed like this:



04-03: Set display

A closed Set (*s1*) and its open derivatives (*s2* and *s3*) will be displayed like this:


s1
s2
s3

04-04: Main display

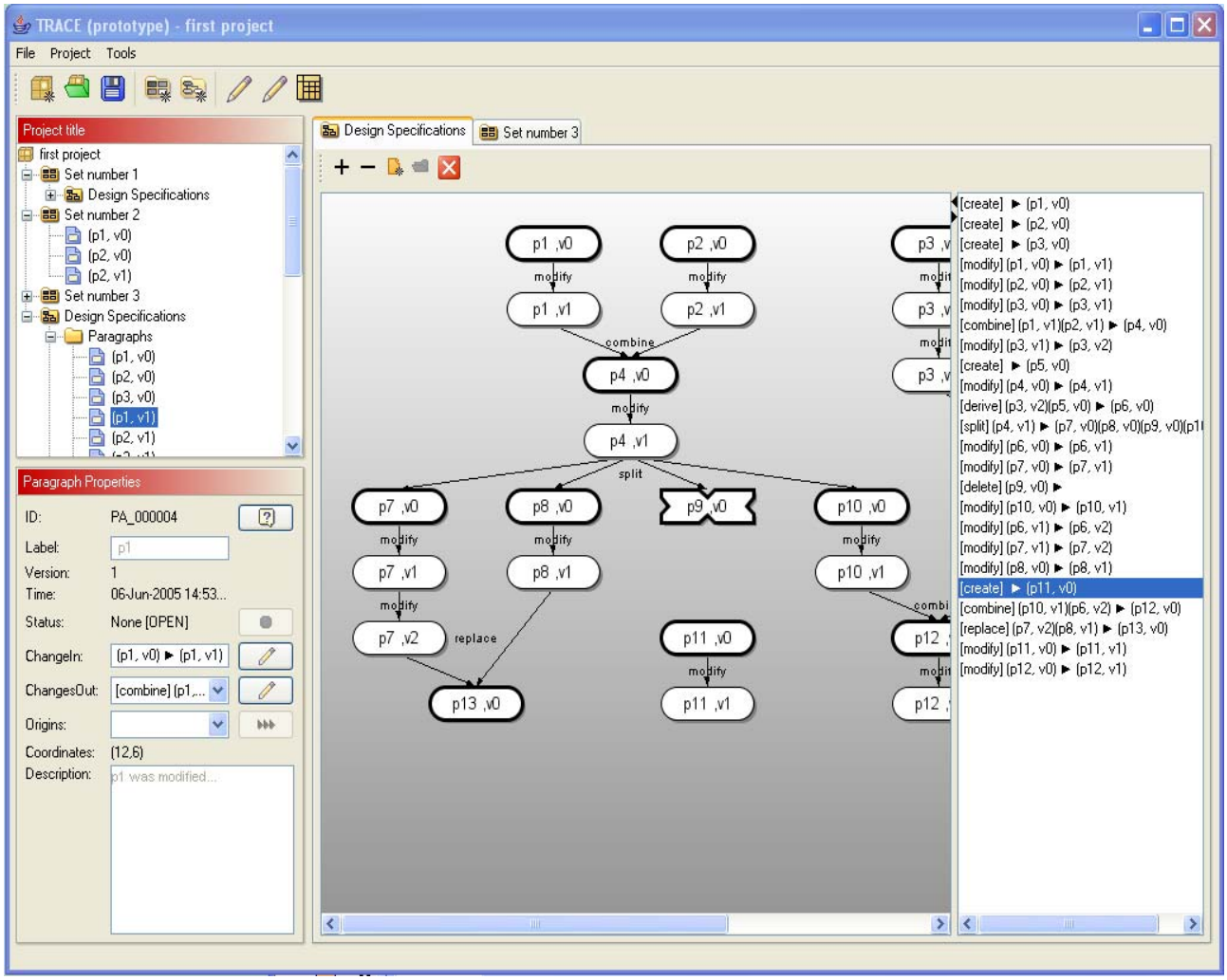


Figure 14. Main display of the TRACE prototype.

