NDT-Merge: A future tool for conciliating Software Requirements in MDE Environments

J. A. García-García
IWT2 Reserch Group.
University of Seville, Spain
ETS Ingeniería Informática
Av. Reina Mercedes S/N
41012 Seville
julian.garcia@iwt2.org

E. Ravel
IWT2 Research Group.
University of Seville, Spain
ETS Ingeniería Informática
Av. Reina Mercedes S/N
41012 Seville
eric.ravel@insa-lyon.fr

M. Urbieta
LIFIA, Facultad de Informática,
UNLP, La Plata, Argentina
Facultad de Informática - UNLP - 50 y 120 - La Plata, Argentina
murbieta@lifia.info.unlp.edu.ar

M.J. Escalona
IWT2 Research Group.
University of Seville, Spain
ETS Ingeniería Informática
Av. Reina Mercedes S/N
41012 Seville
mjescalona@us.es

G. Rossi
LIFIA, Facultad de Informática,
UNLP, La Plata, Argentina
Conicet
Facultad de Informática - UNLP - 50 y 120 - La Plata, Argentina
gustavo@lifia.info.unlp.edu.ar

ABSTRACT
Requirements conciliation can result one of the most expensive and critical tasks in Web development. It particularly depends on analysts' experiences since very frequently they have to use manual solution to cope with requirements conciliation. After some previous work presenting an approach oriented towards Web requirements conciliation, this paper proposes a tool for executing this task. It is based on the Model-driven paradigm and it is included in the context of NDT (Navigational Development Techniques) methodology.

Categories and Subject Descriptors
Documentation – *requirements, conflict detection, Model-driven paradigm.*

General Terms
Algorithms, Management, Documentation, Verification.

Keywords
Requirements, conflict detection, Model-driven paradigm, tool

1. INTRODUCTION

In the information system development process, led or not to the Web, the development team faces up the arduous task of defining system requirements. It is a complex process since it must identify the requirements that the system have to meet in order to satisfy end users and customers' requests. Requirements elicitation of any Web application implies understanding the needs of different groups of stakeholders who, in one way or another, will use the final system. Normally, requirements are agreed by stakeholders so that the semantics and meaning of each business term are well defined and understood. However, if there are different points of view on the same business concept [1], ambiguities and/or contradictions in requirements specification may arise. This may act to the detriment of requirements specification.

Traditionally, conciliation tasks are performed through meeting-based techniques [2] in order to eliminate requirements ambiguity and contradictions. The fact that requirement inconsistencies are not detected on time may entail defects in the Web software. In this context, the effort to correct the faults is several orders of magnitude higher than correcting requirements at the early stages [3][4] of the project.

This paper is contextualized within the line of research analyzed in previous work [9], which described the improvement of techniques for conciliating requirements in multidisciplinary teams. In this work we applied requirement validation concepts in a real project by means of NDT (Navigational Development Techniques) [5] framework and its tool support (NDT-Suite [6]). Through this work we explore the mechanisms offered by the Model-driven paradigm for this purpose and assess the results of implementing them with NDT. In addition, we suggest extending NDT-Suite with a new tool which will automatically analyze the conflicts in requirements specification and will propose solutions to resolve them.

This paper is structured as follows. Section 2 gathers some related work on requirements validation. Section 3 offers a global vision of NDT and we focus on its requirements model. In addition, Section 3 offers conflicts characterization and analysis. Section 4 lays the foundations for the implementation of NDT-Merge tool. In this section, we explain various aspects of NDT-Merge: its architecture, what is the procedure for identifying conflicts and how to resolve these conflicts detected. In addition, we present an example to show how the procedure works. Section 5 presents our main conclusions and suggests further work in this field. Finally, we include an Annex at the end of this paper. This annex provides

iiWAS2012, 3-5 December, 2012, Bali, Indonesia.

a description of the requirements that we have used in the example presented in Section 4.

