Sixth International Conference on Software Engineering Research, Management and Applications

A Traceability Metamodel for Change Management of Non-Functional Requirements

M. Kassab[1], O. Ormandjieva[2], M. Daneva[3]
[1,2] Concordia University, Canada, {moh_kass, ormandj}@cse.concordia.ca
[3] University of Twente, The Netherlands, m.daneva@utwente.nl

Abstract

Requirements changes are an issue in the software development life cycle which often originates from an incomplete knowledge of the domain of interest. Hardly any requirement manifests in isolation, and usually the provision of one requirement may affect the level of provision of another. Understanding the relations among system requirements is essential to ensuring their consistency and change management. In practice, many organizations either focus their traceability efforts on functional requirements (FRs) or else fail entirely to implement an effective traceability process. Tracing non-functional requirements (NFRs) has, by and large, been neglected. In this paper, we propose a metamodel which explicitly captures NFRs and their relations, and which is independent from any programming paradigm. In addition, we present an implementation using XML-based representations for the metamodel and XQuery queries to represent tracing information.

1. Introduction

In the early phases of software development, user requirements are established based on an analysis of business goals and the application domain. Subsequently, architectures of the desired systems are designed and implemented. During this development process, requirements are usually exposed to many changes as the availability of knowledge on the system being developed increases [9]. Traceability, defined as "the ability to describe and follow the life of a requirement in both a forwards and backwards direction" from inception throughout the entire system's life cycle, provides useful support mechanisms for managing requirement changes during the ongoing change process [24, 25]. Moreover, the extent to which traceability is exploited is viewed as an indicator of system quality and process maturity, and is mandated by many standards [23].

In practice, many organizations either focus their traceability efforts on functional requirements (FRs) [21] or else fail entirely to implement an effective traceability process [13, 15]. Tracing Non-Functional Requirements (NFRs) has, on the whole, been neglected. This is mainly because NFRs tend to become scattered among multiple modules when they are mapped from the one-dimensional requirements domain to the n-dimensional solution space. Furthermore, NFRs can often interact, in the sense that attempts to achieve one NFR can help or hinder the achievement of other NFRs at certain functionality. Such an interaction creates an extensive network of interdependencies and trade-offs between NFRs which is not easy to trace [8]. Nevertheless, reports consistently indicate that neglecting NFRs can lead to catastrophic project failures, or at the very least to considerable delays and, consequently, to significant increases in the final cost. Valid examples are: London Ambulance System (LAS) in 1992 [2], Mars Climate Orbiter in 1998 [17], Therac 25: The Medical Linear accelerator [19] and the Mercedes A-Class (1997) [22].

In this paper, we propose a metamodel which explicitly captures the concepts of NFRs, FRs and their relations throughout the software development process, and which is independent of any programming paradigm. The metamodel can be enhanced to provide additional properties and concepts, and instantiated to define a customized traceability model with respect to the required programming type (e.g. object-orientation, procedural programming, etc.).

While the described here metamodel is a useful way to understand the design structure of the concepts, it is not considered a suitable basis for retrieving data on the objects that are instantiated from this model. Thus we provide an implementation using the eXtensible Markup Language (XML)-based representation for the metamodel. We then use XQuery [29] to implement queries to represent requirements tracing information. XQuery, which is a technology under development by the W3C, provides

978-0-7695-3302-5/08 $25.00 © 2008 IEEE
DOI 10.1109/SERA.2008.37

the means to extract and manipulate data from XML documents or any data source that can be captured in XML, such as relational databases or office documents. XQuery uses XPath expression syntax to address specific parts of an XML document.

In this paper, we identify four critical areas in which NFRs require traceability support:

- Impact of changes to FRs on NFRs (inter-model traceability).
- Impact of changes to NFRs on FRs (inter-model traceability).
- Impact of changes to NFRs on sub-NFRs and parent NFRs (intra-model traceability).
- Impact of changes to NFRs on other interacting NFRs (intra-model traceability).

Tracing queries for each of these areas will be implemented using XQuery. Tracing NFRs against these areas is crucial to the long-term maintenance of critical system qualities such as safety, security, reliability, usability, and performance.

The remainder of this paper is organized as follows: Section 2 provides a brief overview of related work. Section 3 introduces the requirements metamodel. Section 4 presents the XML-based representation and implementation of tracing queries. Section 5 proposes a traceability model using the metamodel and the XML-based representation. Section 6 concludes the paper with an early analysis of the applicability and limitations of the approach and some suggestions for future work.

2. Related work

Although prior work on tracing NFRs has been rather limited, a number of traceability approaches have in fact been developed to support related activities while incorporating NFRs in software engineering processes.

In [18], the authors adopt the NFR Framework [8] to show how a historical record of the treatment of NFRs during the development process can also serve to systematically support evolution of the software system. The authors treat changes in terms of (i) adding or modifying NFRs, or changing their relative importance, and (ii) changing design decisions or design rationale. While this study has provided some support for extensions to the NFR Framework, particularly in representing changes to goal achievement strengths, the impact of changes to functional models on non-functional models, and vice-versa, has yet to be discussed.