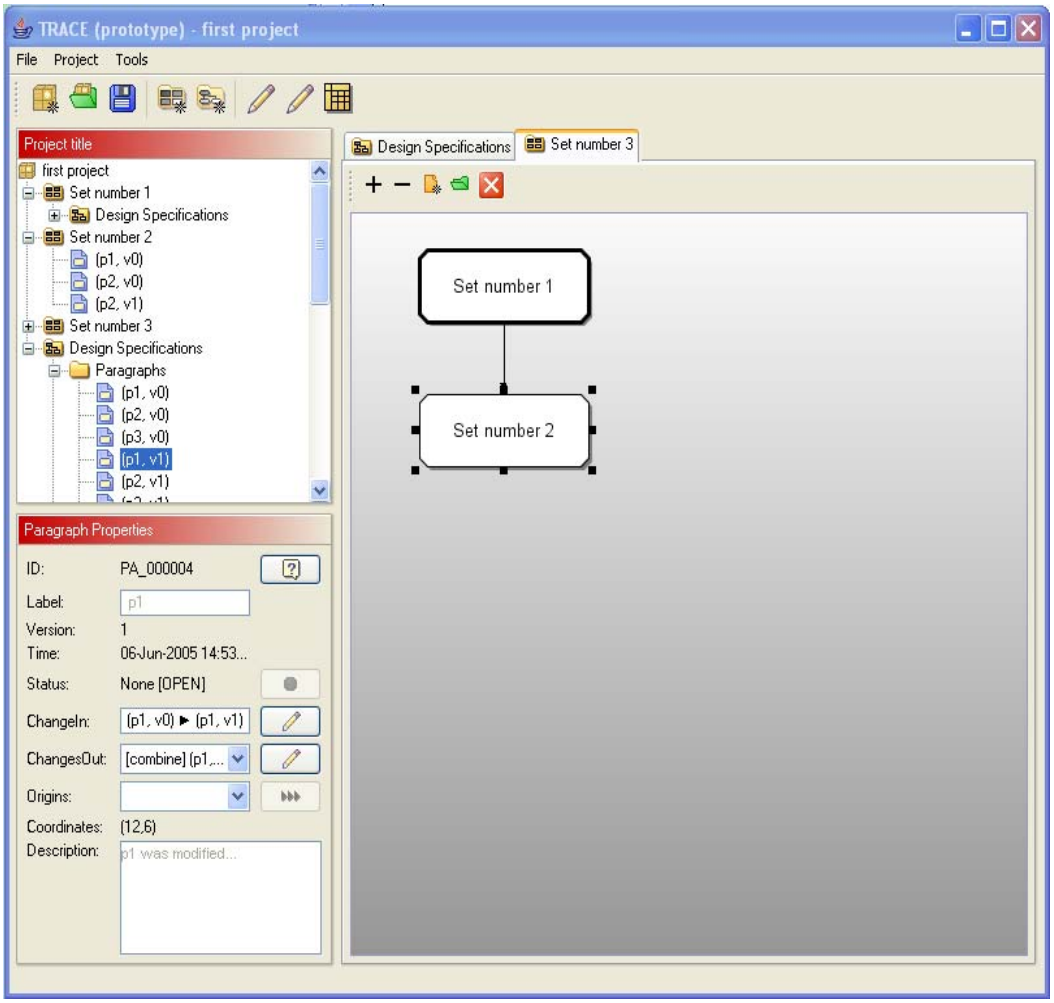


Figure 15. Graphical display of Sets.