2. RELATED WORK

Silva & Do Santos [11] propose the use of Petri Nets as a specific technique to validate the consistency of requirements defined as use cases. This approach generates Petri Nets from use cases and studies their consistency. It seems quite interesting as it tries to normalize requirements validation with an important constraint, as it is oriented to use cases described with a very specific notation. This technique cannot be used, if any special extension of use cases operates or other techniques to describe requirements are applied.

In the Web Engineering field, the situation is not different. Despite some methodologies improved their requirements phase in the last years, the study of the requirements has remained too "handcrafted" and non-systematized yet. Thus, recently, some Web design approaches, such as WebML [12] and NDT, support this idea by means of the Model-driven paradigm. Nevertheless, even offering systematic (or even automatic) support for early testing, the detection of inconsistencies in the specification of requirements is still too "handcrafted" and depends on the analysts' experience and their capability to support the review with customers and users.

Focusing only on the detection of conflicts, [13] presents an approach to identify concerned conflicts. The authors propose using a Multiple Criteria Decision Making method to support aspectual conflicts management in aspect-oriented requirements. It results limited since it points out the treatment of aspect-oriented requirements and it only deals with concerned conflicts.

In other phases of the lifecycle, the conflict-detection process has been deeply studied by the Model-driven community mainly based on UML model conflicts. In [14] the author suggests identifying conflicts in a twofold process: analyzing syntactic differences by raising candidate conflicts and understanding these differences from a semantic view. In [15] is presented approach based reasoning on logic descriptor. In this approach, UML models are transformed into logic descriptor documents that are later processed by a first-order logic engine in charge of reasoning. To our knowledge, our proposal is the first Model-driven approach for validating Web application requirements.

3. BACKGROUND

3.1 NDT: An overview

Navigational Development Technique (NDT) is a Model-driven Web methodology initially defined to deal with requirements on Web applications developments. NDT has evolved in the last years and nowadays it offers a complete support for the whole development lifecycle. It covers the viability study, requirements treatment, analysis, design, construction, implementation, as well as the maintenance and testing phases of software development. Regarding the requirements phase, NDT classifies project requirements according to their nature: information storage, functional requirements, actor requirements, interaction requirements, and non-functional requirements. The following these requirements will be described.

The information storage requirements are divided into storage requirements (RA) and new natures (NA). Information storage requirements define what information will be stored in the system and the relationships established between such information. Each Information storage requirement has associated a number of specific data, which represent the items of information that the requirement will store. The concept of new nature differs from the data type concept. Nature represents a domain as a set of values that have a specific meaning within the system without going into low-level details.

Actor requirements (AC) define what user roles can interact with the system and the relations established between them.

Functional requirements (RF) describe the needs of functionality offered by the system. The functional requirements respond to the question of *what the system can do?* To capture and definition of functional requirements, NDT uses the use case diagrams to graphically represent the functionality of the system. NDT proposes to accompany these diagrams with additional textual information. All this information is registered in the pattern that this methodology proposes to gather the information of the functional requirements.

Interaction requirements define how each user role interacts with the system and how you can navigate through it. Interaction requirements will be defined by Phrases (FR) and Prototypes of Visualization (PV). The Phrases are used to describe how the information will retrieved, whereas Prototypes of Visualization are used to describe how information is displayed in the system, how the user navigates through the system and how the functionality will be offered to the user.

Finally, non-functional requirements (RNF) are used to catalogue any other needs of the system which cannot be classified according to the above requirements. Non-functional requirements can be used to register the following requirements: technical requirements for communications (for example, the protocol used for communications system), reliability requirements, requirements development environment (for example, operating system), portability requirements, accessibility requirements, etc.

In order to define these requirements, NDT provides special patterns and UML techniques, such as the use cases technique for functional requirements specification.