In [15, 16], the authors propose an approach named Goal Centric Traceability, a holistic traceability environment which provides systems analysts with the means to manage the impact of functional change on NFRs. Nevertheless, the impact of changes to an NFR on other NFRs and the functional model is not solved with this solution.

In [5, 10, 14], early NFR integration is achieved by extending UML models to integrate NFRs into functional behavior. Although consideration of the compositional process at the meta level is essential for intra-phase traceability, these approaches only model certain NFRs (e.g. response time, security) in a way that is not necessarily applicable to other requirements; specially those with a broad impact on the system as a whole. Indeed, there is no single, formal method available that is well suited to defining and analyzing numerous system NFRs.

Furthermore, researchers in the Requirements Engineering, Product Line Engineering, and Aspect-oriented Software Development fields came up with other initial approaches to address the traceability of NFRs [1, 3, 6, 7, 9, 11, 12, 16, 23, 25, 26, 28]. These approaches have three important limitations. First, tracing is either tackled within a phase or it does not cover the entire life cycle. For example, tracing has been defined within the requirements analysis phase; that is, from requirements to architecture and from architecture to design. Second, the traceability model that is applied is usually focused on specific programming paradigm elements. The selection of tracing properties, however, might be dependent on the requirements of the corresponding project. The traceability model must therefore be sufficiently generic to cope with the various programming approaches. Third, these approaches use coarse-grained entities for tracing purposes thus there is a risk of imprecise change impact analysis, which in turn results in imprecise estimates of the cost and time involved in implementing a requirement change. The specific challenges faced in state-of-the art traceability practice are described in more detail in [23].

This paper offers a solution to the open research problems discussed in this section. Our solution rests on a metamodel which we designed to be well suited for defining and analyzing numerous types of NFRs, the impact of changes to the functional models on NFRs and vice-versa, and the impact of changes to NFR on other NFRs over the entire life cycle.

3. Explicit NFR modeling

Clearly, NFRs can be included in the project in various phases of the life cycle, and traces should be supported within and across life-cycle phases. In order to explicitly reason about the traceability of

NFRs and their refinements throughout the software development process, it is necessary that the corresponding NFRs and their relations be explicitly modeled. In this section, we introduce the requirement relations metamodel, which is schematically represented in the UML domain model in Figure 1. Therein, NFRs are modeled as parts of a requirements group which is a part of a requirements model.

The left-hand side of Figure 1 presents the functional models, and shows that an FR is realized through the various phases of development by many functional models (e.g. in the object-oriented field, a use-case model is used in the requirements analysis and specification phase, a design class model is used in the software design phase, etc.). Each model is an aggregation of one or more artifacts (e.g. a use-case diagram and a use case for the use-case model, a domain model diagram and a system sequence diagram for the analysis model, a class diagram and a communication diagram for the design model). The artifact by itself is an aggregation of elements (e.g. a class, an association, an inheritance, etc. for the class diagram). Modeling artifacts and their elements in this way gives us the option of decoupling the task of tracing NFRs from a specific development practice or paradigm.

The right-hand side of Figure 1 shows the part of our metamodel that is used to model the hierarchy of NFRs and their relations. The decomposition of the NFRs is supported by non-functional models and can be achieved following, for instance, the goal-driven approach. Four relations are identified, namely, association, decomposition, operationalization and interactivity.

Association. NFRs do not represent stand-alone goals, as their existence is always associated with other goals or concepts. In this work, we define three association points with which an NFR and its derived solutions (the so-called operationalizations) can be associated throughout the software development process:

- The FR (and any element belonging to an artifact modeling functionality in any phase): This refers to the context for functionality-related NFRs. For example, associating the *fast response time* NFR with *place order* functionality would indicate that the system must execute the *place order* functionality within an acceptable length of time. If an NFR is associated with functionality, then some or all the offspring elements that refine this functionality will inherit this association. Yet, an NFR could be associated with an offspring element without being associated with the parent functionality.
- Resource: This refers to external-entity-related NFRs. Example of such NFRs would be: *The software maintainers have to have 2 years of experience in oracle database*. This is operating constraint that is associated with candidates for the maintenance position for the system; which are considered as resources for the project.
- Project: This refers to NFRs which provide a precise context to the project or development process. Examples of such NFRs would be: *The project will follow the Rational Unified Process (RUP)* and *The activities X, Y, Z will be skipped for this project*. We make the note that in this category, we also include the association of project characteristics such as *effort* and *productivity* NFRs.

The association relation requires an explicit specification of *association contracts* between the NFRs and the association points. For instance, the association of fast *response time* NFR with *place order* functionality will specify that the functionality is to be executed within the pre-set *Min/Max* delay limits.