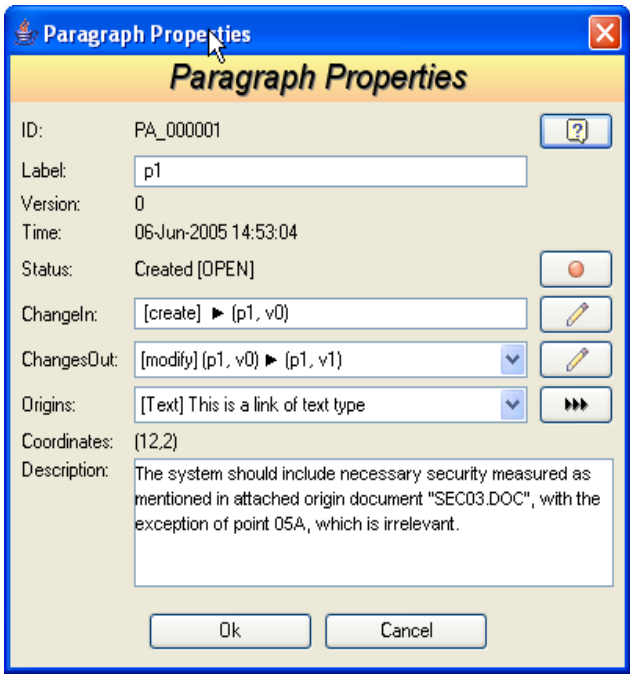


Figure 16. Paragraph properties dialog box.

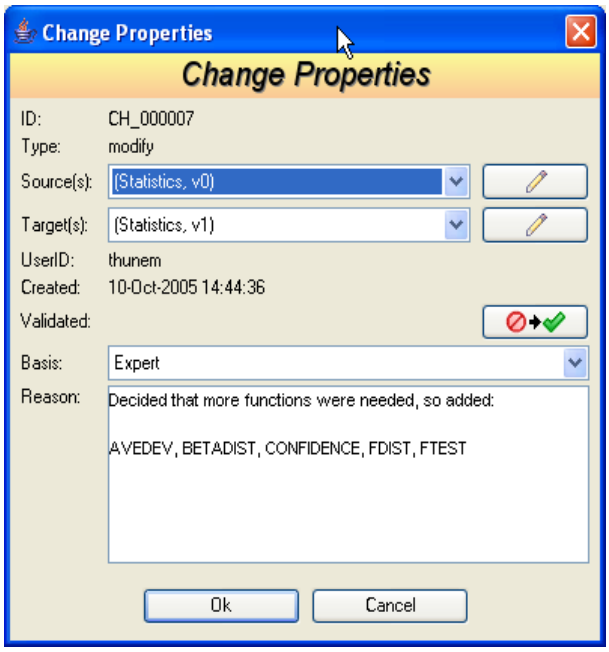


Figure 17. Change properties dialog box.