On the other hand, NDT supports a set of processes to bear out project management and quality assurance and it is globally complemented by a set of free tools grouped in the NDT-Suite [6]. This suite enables the definition and use of every process and task supported by NDT and offers relevant resources to develop software projects in terms of quality assurance, management and metrics. Currently, the suite of NDT comprises the following free Java tools:

- NDT-Profile is a specific profile for NDT developed using Enterprise Architect [19]. NDT-Profile offers the chance of gathering all the artifacts that define NDT easily and quickly, as they are integrated within the tool Enterprise Architect.
- NDT-Quality [20] is a tool that automates most of the methodological review of a project developed with NDT-Profile. It checks both, the quality of NDT methodology in each phase of software lifecycle and the quality of traceability of the MDE rules of NDT.

- NDT-Driver [21] is the key tool for carrying out the transformations among NDT models. It implements a set of automated procedures that enables to perform all transformations MDE among the different models of NDT that were described in the previous section. The data source to use this tool is a project developed with NDT-Profile. For practical purposes, this tool considerably minimize the time spent in the design and development of models from different life cycle phases of NDT, as the basic models it obtains provide the analysts team with a starting point.
- NDT-Prototype is a tool designed to automatically generate a set of XHTML prototypes from the navigation models of a project, described in the analysis phase, developed with NDT-Profile.
- NDT-Glossary [22] implements an automated procedure that generates the first instance of the glossary of terms of a project developed by means of NDT-Profile tool.
- NDT-Checker is the only tool in NDT-Suite that it is not based on the MDE paradigm. This tool includes a set of sheets different for each product of NDT. These sheets give a set of checklists that should be manually reviewed with users in requirements reviews.
- NDT-Counter is a tool that, using use cases points, estimates the effort of a project that is going to be developed with NDT.

In the last ten years, NDT and NDT-Suite were used in a high number of real projects. In fact, NDT-Suite is currently being used in several projects developed by different companies, either public or privates. Public companies such as the Regional Cultural Ministry or the Regional Health Ministry of Andalusia, among others, are working with NDT and NDT-Suite. Private ICT companies in Andalusia are also using NDT in some of theirs projects. The use of NDT and NDT-Suite in numerous projects has provided us with an important feedback. One of these projects was the Mosaico project [7] which was developed for The Regional Cultural Ministry of Andalusia. The idea of this Web system was born from the need of managing all the information on historic heritage in Andalusia. Mosaico was developed by two important companies and it covered 5,670 requirements, out of which 3,253 were functional requirements.

From the experience of this project we know that requirements are difficult to conciliate in projects involving multiple teams. This paper proposes improving the NDT methodology to solve these problems during conciling requirements. In addition, this article proposes extending NDT-Suite with a new tool that will automatically analyze conflicts in requirements specification and will propose solutions to solve these conflicts.

3.2 Characterization and analysis of conflicts

During requirement specification, there may be cases where two or more scenarios that reflect the same business logic differ subtly from each other producing an inconsistency. When these inconsistencies are based on contradictory behaviours, we are facing a requirements conflict [8]. Conflicts are characterized by differences in the features of object, differences between logical (what is expected) or temporal (when it is expected) conflicts of actions, or even differences in terminology that provoke ambiguity.

In this analysis we will emphasize Web application navigation, as well as users' interaction peculiarities that are not covered in the traditional characterization of requirement conflicts [8]. Consequently, we provide an interpretation of each conflict type on the Web application environment: (*i*) Structural conflicts stand for a difference in the data expected to be presented on a Web page by different stakeholders. A stakeholder may demand that data be shown on a Web page to contradict other stakeholder's requirement; (*ii*) Navigational conflicts take place when two Web application requirements may contradict the way in which links are traversed producing navigational conflicts; (*iii*) Semantic conflicts occur when the same real-world object is described with different terms. This situation may generate a false negative in the conflict detection process, since a conflict may not be identified and new terms are introduced into the system space thus increasing its complexity. As a consequence, the same domain object is modelled in two entities with different terminology.