Decomposition. This refers to the relation that decomposes a high-level NFR into more specific sub-NFRs. In each decomposition, the offspring NFRs can contribute partially or fully towards satisficing the parent. Let us consider the requirement, *managing transactions with good security*. The *security* requirement constitutes quite a broad topic [8]. To deal effectively with such a requirement, the NFR may need to be broken down into smaller components, so that an effective solution can be found. Thus, *security* can be decomposed into *integrity*, *confidentiality*, and *availability*. The decomposition can be "ANed" (all NFR offspring are required to achieve the parent NFR goal) or "ORed" (it is sufficient that one of the offspring be achieved instead, the choice of offspring being guided by the stakeholders). In the case of "ANed", as in the *security* example, all the sub-NFRs are also associated with the FR with which the parent NFR is associated. For example, the set of association points with *security* is a subset of the set of *confidentiality*, *integrity*, or *availability* association points. In the case of "ORed", then only the sub-NFRs that are selected by stakeholders will be associated with the FRs with which the parent NFR is associated. Figures (2b) and (2c) illustrate the two situations. The question mark notation "?" in (2c) indicates that a further contribution from the stakeholders is required to determine the existence of the relation.

Operationalization. This refers to the relation that refines the NFR into solutions in the target system that will satisfice the NFR. These solutions provide

operations, processes, data representations, structuring, constraints, and agents in the target system to meet the needs stated in the NFRs. Similar to decomposition, operationalization can be ANed or ORed. In the *confidentiality* example, either implementing authorization or the use of additional ID is sufficient, in which case both operationalizations are ORed. We note, however, that the existence of an association between a parent NFR and a FR (e.g. *security* and *place order*) implies that an association exists between those operationalizations which are derived from the parent NFR and the refinement elements derived from the FR (e.g. the use of additional ID and a method implementing the *place order* FR). Figure (2a) illustrates this situation.

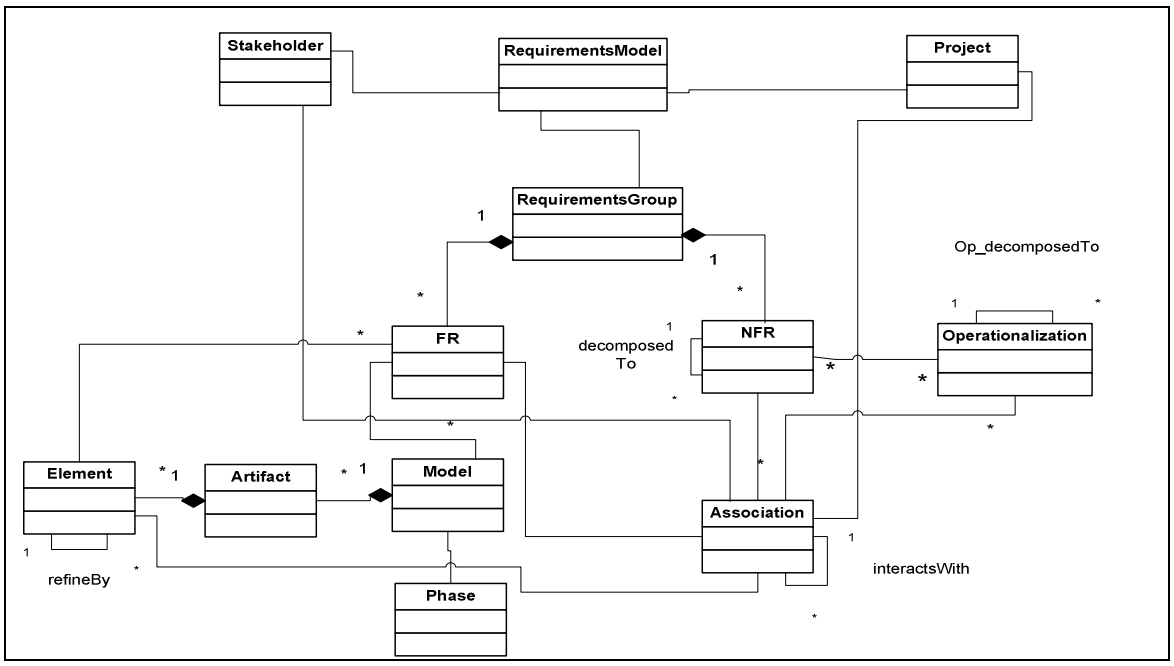

Figure 1: Metamodel for NFRs, FRs, and their relations.