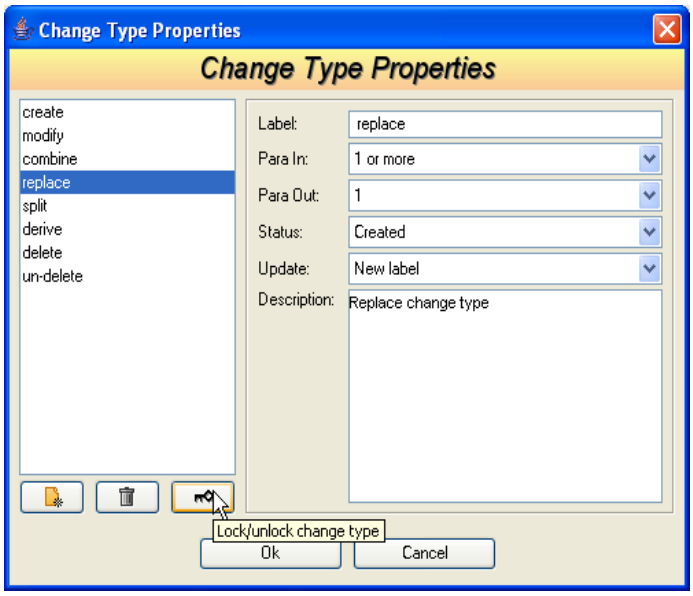


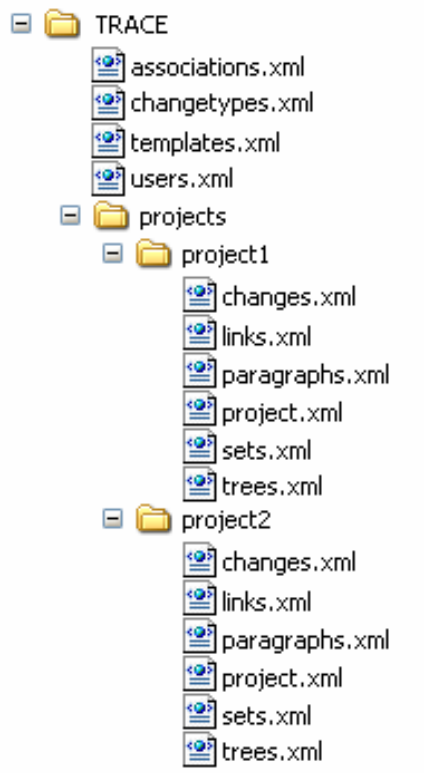
Figure 18. Change Type properties dialog box.

7.3.5 DTD/XML Specifications

All configuration and data files will be stored in various XML format, and the DTDs (Document Type Definitions) are specified here. The formats are kept as simple and similar to each other as possible to facilitate implementation and testing. Examples of XML files are given for each format.

Application-specific files (such as configuration and template files) will be placed in the application directory, while project-specific files will be placed in sub-directories, where the

directory-name is identical to the project name. An example of the directory structure is given in the figure below.



05-01: PARAGRAPH

This file will contain all Paragraphs belonging to a specific project, and will therefore be placed in the appropriate project directory.

Paragraph DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT document (paragraph+)>
<!ELEMENT paragraph (id, label, version, time, status, open, description,
changeIn?, changeOut*, origin*, posX, posY)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT label (#PCDATA)>
<!ELEMENT version (#PCDATA)>
<!ELEMENT time (#PCDATA)>
<!ELEMENT status (#PCDATA)>
<!ELEMENT open (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT changeIn (#PCDATA)>
<!ELEMENT changeOut (#PCDATA)>
<!ELEMENT origin (#PCDATA)>
<!ELEMENT posX (#PCDATA)>
<!ELEMENT posY (#PCDATA)>
```

Paragraph XML example

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE document SYSTEM "paragraphs.dtd">
```

```
<document>
     <paragraph>
          <id>PA_000001</id>
          <label>REQ001</label>
          <version>0</version>
          <time>1115891134921</time>
          <status>1</status>
          <open>false</open>
          <description>Here is some description...</description>
          <changeIn>CH_000001</changeIn>
          <changeOut>CH_000004</changeOut>
          <changeOut> CH_000005</changeOut>
          <origin>LI_000002</origin>
          <posX>5</posX>
          <posY>3</posY>
     </paragraph>
     <paragraph>
          <id>PA_000002</id>
          <label>REQ001</label>
          <version>1</version>
          <time>1115891134921</time>
          <status>0</status>
          <open>true</open>
          <description>Some changes to REQ001...</description>
          <changeIn>CH_000004</changeIn>
          <posX>5</posX>
          <posY>5</posY>
     </paragraph>
</document>
```

05-02: CHANGETYPE

This file will contain all ChangeTypes available in the application, and will therefore be placed in the application directory.

