SUPPORTING TRACEABILITY AND INCONSISTENCY MANAGEMENT BETWEEN SOFTWARE ARTEFACTS

Thomas Olsson[1] and John Grundy[2]

[1] Dept. of Communication Systems
Lund University
Box 118
SE-221 00 Lund
Sweden
thomas.olsson@telecom.lth.se

[2]Dept. of Computer Science
University of Auckland
Private Bag 92019
Auckland
New Zealand
john-g @cs.auckland.ac.nz

Abstract

Software artefacts at different levels of abstraction are closely inter-related. Developers require support for managing these inter-relationships as artefacts evolve during development. We describe a conceptual architecture and prototype for supporting traceability and inconsistency management between software requirements descriptions, UML-style use case models and black-box test plans. Key information models are extracted from each of these different kinds of software artefacts and elements in different models are implicitly or explicitly linked. Changes to one software artefact are detected and propagated to related artefacts in different information models and inform developers of change impacts.

Keywords: traceability, inconsistency management, requirements encoding, use case models, test plans

1 Introduction

As software systems continue to grow larger and more complex the software developer is faced with managing more and more information about a system. Exacerbating this information management problem is the growing demands on time to market and increased software quality. As the amount and diversity of information about software systems grows, so does the need for supporting consistency and traceability among different levels of abstraction for developers [15, 1, 6]. Software information is represented in a wide variety of representations[1], in different notations, managed by different software development tools, and the information is captured with different purposes. Such information includes functional and non-functional requirements, use cases, object-oriented analysis and design models, user interface designs, code, black-box and white-box test plans, user documentation and so on.

Support for relating information across such representations is still quite weak [6, 11]. Therefore, the linking and transformation work has to, a large extent, be done manually. This is an error-prone and tedious process. We describe a new approach to managing fuzzy relationships between high-level software artefacts, namely requirements, use case models and black-box test plans. We formulate an abstract model for each kind of software information and allow elements in one representation to be linked to related elements in another representation by both explicit and implicit means. Links are used to support traceability between different representations, visualisation of cross-representational software information, and change management for inconsistency tracking between representations.

2 Related work

Most software tools support one or more software information models [8, 10]. Most software development projects use multiple tools when developing software, choosing appropriate tools to support different phases of development and different kinds of software information management [8]. The problem with many of the current approaches is that relating information across different artefacts is either not possible or very simplistic. Most tools support working only with one part of the development process, e.g. requirements or design, with limited support to relate the information to other tools or parts of the process [1, 10]. Tools that support multiple phases of development and multiple information models typically provide limited traceability and consistency management between artefacts [5, 9]. More elaborate support is usually found in low-level representations like design or source code. For high-level representations, support is very basic if even present at all.

[1] Representation, artefact and document are in this context the same, referring to the entity where the information is represented and stored.

There has been a substantial amount of research put into issues such as inconsistency management [13], traceability [18], logic-based formal methods [1], software tool integration [5, 8] and round-trip engineering [12]. Much of this research is, however, usually focused on either low-level abstractions or on formal methods to encode software information. Many current multi-level software information management approaches require the notations worked with have to have a logic basis to support consistency manaement [1, 15, 11]. Most of these approaches usually focus on one or closely related types of documents, not allowing the developer to create relations across quite different types of software artefacts[17, 13, 6, 11].Many integrated software environments utilise a common data model for all software artefacts [1, 3, 5], enforcing representational consistency. We have found this approach does not sufficiently tolerate the fuzzy inter-relationships many software artefact representational elements have. Many changes to one model cannot be automatically applied to another without losing significant information or making incorrect assumptions [6].