Our approach allows identifying, analyzing and solving conflicts and was introduced in [9] through WebSpec [10] as a Web requirement metamodel. Next, we will present a brief summary of our approach shown in Figure 1:

1. *Requirement gathering*. A Software Requirement Specification (usually in natural language) is produced by means of well-known requirement elicitation techniques such as meetings, surveys or Joint Application Development (JAD), among others.
2. *Requirement modelling*. Web application requirements are formalized by using a requirement Domain Specific Language (DSL) (e.g. WebSpec or NDT) giving a formal requirement model as a result.
3. *Structural analysis of the Web requirements model*.

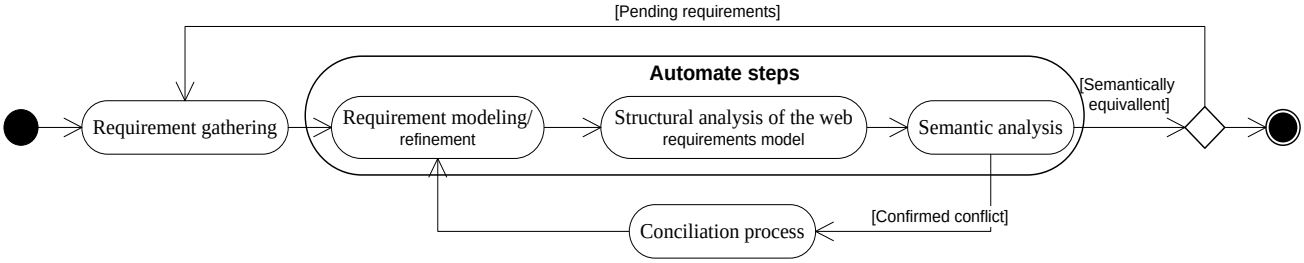

Figure 1. The Overall Process for Detecting Requirement Conflicts

Structural and navigational conflicts of a candidate are identified by means of an algebraic comparison of requirement models obtained in step 2. Additionally, navigation paths are evaluated in order to check their consistency.

4. *Semantic analysis*. Candidate conflicts are analyzed and semantic equivalences are detected. For each candidate conflict, both the new requirement and the compromised requirement are translated from a high abstraction level (the requirements DSL) into a minimal form by using simple elements so as to detect semantic differences.
5. *Conciliation process*. Once the existence of a conflict is confirmed, we must start conciliating requirements. This process demands the establishment of a communication channel among those stakeholders concerned in the conflict.
6. *Refinement*. When a conflict is confirmed some adjustment and tuning must be done in order to remove the identified conflict and reach a consistent state.

The process is applied iteratively each time a new set of requirement emerges. The new incoming set of requirements is checked with each of the already consolidated requirements of the system space.

4. NDT-Merge

In this section, we reflect about the future NDT-Merge tool. NDT-Merge provides developers the opportunity to merge the two requirement phases of projects developed using NDT-Suite, which represent the same project developed by separate teams. The software is still being developed but we give here the main aspects.

4.1 Overall Architecture

Figure 2 show the proposed architecture of NDT-Merge. To respect conventions and facilitate the development, the basic architecture supposes three levels: Data Access, Business Logic and Presentation.

The Presentation Level contains the user interface which mainly presents to users who can choose the options for the treatments. With this interface, the user can configure the tool to select for instance a subset of NDT types of requirements to work on. These options put conditions to the conflict detector which is part of the next level in order to merge only selected requirements aspects of two projects. We give here the types of requirements used in NDT.

The Business Logic Level contains two packages permitting conflict detection and their resolution. The detector module will implement a generic architecture which will be able to deal with any type of requirement. This module is responsible of the detection of structural, link and semantic conflicts occurring in the whole requirements phase. In the following subsection, an example of the algorithm is explained. The detector uses the two project's data, using the lower level to get information from the databases. It establishes what objects, from the two projects, in each NDT model represent a same concept using a well spread semantic analysis described below, this allows establishing the structural differences between the two objects. Then, the resolver module deals with the conflicts to resolve them.