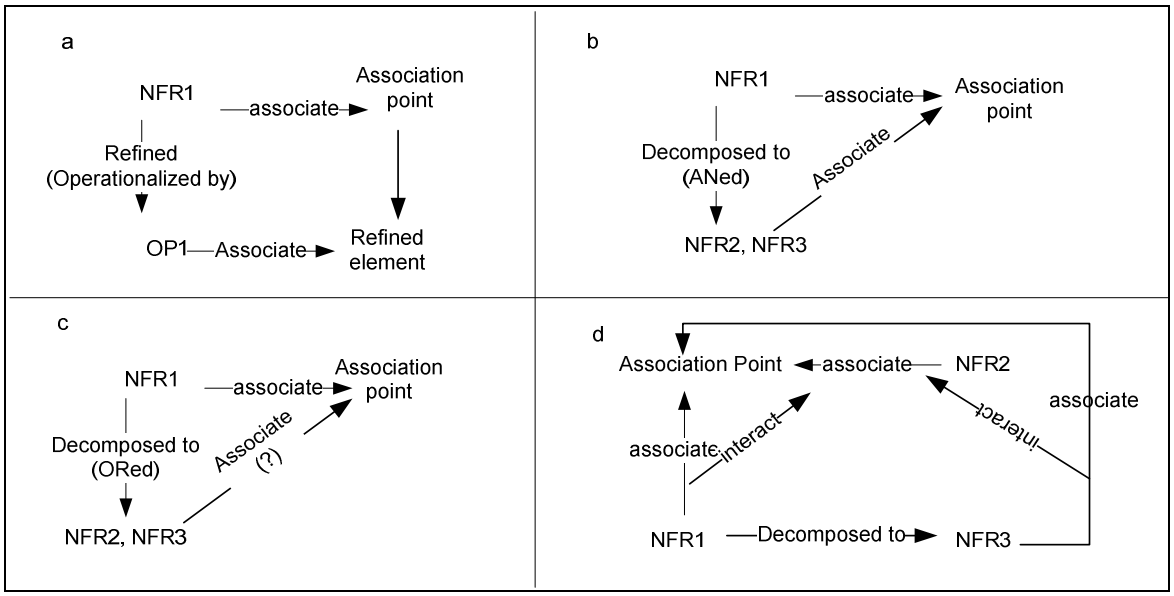

Figure 2: Implicit relations among NFRs and association points.

Interactivity. NFRs by themselves do not interact, as they represent static goals to be achieved. However, their associations with functionalities could interact, in that attempts to achieve one NFR at a certain association point can hinder (negative interaction) or help (positive interaction) the achievement of other NFRs at the same association point, e.g. *security* and *performance* at *place order* functionality. Two NFRs negatively affect each other if they can be traced to the same association point and, at the same time, compete for the same resources. This can be illustrated with the *security* and *performance* example above (the resource is CPU time).

Conflict-resolving algorithms need then be applied to solve the conflicts between negatively interacting NFRs based on an optimization of the resources.

Positive interaction would involve an offspring NFR and its parent NFR in the case of "ANed" decomposition. In "ORed" decomposition, only the sub-NFRs, which are selected by the stakeholders, will positively affect the parent NFR. The interaction is not necessarily a symmetrical relation.

If one association A_1 (between *security* and *place order*, for example) has an interaction with another association A_2 (*performance* and *place order* (see Figure 7)), then at least one sub-NFR refined from NFR_1 (*confidentiality* refined from *security*) has an association (*confidentiality* and *place order*) that interacts with A_2. This situation is generalized in Figure (2d).

4. XML-Based representation and XQuery implementation

While the metamodel described in section 3 is a useful way to understand the design structure of the concepts, it is not considered a suitable basis for retrieving data on the objects that are instantiated from this model. Thus, the model has to be transformed into another model which facilitates querying on the information. In this paper, we use the XML models to instantiate the proposed metamodel and represent tracing information. We instantiate the metamodel by defining the XML-document structure according to the metamodel in the Document Type Definition (DTD) shown in Figures 3 to 5.

```
<!ELEMENT NFRs (NFR+)>
<!ATTLIST NFRs
    name CDATA #REQUIRED
>
<!ELEMENT NFR (NFRname, interaction?, association?,
operationalization?)>
<!ATTLIST NFR
    NFRid ID #REQUIRED
    type CDATA #REQUIRED
>
<!ELEMENT NFRname (#PCDATA)>
<!ELEMENT association (functionalelement | FR,
associationcontract)*>
<!ELEMENT functionalelement (#PCDATA)>
<!ELEMENT FR (#PCDATA)>
<!ELEMENT associationcontract (#PCDATA)>
<!ELEMENT interaction (interactingwith)>
<!ATTLIST interaction
    associationpint CDATA #REQUIRED
>
<!ELEMENT interactingwith (#PCDATA)>

<!ELEMENT operationalization (op)>
<!ELEMENT op (#PCDATA)>
```

Figure 3. DTD structure representation for NFRs.

```
<!ELEMENT FRs (FR+)>
<!ATTLIST FRs
    name CDATA #REQUIRED
>
<!ELEMENT FR (FRname, realization)>
<!ATTLIST FR
    FRid ID  #REQUIRED
>
<!ELEMENT FRname (#PCDATA)>

<!ELEMENT realization (realizingelement+)>
<!ELEMENT realizingelement (realizingelement*)>
<!ATTLIST realizingelement
    realizingelementid ID #REQUIRED
>
```

Figure 4. DTD structure representation for FR.

```
<!ELEMENT NFRDecomposition (RootNFR+)>
<!ATTLIST NFRDecomposition
    name CDATA #REQUIRED
>
<!ELEMENT RootNFR (decomposition)>
<!ATTLIST RootNFR
    NFRid ID #REQUIRED
>
<!ELEMENT decomposition (subnfr+)>
<!ELEMENT subnfr (subnfr*,)>
<!ATTLIST subnfr
    subnfrid ID #REQUIRED
    type CDATA #REQUIRED
```

Figure 5. DTD structure representation for NFR decomposition.