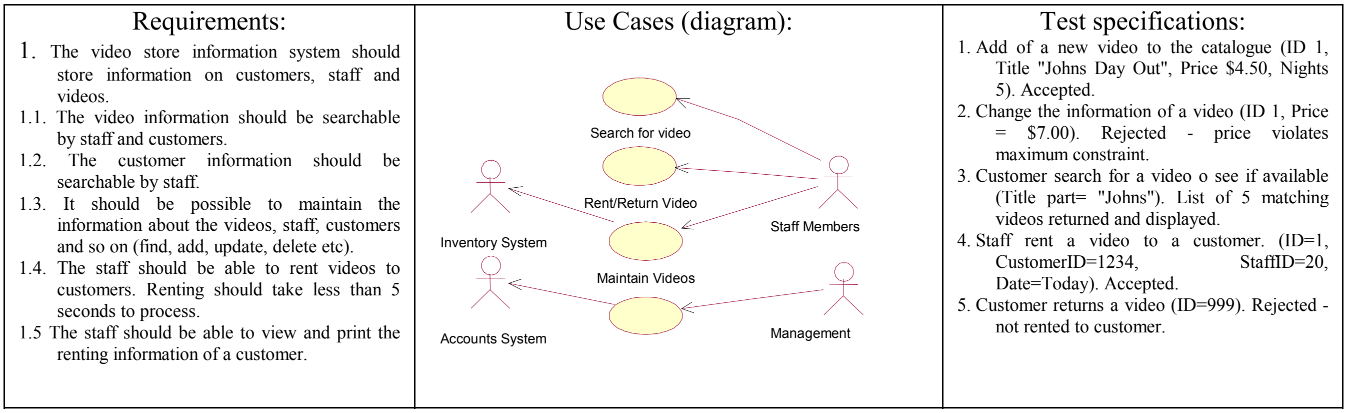
Tool data exchange research adopts a different approach whereby information in multiple tools can be exchanged in common formats, or associated by a linking representational model [8, 19, 3]. Common data format approaches for the most part do not track changes to one representational model against another and thus typically loose information or can not perform many change propagations necessary [19]. The Xlinkit toolkit [11] allows developers to express required consistency constraints between software artefact information models, with developers informed of inconsistencies present. Xlinkit requires XML-encoded artefact data and detects inconsistencies by checking for constraint violations. JComposer provides an environment and architecture for relating software artefact elements at different levels of abstraction and propagating changes between representations by the use of "change descriptions" [7]. This approach supports traceability and inconsistency management but has not been applied to high-level artefact consistency like requirements nor test plan content.

Viewpoints have been used in many projects to manage multiple views on software information [4, 2, 20]. Most approaches assume hard consistency between views, either not allowing for inconsistency or if tolerating inconsistency, using very formal representation models that are hard to present to developers [6]. Hyperslices provide a general mechanism for taking multiple perspectives of complex system artefacts but traceability and inconsistency management is not specified [16].

3 Our Approach

We have conducted an empirical study to better understand how software developers use and manage software information artefacts in the workplace, and to understand how tool support should contribute to this [14]. Key findings to date indicate that what is required for high-level artefact management is support for traceability and inconsistency management between views that informs developers of key interrelationships and where changes in one model have impacted another. While the quality of many high-level information representations is often considered to be low, most participants in the survey indicated that these higher level software system representations are often a key communication means between developers.

A key underlying assumption is that information about the same part of a piece of software is present in several places, levels of abstraction and in different representations, as illustrated in Figure 1. In this example, the requirements of an on-line video system are sketched out. The informal, natural language-based requirements codification are related to formalisations of these requirements, such as a UML use case model as illustrated in the middle column. Overlapping information includes user/actor identification, functions/actions, data input and output and non-functional constraints/special conditions. Similarly, black-box test plan items are related to use case and requirements elements.



Requirements:

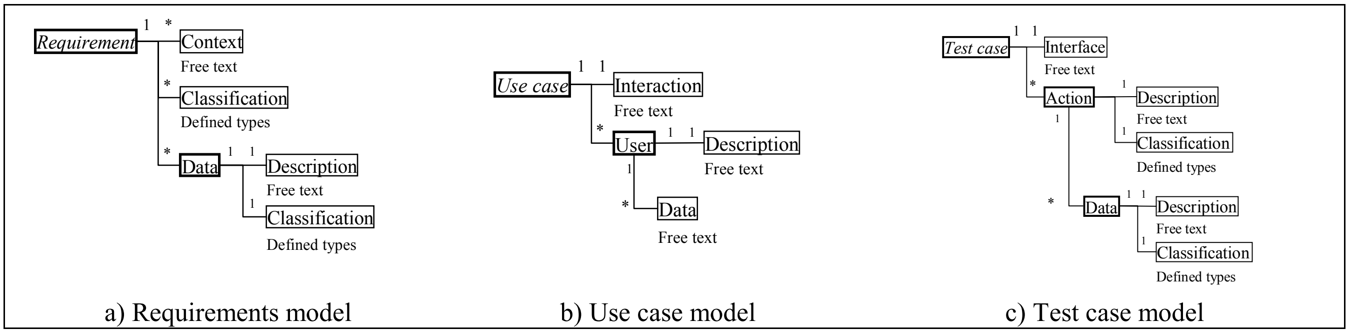
1. The video store information system should store information on customers, staff and videos.
1.1. The video information should be searchable by staff and customers.
1.2. The customer information should be searchable by staff.
1.3. It should be possible to maintain the information about the videos, staff, customers and so on (find, add, update, delete etc).
1.4. The staff should be able to rent videos to customers. Renting should take less than 5 seconds to process.
1.5 The staff should be able to view and print the renting information of a customer.

Use Cases (diagram):

Test specifications:

1. Add of a new video to the catalogue (ID 1, Title "Johns Day Out", Price $4.50, Nights 5). Accepted.
2. Change the information of a video (ID 1, Price = $7.00). Rejected - price violates maximum constraint.
3. Customer search for a video o see if available (Title part= "Johns"). List of 5 matching videos returned and displayed.
4. Staff rent a video to a customer. (ID=1, CustomerID=1234, StaffID=20, Date=Today). Accepted.
5. Customer returns a video (ID=999). Rejected - not rented to customer.

Figure 1 Example on how information overlaps among artefacts.



a) Requirements model b) Use case model c) Test case model

Figure 2 Meta models

Given this multiple representation information overlap we aim at identifying and characterizing key information relationships between elements of quite different software artefacts. In this paper we focus on those between use case models, functional and non-functional requirements descriptions, and black-box test plans. Our approach is to summarise the key software information content in each representation by extracting "essential information" from each representational model into abstracted representational models. We then identify relationships between abstract elements from each representation and create relations among these elements of different representations. These relationships can either be created explicitly by developers specifying them or implicitly by heuristics and automatically created by a software tool. There is a need to explicitly create relations since a full automation is far away or even not possible, especially since our current focus is currently on high-level natural language documents that often lack well-defined formal abstractions for all software artefacts in the representation. Having created relationships across representations a developer can navigate among views. A view might be an entire artefact or parts of it, or may also be a combination of parts from different artefacts. By having views the developer does not have to picture the relationships in their head but can instead see them on the computer screen. The relationships can also be used for change impact analysis, inconsistency management, traceability and so on.