ChangeType DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT document (changetype+)>
<!ELEMENT changetype (label, paraIn, paraOut, description, resultStatus,
update, locked)>
<!ELEMENT label (#PCDATA)>
<!ELEMENT paraIn (#PCDATA)>
<!ELEMENT paraOut (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT resultStatus (#PCDATA)>
<!ELEMENT update (#PCDATA)>
<!ELEMENT locked (#PCDATA)>
```

ChangeType XML example

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE document SYSTEM "changetypes.dtd">
<document>
     <changetype>
          <label>create</label>
          <paraIn>0</paraIn>
          <paraOut>1</paraOut>
```

```
          <description>This is the create changetype</description>
          <resultStatus>1</resultStatus>
          <update>1</update>
          <locked>true</locked>
     </changetype>
     <changetype>
          <label>modify</label>
          <paraIn>1</paraIn>
          <paraOut>1</paraOut>
          <description>This is the modify changetype</description>
          <resultStatus>0</resultStatus>
          <update>2</update>
          <locked>true</locked>
     </changetype>
     <changetype>
          <label>split</label>
          <paraIn>1</paraIn>
          <paraOut>3</paraOut>
          <description>This is the split changetype</description>
          <resultStatus>1</resultStatus>
          <update>1</update>
          <locked>true</locked>
     </changetype>
     <changetype>
          <label>delete</label>
          <paraIn>1</paraIn>
          <paraOut>0</paraOut>
          <description>This is the delete changetype</description>
          <resultStatus>128</resultStatus>
          <update>0</update>
          <locked>true</locked>
     </changetype>
</document>
```

05-03: CHANGE

This file will contain all Changes belonging to a specific project, and will therefore be placed in the appropriate project directory.

Change DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT document (change+)>
<!ELEMENT change (id, type, source*, target*, status, userID, time, rea-
son, basis, validated, validateUserID)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT source (#PCDATA)>
<!ELEMENT target (#PCDATA)>
<!ELEMENT status (#PCDATA)>
<!ELEMENT userID (#PCDATA)>
<!ELEMENT time (#PCDATA)>
<!ELEMENT reason (#PCDATA)>
<!ELEMENT basis (#PCDATA)>
<!ELEMENT validated (#PCDATA)>
<!ELEMENT validateUserID (#PCDATA)>
```

Change XML example

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE document SYSTEM "changes.dtd">
<document>
     <change>
          <id>CH_000001</id>
          <type>create</type>
          <sources></sources>
          <targets>PA_000002</targets>
          <status>0</status>
          <userID>UID_0025</userID>
          <time>1115891134921</time>
          <reason>Paragraph created due to specs in doc...</reason>
          <basis>0</basis>
          <validated>false</validated>
          <validateUserID></validateUserID>
     </change>
     <change>
          <id>CH_000002</id>
          <type>split</type>
          <sources>PA_000002</sources>
          <targets>PA_000005, PA_000006</targets>
          <status>0</status>
          <userID>UID_0025</userID>
          <time>1115891184791</time>
          <reason>Paragraph split on recommendation from...</reason>
          <basis>2</basis>
          <validated>true</validated>
          <validateUserID>thunem</validateUserID>
     </change>
</document>
```

05-04: HISTORYTREE

This file will contain all HistoryTrees belonging to a specific project, and will therefore be placed in the appropriate project directory.

HistoryTree DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT document (historytree+)>
<!ELEMENT historytree (id, label, paragraph*, change*, createTime, last-
ChangeTime)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT label (#PCDATA)>
<!ELEMENT paragraph (#PCDATA)>
<!ELEMENT change (#PCDATA)>
<!ELEMENT createTime (#PCDATA)>
<!ELEMENT lastChangeTime (#PCDATA)>
```

HistoryTree XML example

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE document SYSTEM "historytrees.dtd">
<document>
     <historytree>
          <id>HT_000001</id>
          <label>Requirements specifications</label>
```

```
          <paragraph>PA_000001</paragraph>
          <paragraph>PA_000002</paragraph>
          <paragraph>PA_000003</paragraph>
          <paragraph>PA_000004</paragraph>
          <paragraph>PA_000005</paragraph>
          <paragraph>PA_000006</paragraph>
          <paragraph>PA_000007</paragraph>
          <change>CH_000001</change>
          <change>CH_000002</change>
          <change>CH_000003</change>
          <change>CH_000004</change>
          <createTime>1115891134921</createTime>
          <lastChangeTime>1118267816921</lastChangeTime>
     </historytree>
     <historytree>
          <id>HT_000002</id>
          <label>Design specifications</label>
          <paragraphs>PA_000008</paragraphs>
          <paragraphs>PA_000009</paragraphs>
          <paragraphs>PA_000010</paragraphs>
          <paragraphs>PA_000011</paragraphs>
          <paragraphs>PA_000012</paragraphs>
          <paragraphs>PA_000013</paragraphs>
          <paragraphs>PA_000014</paragraphs>
          <changes>CH_000005</changes>
          <changes>CH_000006</changes>
          <changes>CH_000007</changes>
          <changes>CH_000008</changes>
          <createTime>1115891134921</createTime>
          <lastChangeTime> 1119380718787</lastChangeTime>
     </historytree>
</document>
```