For the purposes of this work, we decided to use XQuery [29] to operate on the data to yield the desired results of tracing information. Queries are very powerful mechanism, since XQuery is a full-

blown functional programming language with strong typing. The evaluation of the query expression reads a sequence of XML fragments or atomic values and returns a sequence of XML fragments or atomic values that are the query result.

The discussion in this section will be illustrated through examples from the invoice system case-study [20]. The invoice system is a web-based system that is capable of receiving multiple orders or cancellation requests at the same time. The system requires its users to have a certain level of privileges to access any of the functionalities except when searching for a product. The privileges are granted automatically upon successful authentication. In this work, we will limit the scope of the discussion to two functionalities, namely, *view orders* and *place order*. Figure 7 (see Appendix 1) presents these two main pieces of functionality decomposed into elements of use-cases, events and methods. Three NFRs are also presented: *security, performance and scalability*.

While recording data on the captured requirements and its relations, it is hard to ensure the completeness of the data as the majority of the instances of the relations are not directly stated by stakeholders but they hold as valid relations by induction. For example, *security* could be known as being associated with *place order* functionality. *Confidentiality* which is derived from "ANed" decomposition from *security* is also associated with *place order* according to figure (2b). This information on *confidentiality* association could be missed when recording the data on the NFRs (and generating the corresponding XML document) yet this relation has to be traced upon possible related requested changes in requirements. Our tracing mechanism considers this situation and is implemented so that it provides the suitable solution.

We identify four critical areas in which NFRs require traceability support. These areas are discussed in the following sub-sections.

4.1 Impact of changes to functional models on NFRs

When a change is initiated in a FR, the set of NFRs potentially affected needs to be identified and retrieved. This is accomplished by first retrieving all the directly associated NFRs to the changed FR:

```
//FR_CHANGED refers to ID of the changed functionality.

<result>
{
for $x in doc("NFRs.xml")/NFRs/NFR
where $x/association/FR = "FR_CHANGED"
return data($x/@NFRid)
}
</result>
```

In order to ensure the completeness of the trace and the consistency among requirements, it is important that all NFRs associated with all elements derived from the changed FR be analyzed as well. This should be done in a recursive manner to cover all possible derived elements.

```
<result>
{
for $c in (
for $x in doc("FRs.xml")/FRs/FR
where $x/@FRid = "FR_CHANGED"
return data($x/realization/realizingelement/descendant-or-
self::realizingelement/@realizingelementid))
for $b in doc("NFRs.xml")/NFRs/NFR
where $b/association/functionalelement = $c
return data($b/@NFRid)
}
</result>
```

In the case study of the Invoicing System (see Figure 7, Appendix), if a change is requested to *place order* functionality, then the above two query expressions will retrieve *security*, *performance*, and *scalability* as potentially impacted NFRs.

4.2. Impact of changes to non-functional models on functional models

To complete the inter-model traceability, we should complement the query in 4.1 by considering the impact of changes to NFRs on the functional model. When a change is initiated in an NFR, then the set of all association points of the FR type or the element type should be retrieved and analyzed against the potential change.

```
<result>
   {for $x in doc("NFRs.xml")/NFRs/NFR
  where $x/@NFRid ="NFR_CHANGED"
return data($x/association/FR union
$x/association/functionalelement)
  }
</result>
```

In the Invoicing System case study (see Figure 7, Appendix), if a change is requested to a *security* requirement, then the above query expression will retrieve *place order* functionality, use $case_3$, use $case_4$, and the events *select a product* and *place a payment*, and the methods *placeOrderSession.makeOrder* and *orderCatalogue.makeOrder*.

4.3 Impact of changes to NFRs on lower/ higher-level NFRs

The change to one NFR can be propagated down to offspring NFRs or up to parent NFRs in a recursive manner through the decomposition links. This type of traceability enables the analyst to understand the impact of lower-level change on high-level goals, and vice versa. The following XQuery expression implements the tracing queries for the downward direction:

```
//NFR_CHANGED refers to ID of the changed NFR.

<result>
{
for $x in doc("NFRs.xml")/NFRDecomposition/RootNFR
where $x/@NFRid = "NFR_CHANGED"
or $x/decomposition/subnfr/@subnfrid =
"NFR_CHANGED"
return $x
}
</result>
```

In the Invoicing System case study (see Figure 7, Appendix), if a change is requested to a *space* requirement, then the above query expression will retrieve the *primary space*, *secondary space*, and *performance* requirements.

4.4 Impact of changes on interacting associations

To complete intra-model traceability, it is necessary to establish traces between interacting NFRs at certain association points (interacting associations). The following XQuery expression implements this query:

```
<result>
   {for $x in doc("NFRs.xml")/NFRs/NFR
  where $x/@NFRid ="NFR_CHANGED"
return data( $x/interaction/interactingwith)
  }
  ,
  {
   for $x in
doc("NFRs.xml")/NFRs/NFR/interaction/interactingwith
where $x = "NFR_CHANGED"
return data($x)
  }
</result>
```

In the Invoicing System case study (see Figure 7, Appendix), if a change is requested to a *performance* requirement on *place order* functionality, then the above query expression will retrieve the *security* requirement on that functionality. If a change is requested to a *security* requirement, then the *performance* requirement will be retrieved.