The Data Access Level contains the NDT-Access module which contains classes providing access services to the database. We use this tool as a low level unity which allows upper processes to get any type of object needed. In our case, only requirements will be accessed.

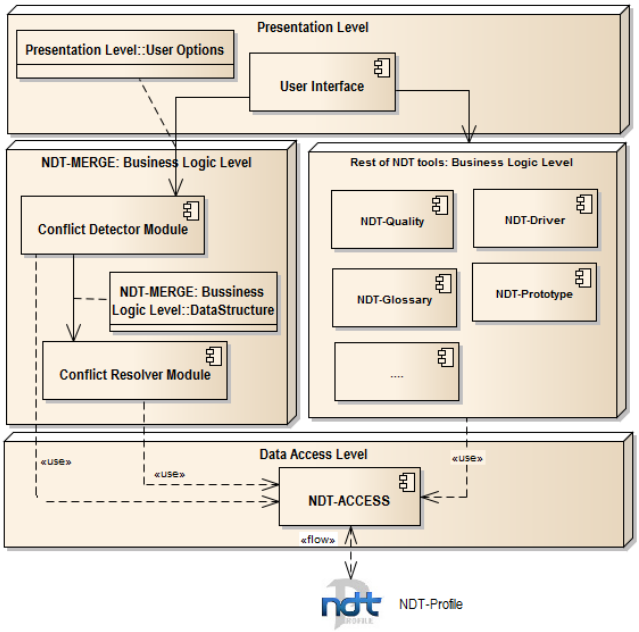

Figure 2. Architecture of NDT-Merge

4.2 Conflict identification procedure

4.2.1 Conflicts in NDT

The structural analysis of the models proposed in [10] supposed that two requirements to be compared have the same identifiers, the name. We try to deal in a more general case supposing that the objects representing the same concept can have different name. For example: "Product" and "Good" could represent the same concept in a particular context. Therefore a process is needed to match objects from one version to the others, and link them to a same concept. We use the description, one of the properties offered by NDT to characterize objects.

We consider three types of conflicts. The most difficult to solve solve is the *semantic conflict*; it occurs when two requirements, or components of requirements, have same of rather identical names but describe different objects. The *link conflict* represents the difference of related object lists between two objects describing the same requirement and it is like a generalization of the *navigational conflict* described in Webspec [9]. The *structural conflict* represents any other difference between the two objects representing the same requirement. It can be then name, the inner components, the various properties. Even the description has to be marked as contentious, except if it was exactly the same, in order to make a choice later in the resolution. We give level of severity for the conflicts. A *semantic conflict* is always at the high level because it can create misunderstandings quickly. The *link conflict*

has a medium severity and for the structural conflict it depends on the level of the conflict and can be from the lowest severity to the highest one.

4.2.2 Semantic Conflict Analysis

The identification of same concepts depends on the analysis of the objects description. We carried out the analysis of text using the technique described in [16] and [18]: the vector space model. This technique has been used in a similar manner in [17], but in our paper we use the statistic *term frequency-inverse document frequency*. This technique associates a mathematical equivalence to any text, i.e., n-dimensional vector where n is the numbers of terms of the text. Each component stores the weight of each term. This weight of each word is calculated by the multiplication of two parameters: $tf * idf$. On the one hand, tf indicates the frequency of the word in the text, i.e., the number of occurrences of the term in the text divided by the total number of terms in the text. On the other hand, idf is the inverse document frequency, and it evaluates the importance of the considered term in the whole set of descriptions. Its definition allows giving a greater weight to the less frequent terms, which are considered as the most characteristic words. It is calculated by taking the logarithm of the quotient obtained by dividing the number of descriptions by the number of descriptions that contains the term. Then, the mathematical expression of idf is presented.

$$idf(t, D) = log \frac{|D|}{1 + |\{d \in D : t \in d\}|}$$

With:

- $|D|$. D is the corpus or set of descriptions analysed and $|D|$ is the number of descriptions in the corpus.
- $1 + |\{d \in D : t \in d\}|$ This mathematical expression represents the number of descriptions in which the term t appears. The number of descriptions where the term t appears. This expression avoids a division-by-zero in the case in which the term would be absent.