4 Relationship Specification and Usage

4.1. Meta model of Abstract Representational Elements

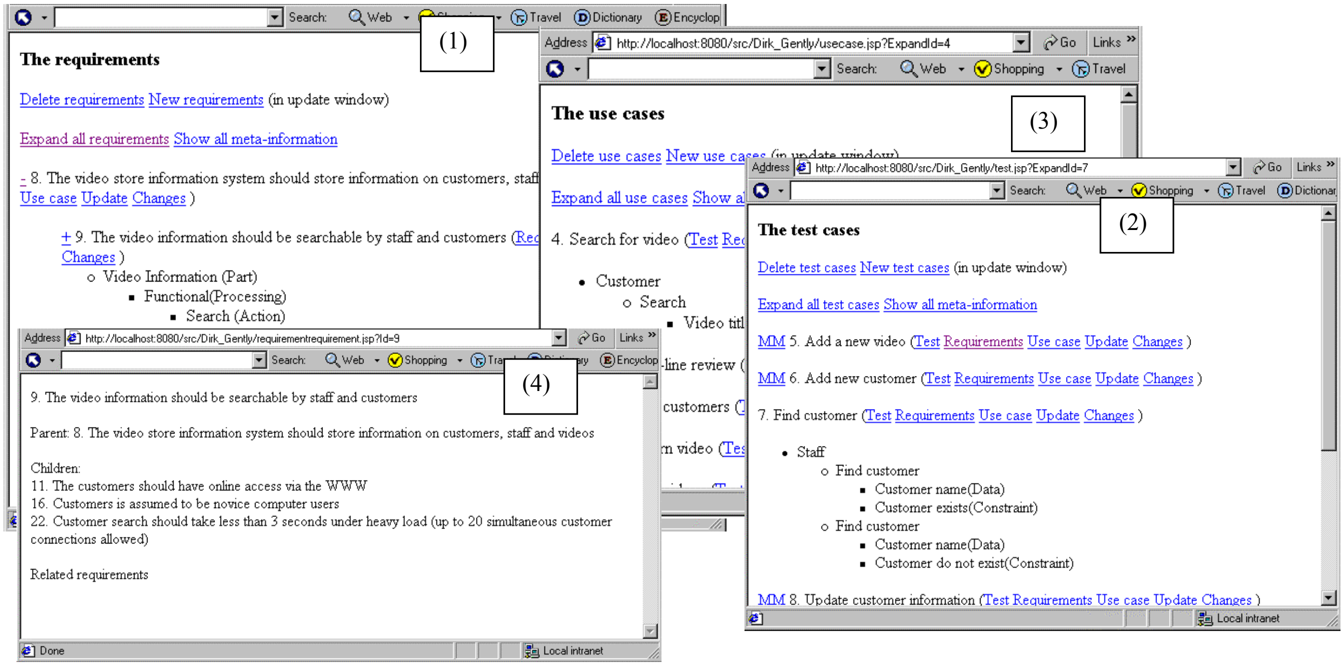
We aim to support developers creating and relating software information artefacts across representations where elements are often imprecise, inter-relationships often ill-defined and change impacts unclear on related elements. Instead of trying to define very detailed structural and semantic representations for all of the software representations we are considering, we adopt an approach that focuses on capturing key information from each that can be related to other representations. Figure 2 summarises our current meta-models - typical sources of this information include Word and Powerpoint documents, CASE tool databases and testing tool scripts. We begin with a very basic model of each of these software representation informational models being considered. For example, a use-case is characterized by three basic elements: the involved actors, the interaction performed by the action and data used in the interaction. The details associated with each meta-model element can be further refined to capture the information about a software system in more structured detail. A grouping mechanism allows developers to put related requirements elements together into hierarchies, the use-cases into sequences, and test cases into related sets of test plans.

4.2. Inter-representational Relationships

Various relationships can be specified between elements and groups of elements in each software representational model, some outlined in Table 1.

Relationship	Elements Related
Exact (1:1)	Exact duplication of information from one representation in another e.g. screen name in use case action and test case
Specialisation	More detailed information in one model based on information in another e.g. general functional requirement and detailed use case actions for this requirement
Generalisation	More general (abstract) information in one model based on information in another e.g. test plan values and functional requirement or non-functional requirement they are used to validate
Similar	Similar concepts in different representations e.g. one group of users in requirements and related actors in use cases
Splits (1:many)	Element or group of elements in one model explodes into multiple elements in another e.g. one functional requirement into multiple test plans or use cases/use case actions.
Merges (many:1)	Group of elements or multiple groups from one model merges into a single element or single group e.g. one unit under test relates to multiple non-functional constraints on it.
Exact group (m:n)	Each element of group in one representation related to an element in group of other representation e.g. each use case action to a test plan for interface.

Table 1. Relationships between elements.



The requirements

Delete requirements New requirements (in update window)

Expand all requirements Show all meta-information

- 8. The video store information system should store information on customers, staff
Use case Update Changes)

+ 9. The video information should be searchable by staff and customers (Req
Changes)
  - Video Information (Part)
    - Functional(Processing)
      - Search (Action)

http://localhost:8080/src/Dirk_Gently/usecase.jsp?ExpandId=4

The use cases

Delete use cases New use cases (in update window)

Expand all use cases Show al

4. Search for video (Test Req

- Customer
  - Search
    - Video titl

-line review (

customers (T

rn video (Tes

http://localhost:8080/src/Dirk_Gently/test.jsp?ExpandId=7

The test cases

Delete test cases New test cases (in update window)