Requirements change management requires not only an analysis of the impact of a given change but also a change authorization mechanism to ensure the consistency of the proposed changes with the remaining traceability model. In what follows, we address this need.

4.5 Change authorization

Change to an NFR or FR would lead eventually to a substitution of part of the existing relations in the traceability model with the implied by the change new associations and interactions. Such change can be authorized if and only if it the substitution is consistent with the existing traceability model relationships. For instance, a change affecting an association relation has to conform to the corresponding association contract. This can be illustrated on the security NFR which affects negatively the response time NFR. Increasing the security level would lead to decreasing response time, which still has to comply with the Min/Max delay association contract between the *response time* NFR and the *place order* functionality. A decision on any accepted change in any of the retrieved data should be recorded in the corresponding XML document.

It is important to note that one change request can establish a chain of other requests. For example, the need to change one FR may generate the need to accept changes to other NFRs. In response to the NFR changes, the analysts may well see a need to change further sub-NFRs or interacting NFRs. We are currently working on establishing a formal algorithm for the change authorization process.

5. Traceability Model

NFR tracing occurs through three distinct activities: requirement development, impact detection, and evaluation/decision-making. Each phase ensures that FR and NFRs are treated jointly and in an integrated fashion. These activities are depicted in Figure 6.

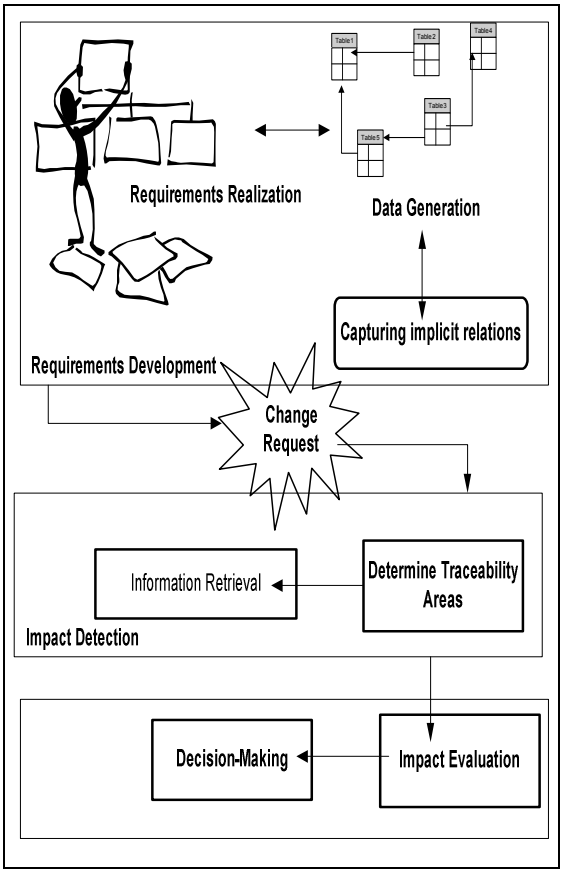

Figure 6. NFR-tracing activities

FRs capture the intended use of the system. This use may be expressed as responsibilities, services, tasks, or functions which the system is required to perform [30]. Identifying FRs is a process which involves discussions with stakeholders, arranging requirement elicitation workshops, building prototypes, and negotiating both FRs and NFRs. The FRs are then specified through various analysis and design models [30]. NFRs which are relevant to the problem domain are captured at the same time that the FRs are identified. A popular approach to capture NFRs along with their interactions, associations, and decompositions includes the adoption of NFR catalogs [8] with historical data on possible relations of NFRs in a similar problem domain. The corresponding XML document should be generated with the captured data.

The NFRs and their relations explicitly stated above are further analyzed to capture implicit NFR relations which are not explicitly stated by the stakeholders (see Figure 2). The recorded data should then be maintained by adding new entries for any new discoveries on relations.

Impact detection is dependent on the effectiveness of the traceability mechanism in establishing correct links between functional and non-functional models and within their corresponding hierarchical models. Triggered by a change request, the potentially impacted area has to be identified, and then the corresponding query should be executed. Once the retrieval algorithm has returned a set of potentially impacted requirements/elements, the evaluation phase can commence. To analysts, this means they can now filter the retrieved requirements/elements to remove any non-relevant ones.

6. Conclusion

The tendency for NFRs to have a wide-ranging impact on a software system, and the strong interdependencies and tradeoffs that exist between NFRs and the software architecture, leave typical existing traceability methods incapable of tracing them. In this paper, we propose a metamodel for requirement relations, and we propose a traceability mechanism under the umbrella of XML models to track the allocation of requirements to system components, and control changes to the system. One of the advantages of our approach is that it forces system analysts to think about and capture the hierarchical relations within NFRs, the hierarchical relations within FRs, and the relations between NFR and FR hierarchies. Our approach helps systems analysts understand the relationships that exist within and across NFRs in the various phases of development. The paper proposes a method for tracing a change applied to an NFR in the traceability model, which results in a "slice" of the model containing all model entities immediately reachable from that NFR within the hierarchy.