05-05: SETS

There will be several files containing Set information: a application-specific template-file containing template sets available to all projects (placed in the application directory), and project-specific files containing the sets specific to each project (placed in appropriate project directories).

Set DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT document (set+)>
<!ELEMENT set (id, label, paragraph*, change*, createTime, lastChangeTime,
tree*, subSet*, parentSet?, childSet?, open, closeTime, posX, posY)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT label (#PCDATA)>
<!ELEMENT paragraph (#PCDATA)>
<!ELEMENT change (#PCDATA)>
<!ELEMENT createTime (#PCDATA)>
<!ELEMENT lastChangeTime (#PCDATA)>
<!ELEMENT tree (#PCDATA)>
<!ELEMENT subSet (#PCDATA)>
<!ELEMENT parentSet (#PCDATA)>
<!ELEMENT childSet (#PCDATA)>
<!ELEMENT open (#PCDATA)>
<!ELEMENT closeTime (#PCDATA)>
<!ELEMENT posX (#PCDATA)>
```

```
<!ELEMENT posY (#PCDATA)>
```

Set XML example

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE document SYSTEM "sets.dtd">
<document>
     <set>
          <id>SE_000001</id>
          <label>Version 1.0</label>
          <paragraph>PA_000017</paragraph>
          <paragraph>PA_000018</paragraph>
          <paragraph>PA_000019</paragraph>
          <paragraph>PA_000020</paragraph>
          <createTime>1129019868593</createTime>
          <lastChangeTime>1129020058312</lastChangeTime>
          <childSet>SE_000002</childSet>
          <open>true</open>
          <closeTime>0</closeTime>
          <posX>16</posX>
          <posY>7</posY>
     </set>
     <set>
          <id>SE_000002</id>
          <label>Version 1.1</label>
          <paragraph>PA_000019</paragraph>
          <paragraph>PA_000020</paragraph>
          <paragraph>PA_000024</paragraph>
          <paragraph>PA_000025</paragraph>
          <createTime>1129019907937</createTime>
          <lastChangeTime>1129020064000</lastChangeTime>
          <parentSet>SE_000001</parentSet>
          <childSet>SE_000003</childSet>
          <open>true</open>
          <closeTime>0</closeTime>
          <posX>16</posX>
          <posY>14</posY>
     </set>
     <set>
          <id>SE_000003</id>
          <label>Version 1.2</label>
          <paragraph>PA_000019</paragraph>
          <paragraph>PA_000020</paragraph>
          <paragraph>PA_000024</paragraph>
          <paragraph>PA_000026</paragraph>
          <createTime>1129020025687</createTime>
          <lastChangeTime>1129020025687</lastChangeTime>
          <parentSet>SE_000002</parentSet>
          <open>true</open>
          <closeTime>0</closeTime>
          <posX>16</posX>
          <posY>21</posY>
     </set>
</document>
```

05-06: LINKS

This file will contain all HistoryTrees belonging to a specific project, and will therefore be placed in the appropriate project directory.