Expand all test cases Show all meta-information

MM 5. Add a new video (Test Requirements Use case Update Changes)

MM 6. Add new customer (Test Requirements Use case Update Changes)

7. Find customer (Test Requirements Use case Update Changes)

- Staff
  - Find customer
    - Customer name(Data)
    - Customer exists(Constraint)
  - Find customer
    - Customer name(Data)
    - Customer do not exist(Constraint)

MM 8. Update customer information (Test Requirements Use case Update Changes)

http://localhost:8080/src/Dirk_Gently/requirementrequirement.jsp?Id=9

9. The video information should be searchable by staff and customers

Parent: 8. The video store information system should store information on customers, staff and videos

Children:
11. The customers should have online access via the WWW
16. Customers is assumed to be novice computer users
22. Customer search should take less than 3 seconds under heavy load (up to 20 simultaneous customer connections allowed)

Related requirements

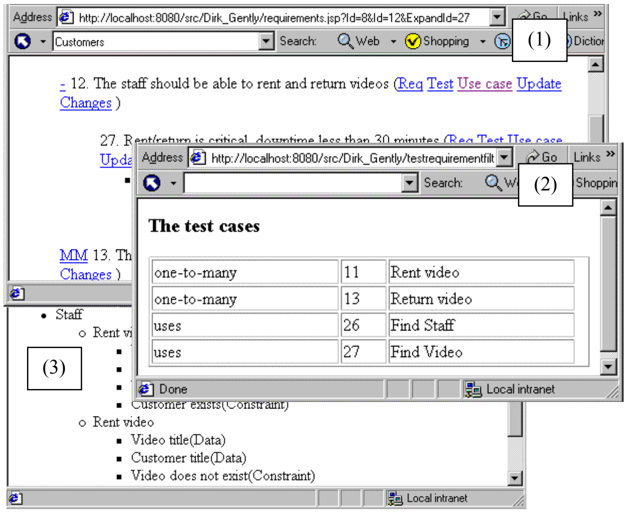
Figure 3. Example on the meta model in a web based prototype.

4.3. Artefact Representation

We have developed a prototype tool to support the capture, summarisation and linking of software information. This includes support for extracting information from source documents, viewing artefact information in its summarised form, tracking changes to artefact elements and managing change to artefacts in different representations. The ultimate aim of this tool is to provide value-added support for existing software artefact management tools. Figure 3 illustrates the capture and display of software artefact data in our prototype. (1) shows a requirements outline for the on-line video system, captured from a Word document. This describes key functional and non-functional constraints. Requirements can be expanded or collapsed for managing complex system descriptions. (2) shows a use case model for the video system, captured from a CASE tool. This shows key actors, use cases, actions within use cases, and data input, output and processing within use case actions. (3) shows black-box test plan, captured from a testing tool's test database. This shows test plan actions, input data and expected result data. (4) shows a requirement summary.

4.4. Relationships across artefacts

Relationships between elements and groups in different representations can be implicitly inferred from the meta-model element relationships or data for a system being modelled. For example, users in the requirements model can be associated 1:1 with actors in the use case model with the same names/roles. A group of test plans associated with a single named user interface can be associated with the use case actions for this named interface and requirements constraining the named interface or describing its functions. Explicit relationships are defined by the tool user to give further information about related representational elements. These relationships can be used to express exact element correspondence, specialisation, generalisation, and various forms of cross-representational loose element association. Figure 4 shows three views. (1) is the requirements view for the rent/return video part of the system. In (2) the user is viewing relationships between one of the requirements and test plan elements. In (3) a test plan element's data is being viewed.



http://localhost:8080/src/Dirk_Gently/requirements.jsp?Id=8&Id=12&ExpandId=27

Customers

- 12. The staff should be able to rent and return videos (Req Test Use case Update Changes)

27. Rent/return is critical, downtime less than 30 minutes (Req Test Use case
Upda

MM 13. Th
Changes)