In addition, change management would require not only a mechanical tracing of the effects of change, but also a reasoned approach to gauging the consistency of the changes within the traceability model. Due to the complexity of the above relations in the traceability model, a change analysis mechanism is required to ensure the consistency of the proposed changes before they are authorized. Our future work includes the development of consistency rules based on the formal presentation of the FR and NFR hierarchies and their relations, rules which will be automatically checked before a change is authorized.

As this evaluation is an early one, our approach has not been validated using an empirical research method [27]. We plan to remedy this by carrying out case studies. At this time, we have provided a "proof-

of-concept" (as it is called in [27]), and can only give an early assessment of some threats to the validity of our approach and discuss what could be done to address them. We have considered the question of what types of projects would be best served by using our approach. The invoice management system used for illustration purposes in this paper is only representative in terms of its size and level of complexity for a small information systems project. It remains to be seen to what extent our experience with our approach as applied to this application can be scaled to other project settings, for example business information systems or large embedded system projects. We believe, however, that the Invoicing System is a typical example of a business information systems project, and so our results are transferrable. Nevertheless, further empirical investigation must take place to confirm or refute this.

References

[1] A. Egyed, P. Grunbacher, "Identifying Requirements Conflicts and Cooperation: How Quality Attributes and Automated Traceability Can Help", *IEEE Software*, 21(6), 2004, p.p.50-58.

[2] A. Finkelstein, and J. Dowell, "A Comedy of Errors: The London Ambulance Service Case Study", *Proc. Eighth International Workshop Software Spec and Design*, 1996, pp. 2-5.

[3] A. Finkelstein and W. Emmerich, "The Future of Requirements Management Tools", *In Information Systems in Public Administration and Law*, 2000.

[4] A. M. Salem, "Improving Software Quality through Requirements Traceability Models", *Proc. of Int. Conf on Computer Systems and Applications*, 2006, pp. 1159- 1162.

[5] A. Moreira, J. Araujo and I. Brito, "Crosscutting Quality Attributes for Requirements Engineering", *In 14th Int. Conf. on Soft. Eng. and Knowledge Eng. 2002*, Ischia, Italy, 2002, pp. 167-174.

[6] B. Ramesh, and M. Jarke, "Toward a Reference Model for Requirements Traceability", *IEEE Transactions on Software Engineering*, 27(1), 2001, pp. 58-93

[7] C. Hofmeister, R.L. Nord, D. Soni, "Global Analysis: moving from software requirements specification to structural views of the software architecture", *IEEE Proceedings -- Software*, Vol. 152 Issue 4, Aug2005, pp.187-197.

[8] Chung L., Nixon B.A. , Yu E. and Mylopoulos J., *Non-functional Requirements in Software Engineering*, Kluwer Academic Publishing, 2000.

[9] D. Jacobs, "Requirements Engineering so Things Don't Get Ugly", *Companion to the Proc. of 29th Int. Conf. on Software Engineering*, 2007, pp. 159-160.

[10] E. Baniassad, S. Clarke, "Theme: An Approach for Aspect-Oriented Analysis and Design". *26th International Conference on Software Engineering (ICSE 2004),* 23-28 May 2004, Edinburgh, UK. IEEE Computer Society 2004, pp. 158-167.

[11] E. Baniassad, P.C. Clements, J. Araujo, A. Moreira, A. Rashid, B.Tekinerdogan, "Discovering Early Aspects", *IEEE Software*, Jan-Feb 2006, 23(1), pp. 61-70.

[12] E. Niemelä, A. Immonen, "Capturing Quality Requirements of Product Family Architecture", *Information and Software Technology*, Vol. 49, Issues 11-12, November 2007, pp. 1107-1120.

[13] F. A. Blaauboer, K. Sikkel, M.N Aydin, "Deciding to Adopt Requirements Traceability in Practice", In *Proc. of 19th Int. Conf. on Advanced Information Systems Engineering (CAiSE'07)*, Springer *Lecture Notes in Computer Science 4495*, Norway, 2007, pp. 294-308.

[14] J. Araujo, A. Moreira, I. Brito, and A. Rashid, "Aspect-Oriented Requirements With UML", *Workshop on Aspect-Oriented Modeling with UML (held with UML 2002,* Dresden, Germany, 2002.

[15] J. Cleland-Huang, "Toward Improved Traceability of Non-Functional requirements", *Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering*, Long Beach, California, 2005, pp. 14 – 19.

[16] J. Cleland-Huang, R. Settimi, O. BenKhadra, E. Berezhanskaya, S. Christina, "Goal Centric Traceability for Managing Non-Functional Requirements", *Proceedings of the 27th international conference on Software engineering*, 2005, pp. 362 – 371.

[17] K. K Breitman, J. C. S. P. Leite and A. Finkelstein, "The World's Stage: A Survey on Requirements Engineering Using a Real-Life Case Study", *Journal of the Brazilian Computer Society No 1 Vol. 6*, Jul. 1999, pp.13-37.

[18] L. Chung, B.A. Nixon, E. Yu, "Using Non-Functional Requirements to Systematically Support Change", *Proceedings of the Second IEEE International Symposium on Requirements Engineering*, 1995, York, U.K., pp. 132 – 139.