Link DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT document (link+)>
<!ELEMENT link (id, type, string)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT string (#PCDATA)>
```

Link XML example

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE document SYSTEM "links.dtd">
<document>
     <link>
          <id>LI_000001</id>
          <type>0</type>
          <string> This Paragraph was included due to a talk...</string>
     </link>
     <link>
          <id>LI_000002</id>
          <type>1</type>
          <string>p08-basis.doc</string>
     </link>
     <link>
          <id>LI_000003</id>
          <type>2</type>
          <string>       http://standards.ieee.org/catalog/olis/index.html
</string>
     </link>
     <link>
          <id>LI_000004</id>
          <type>3</type>
          <string>PA_000004</string>
     </link>
     <link>
          <id>LI_000005</id>
          <type>4</type>
          <string>SE_000004</string>
     </link>
</document>
```

05-07: PROJECT

This file will contain relevant information regarding a specific Project, and will therefore be placed in the appropriate project directory. A list of all available projects will be placed in the application directory.

Project XML example

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
     <comment>---Project information---</comment>
     <entry key="description">Software Development Project for a Calcula-
```

```
tor application providing a number of mathematical functions.</entry>
     <entry key="time">1128936487531</entry>
</properties>
```

05-08: File Associations

This file will contain all file associations, linking specific applications to specific file types.

Association XML example

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
     <comment>---file associations---</comment>
     <entry key="XML">C:\PROGRAM FILES\XML MARKER\XMLMARKER.EXE</entry>
     <entry              key="PPT">C:\PROGRAM              FILES\MICROSOFT
OFFICE\OFFICE11\POWERPNT.EXE</entry>
     <entry key="TXT">C:\PROGRAM FILES\TEXTPAD 4\TEXTPAD.EXE</entry>
</properties>
```

8. Appendix C: The Dissemination Activities

The activities related to the project MORE included a presentation in an NKS initiated seminar on decommissioning projects in Nordic countries (Roskilde, Denmark, September 13-15, 2005), a paper presentation and demonstration of the prototype during SAFECOMP 2005 conference (Fredrikstad, Norway, September 28-30, 2005), a paper presentation and demonstration during the EHPG 2005 (Lillehammer, Norway, October 17-21, 2005), a project meeting (October 18, 2005), and a paper presentation and demonstration during an IAEA special meeting (Espoo, Finland, November 22-24, 2005). The participation during the two latter events also included meetings and discussions with the staff members of FORTUM who are involved in modernisation projects at Loviisa NPP.

The following provides brief description of each activity.

8.1 NKS Initiated Seminar on Decommissioning

The seminar was arranged by NKS and in collaboration with Dansk Dekommissionering. The focus was on decommissioning activities in Nordic countries, and the aim was to allow as many as possible presentations of 5-10 minutes duration. The last day of the seminar was reserved for group work based on a pre-prepared set of questions and issues to discuss.

8.2 SAFECOMP 2005

Since it was established in 1979, by the European Workshop on Industrial Computer Systems, Technical Committee 7 on Reliability, Safety and Security, EWICS TC7, SAFECOMP has contributed to the progress of the state-of-the-art in dependable applications of computer-systems. SAFECOMP is an annual event covering the state of-the-art, experience and new trends in the areas of computer safety, reliability and security regarding dependable applications of computer systems.

SAFECOMP 2005 focused on dependability of critical computer applications. Due to the increasing awareness and importance of security issues of critical computer-based systems, SAFECOMP 2005 emphasised work in this area. Nowadays practical experience points out the need of multidisciplinary approaches to deal with the nature of critical complex settings. SAFECOMP 2005, therefore, was open to multidisciplinary work enhancing understanding across disciplines.

8.3 EHPG 2005

The EHPG meeting in 2005 was the 32nd in the series of Enlarged Halden Programme Group meetings. It was arranged in order to promote dissemination of the results of the Halden Project's research activities, and further to identify and discuss the research priorities of the member organisations of the Project.

The meeting reviewed activities in all the main areas of the Project's work. Reports on the joint programme results and on results from participant sponsored programmes were presented, as well as papers on related work performed at the participants' own establishments.

Invited papers reviewing topics of interest within the scope of the Project's activities were equally presented.