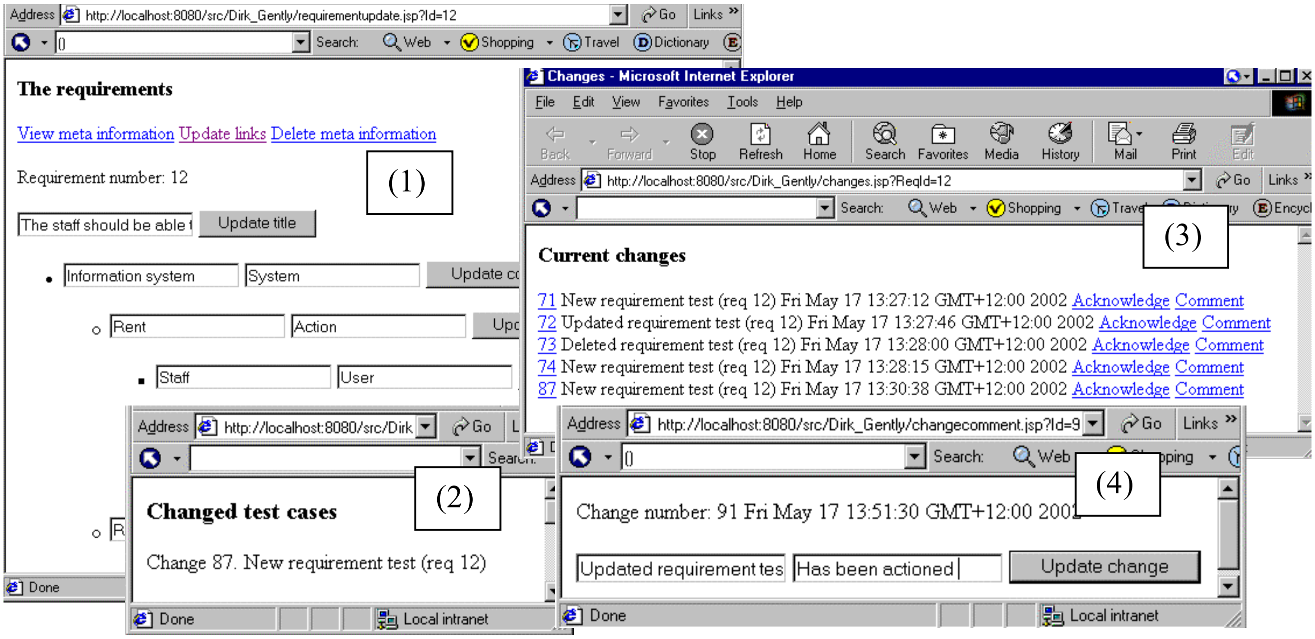
- Staff
  - Rent vi
    - Customer exists(Constraint)
  - Rent video
    - Video title(Data)
    - Customer title(Data)
    - Video does not exist(Constraint)

http://localhost:8080/src/Dirk_Gently/testrequirementfilt

The test cases

one-to-many	11	Rent video
one-to-many	13	Return video
uses	26	Find Staff
uses	27	Find Video

Figure 4 Inter-notational relationships and navigation.



Address http://localhost:8080/src/Dirk_Gently/requirementupdate.jsp?Id=12

The requirements

View meta information Update links Delete meta information

Requirement number: 12

(1)

The staff should be able t | Update title

- Information system | System | Update co
  - Rent | Action | Upd
    - Staff | User
  - R

Done

Changes - Microsoft Internet Explorer

Address http://localhost:8080/src/Dirk_Gently/changes.jsp?ReqId=12

Current changes

(3)

71 New requirement test (req 12) Fri May 17 13:27:12 GMT+12:00 2002 Acknowledge Comment
72 Updated requirement test (req 12) Fri May 17 13:27:46 GMT+12:00 2002 Acknowledge Comment
73 Deleted requirement test (req 12) Fri May 17 13:28:00 GMT+12:00 2002 Acknowledge Comment
74 New requirement test (req 12) Fri May 17 13:28:15 GMT+12:00 2002 Acknowledge Comment
87 New requirement test (req 12) Fri May 17 13:30:38 GMT+12:00 2002 Acknowledge Comment

Address http://localhost:8080/src/Dirk

Changed test cases

(2)