[19] L. Leveson, and C.S. Turner, "An Investigation of the Therac-25 Accidents", *IEEE Computer.,* 26, 7, 1993, pp.18-41.

[20] M. Sighireanu, K.J Turner, "Requirement Capture, Formal Description and Verification of an Invoicing System", Research Report, available at

http://www.inrialpes.fr/vasy/Publications/Sighireanu-turner-98.html

[21] M. Weber, J. Wesbrot, "Requirements Engineering in Automotive Development: Experiences and Challenges", *IEEE Software* 20(1), pp.16-24.

[22] Mercdes A-Class: "Mercedes: Wie sicher ist die A-Klasse?", German news magazine: "Der Spiegel", ISSN 0038-7452, Oct. 27th 1997, p. 120; English translation: http://www.geocities.com/MotorCity/downs/9323/aclacap.htm, last visited on Feb. 11th, 2005.

[23] N. Aizenbud-Reshef, B.T. Nolan, J. Rubin, Y. Shaham-Gafni , "Model Traceability", *IBM System Journal*, 45(3), 2006, pp. 515-526.

[24] O. Gotel, "Contribution Structures for Requirements Traceability", London, England: Imperial College, Department of Computing, 1995.

[25] O. Gotel and A. Finkelstein, "An Analysis of the Requirements Traceability Problem", *Proc. First International Conf. Requirements Eng.*, Colorado, U.S.A, 1994, pp. 94-101.

[26] P. Letelier, "A Framework for Requirements Traceability in UML-Based Projects", *Proc. of the 1st Int. Workshop on Traceability in Emerging Forms of Software Engineering*, Edinburgh, 2002, pp. 30–41.

[27] R.J. Wieringa and J.M.G. Heerkens, "The methodological soundness of requirements engineering papers: a conceptual framework and two case studies", *Requirements engineering*, 11(4), 2006, pp. 295-307.

[28] V. Winter, H. Siy, M. Zand, P. Aryal, Aspect Workshop at AOSD'06, Bonn, Germany, 2006.

[29] W3C XML Query (XQuery): http://www.w3.org/XML/Query/

[30] Wieringa, R.J., *Design Methods for Reactive Systems*, Morgan Kaufman, NY, 2001

Appendix 1

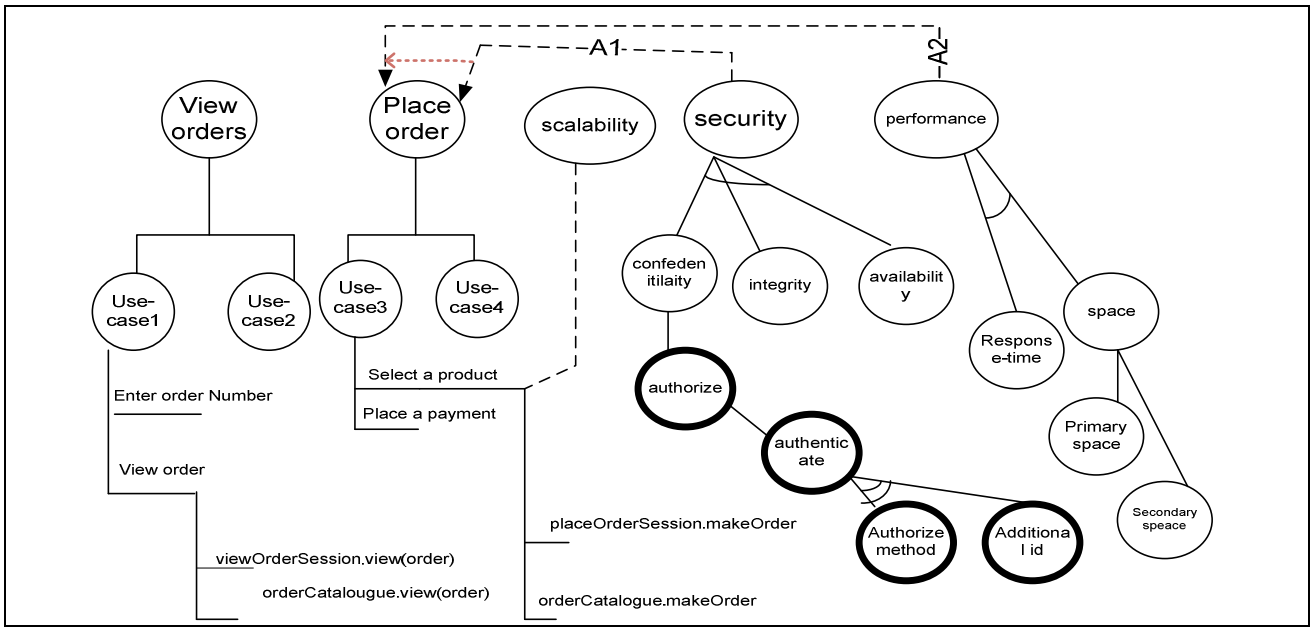

Figure 7. Illustration of FR and NFR relations through the Invoicing System case study.