8.4 Project Meeting (Minutes)

The meeting was a combined project meeting and a meeting with FORTUM/Loviisa. The present were: Samuli Savolainen (Loviisa NPP), Olli Ventä (VTT), Janne Valkonen (VTT), Jan Posmyr (IFE), Atoosa P-J Thunem (IFE), Harald P-J Thunem (IFE), and Rune Fredriksen (IFE).

Atoosa P-J Thunem presented an introduction to the TACO and MORE projects and the traceability model developed in the project TACO. She explained that this model will be further improved in the project MORE, along with the TRACE tool supporting an approach for dependable requirements engineering. She stressed the need for one or several test cases in order for the project MORE and its results to become more applicable towards modernisation projects and other activities (e.g., maintenance improvement activities) at NPPs.

Samuli Savolainen suggested that he could ask people at Loviisa to become involved. The best person might be someone from the QA department or archive. The plan was therefore:

1. A group visit to Loviisa to see the small case study
2. Access to some documentation of the case study (An issue might be that the documentation is in Finnish)

Atoosa P-J Thunem will send an email to Samuli Savolainen about the intention behind a contact with Loviisa, including a brief introduction to the project (and remembering to point out that the work will be performed by the members of the project MORE, and will not cost anything for Loviisa beyond providing the test case). She will mention that the project members would like to come to Loviisa for a visit in December 2005. Samuli Savolainen will forward the email directly to Markku Tiitinen, Mikko Pihlatie and Arvo Vuorenmaa.

The MORE team would also like to get input on how people at Loviisa work with traceability issues and how these challenges should be dealt with in the future.

8.5 IAEA Meeting

The purpose of the meeting was to provide an international forum for presentation and discussion of experience in implementing and licensing digital I&C systems and equipment in nuclear power plants. The meeting was intended for I&C experts from power utilities, vendor companies, licensing bodies, research organisations and academic institutions. The meeting provided both experience from earlier projects and descriptions of new and planned I&C projects. The meeting was hosted by VTT and was attended by 85 participants from 24 countries presenting 27 papers. During the meeting, new innovative methods and tools to test and validate the implementation and operation of digital systems were also presented. In addition, a TECDOC initiated in August 2005 and with the focus on implementing and licensing digital I&C systems and equipment in nuclear power plants was further discussed during the meeting.

Bibliographic Data Sheet **NKS-133**

Title	MORE. Management of Requirements in NPP Modernisation Projects. Project Report 2005
Author(s)	Atoosa P-J Thunem[1], Rune Fredriksen[1], Harald P-J Thunem[1], Olli Ventä[2], Janne Valkonen[2] and Jan-Erik Holmberg[2]
Affiliation(s)	[1]IFE, Norway [2]VTT Technical Research Centre of Finland
ISBN	87-7893-195-9 *Electronic report*
Date	April 2006
Project	NKS_R_2005_47 MORE
No. of pages	72
No. of tables	1
No. of illustrations	6
No. of references	8
Abstract	The overall objective of the project MORE is to improve the means for managing the large amounts of evolving requirements in Nordic NPP modernisation projects. In accordance to this objective, the activity will facilitate the industrial utilisation of the research results from the project TACO. On the basis of experiences in the Nordic countries, the overall aim of the TACO project has been to identify the best practices and most important criteria for ensuring effective communication in relation to requirements elicitation and analysis, understandability of requirements to all parties, and traceability of requirements. The project resulted in the development of a traceability model for handling requirements from their origins and through their final shapes. Particular emphasis for the MORE project in 2005 was put on utilising a prototype of a tool (TRACE) intended to support an adopted approach to dependable requirements engineering, suitable for modelling and handling large amounts of requirements related to all stages of the systems development process and not only those traditionally including requirements at high-level stages.
Key words	MORE, tracability of requirements, dependable requirements engineering, TRACE

Available on request from the NKS Secretariat, P.O.Box 49, DK-4000 Roskilde, Denmark.
Phone (+45) 4677 4045, fax (+45) 4677 4046, e-mail nks@nks.org, www.nks.org