Change 87. New requirement test (req 12)

Done | Local intranet

Address http://localhost:8080/src/Dirk_Gently/changecomment.jsp?Id=9

Change number: 91 Fri May 17 13:51:30 GMT+12:00 2002

(4)

Updated requirement tes | Has been actioned | Update change

Done | Local intranet

Figure 5 Change propagation and management example

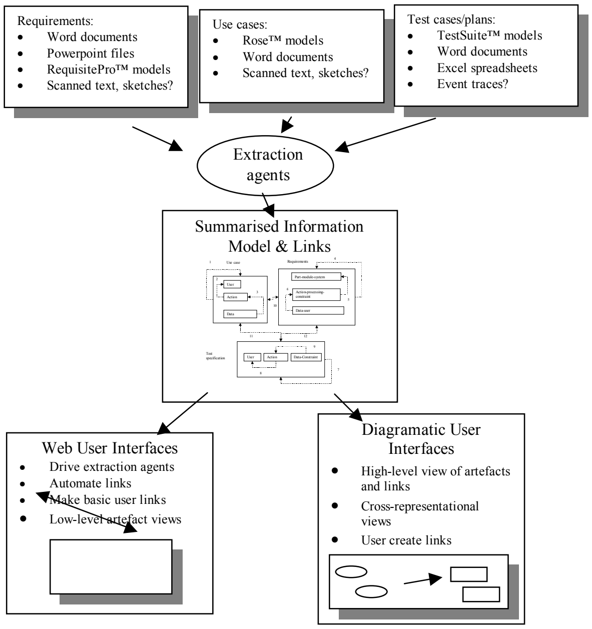


Figure 6. Prototype tool architecture.

4.5. Change propagation

To support inconsistency management all changes made to artefacts, whether explicitly in the tool or by importing changed data from other tools, are tracked by change representations. The developer is informed when viewing elements in other representations that changes have been made that impact them. Some changes can be automatically applied by the tool e.g. renaming an actor might result in a one-to-one, same-named user being renamed in requirements and test plan representations. Other changes need developer intervention. For example, the effects of strengthening a non-functional performance or user interface interaction constraint must be manually applied to a use case model.

Our prototype tool captures changes, represents them as change description objects and displays these to developers when appropriate. For example, in Figure 5 when a requirement is modified (1), all use case elements and test plan elements impacted by this change are updated to indicate they may need to be modified (2). Change descriptions may be hidden by the developer, indicated as "actioned" or a summary report of impacted items not yet inspected obtained. Changes are also associated with the originating element providing a change history (3). Changes can be accepted by developers and hidden from change lists, and can be annotated to support developer discussion (4).

5 Prototype Tool

Our tool prototype's architecture is illustrated in Figure 6. We use a set of "extraction agents" to capture summarised information from a wide variety of software information sources. These include Word™, Excel™ and PowerPoint™ documents; Rose™, TestSuite™ and RequisitePro™ CASE tool data models; and possibly may extend to scanned text or even diagram sketches and Robot™ test driver event traces. The web-based tool interface is used to view summarised artefact data and to support basic explicit linking of elements in different representational models. The user can move between different artefacts via these links, supporting traceability, and changes to elements in one representation are propagated to linked elements in other representational models. Where possible, changes to these linked elements are made, but often "change messages" documenting related element changes [6] are used to indicate change impacts. We are also prototyping a visual tool used to provide high-level diagrammatic views on artefacts.

6 Summary

We have described an approach to supporting traceability and change management between functional and non-functional requirements summaries, use case models and black-box test plans. The essence of our approach is to summarise these different software information models, distilling their "core" elements, element properties and inter-element relations. Implicit (automatic) and explicitly made links are then created between elements in different software representation models. This allows developers to navigate between models using related element links; to have cross-representational views provided; and to support change impact visualisation and management. We have prototyped an information management tool using a combination of data integration components providing information extraction from a wide variety of common software information models; a database capturing summarised information models and cross-linked elements; and web-based data capture, linking, viewing, navigation and change management views. Key extensions will include providing richer information visualisation including graphical link and notational element display. Extraction agents need to support both complex document parsing and data extraction as well as change detection and ultimately document update. We intend to make the representation meta-models editable so different organizations can specify their own extensions.

References

1. Allison W., Carrington D., Jones T., Stewart-Zerba L., Welsh J. Visualising software documents in a generic development environment. In *Proceedings of the 1997 Australian Software Engineering Conference*, IEEE CS Press, pp.49-59.
2. Emmerich, W., CORBA and ODBMSs in Viewpoint Development Environment Architectures., In *Proceedings of the 4th International Conference on Object-Oriented Information Systems,* Springer Verlag, 1997, pp. 347-360.
3. Furguson, R.I., Parrington, N.F., Dunne, P., Archibald, J.M. and Thompson, J.B. MetaMOOSE – an Object-oriented Framework for the Construction of CASE Tools, *Proceedings of CoSET'99*, Los Angeles, 17-18 May 1999, University of South Australia, pp. 19-32.
4. Finkelstein, A., Gabbay, D., Hunter, A., Kramer, J., Nuseibeh, B. Inconsistency Handling in Multiperspective Specifications, *IEEE Transactions on Software Engineering*. Vol. 20, no. 8, pp. 569-578, August, 1994.
5. Gray, .J.P., Liu, A. and Scott, L. Issues in software engineering tool construction, *Information and Software Technology*, **42** (2), Elsevier, 73-77.
6. Grundy, J.C., Hosking, J.G., Mugridge, W.B. Supporting inconsistency management for multiple-view software development environments, IEEE Transactions on Software Engineering, vol. 24, no. 11, November 1998.
7. Grundy, J.C., Mugridge, W.B. and Hosking, J.G. Constructing component-based software engineering environments: issues and experiences, *Journal of Information and Software Technology*, Vol. 42, No. 2, January 2000, pp. 117-128.
8. Grundy, J.C. and Hosking, J.G. Software Tools, In Wiley Encyclopedia of Software Engineering, 2nd Edition, Wiley Interscience, December 2001.
9. Harrison, W., Ossher, H. and Tarr, P. Software Engineering Tools and Environments: A Roadmap, *The Future of Software Engineering*, Finkelstein, A. Ed., ACM Press, 2000.
10. Meyers, S. Difficulties in Integrating Multiview Editing Environments, IEEE Software, **8** (1), 1991, pp. 49-57.
11. Nentwich, C., Emmerich, W., Finkelstein, A. Static Consistency Checking for Distributed Specifications, In Proceedings of the 2001 Automated Software Engineering Conference, San Diego, November 26-28 2001.
12. Murphy, G., Notkin, D., and Sullivan, K. Software Reflexion Models: Bridging the Gap between Source and High-Level Models. *ACM SIGSOFT Symposium on The foundations of software engineering*. Pp. 18-28, 1995.
13. Nuseibeh, B., Easterbrook, S., Russo, A. Leveraging Inconsistency in Software Development. *IEEE Computer*. Vol. 33, no. 4, pp. 24-29, April 2000.
14. Olsson, T., Runeson, P., Software document use: A qualitative survey, Technical report, Dept. of Communication systems, Lund University.
15. Opdahl, A., Toward a Faceted modelling language, European Conference on Information Systems. Pp. 353–366, 1997.
16. Ossher, H. and Tarr, P. Multi-Dimensional Separation of Concerns and The Hyperspace Approach. In *Proceedings of the Symposium on Software Architectures and Component Technology: The State of the Art in Software Development*, Kluwer, 2000.
17. Rational Corp, Rational Rose™ CASE Tool, www.rational.com.
18. Ramesh, B., Jarke, M. Toward Reference Models for Requirements Traceability. *IEEE Transactions on Software Engineering*. Vol. 27, no. 1, pp. 58-93, January, 2001.
19. Reiss SP. The Desert environment. *ACM Transactions on Software Engineering & Methodology*, **8** (4), Oct. 1999, pp.297-342.
20. Tarr, P., Ossher, H., Harrison, W. and Sutton, S.M. N Degrees of Separation: Multi-Dimensional Separation of Concerns. In *Proceedings of the International Conference on Software Engineering (ICSE 21),* May 1999.