Finally, the mathematical expression of $tf * idf$ is presented:

$$tf * idf(t, d, D) = tf\ (t, d) * idf(t, D)$$

4.2.2.1 An Example

In our example, we merge two projects developed with NDT-Profile and we only focus on the model information storage requirements (RAs and NAs). Figure 3 shows the information storage requirements of the first project and Figure 4 shows the information storage requirements of the second project.

All the requirements that we use in this example are described in detail in the Annex.

The similarity of the descriptions is evaluated considering that descriptions are vectors of words. Since we consider vectors we have to apply a single order of words. All the words of the whole set of descriptions have to be considered and each new one is a new dimension in the vector. Then, the description's original order is not relevant, it is only necessary to have all the words.

After building the two vectors (one for each description or text), we can know what is the similarity between the two descriptions. For this, we apply the cosine to calculate of the angle between two vectors. The cosine with value 1 implies that the angle between the vectors is 0, which implies that the texts are similar.

$$cos(\alpha) = \frac{V1.V2}{||V1||.||V2||}, (0 \leq cosine \leq 1)$$

To apply the technique described, first of all, the words are stemmed to their roots so that plurals, verbal forms or other forms are not considered. We also don't consider pronouns, articles and other connexion terms. Then, the cosine similarity described in [16] is applied; the algorithm calculates cosines between two vectors. Therefore we understand that all the relevant words of the corpus have to be represented in the vectors.

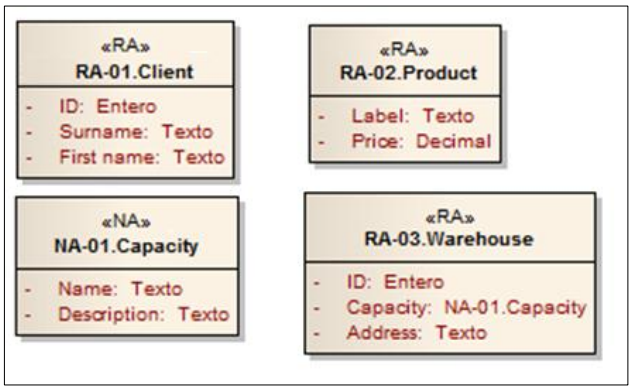

Figure 3. Information Storage Requirements from 1st Project

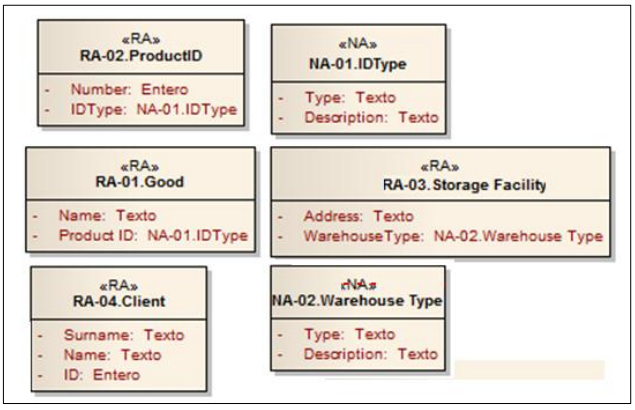

Figure 4. Information Storage Requirements from 2nd Project

Table 1 shows the description of two requirements. The corpus in this short example is only two descriptions.

Table 1. Similarity between Requirements (I)

NA-02.Warehouse Type	NA-01.Capacity
The capacity of the storage facilities is defined in this object. Warehouses are from three different kinds.	This represents the capacities defined for warehouses. Warehouses contain different quantities of products to be sold.

Table 2 shows the *tf* and *idf* of the underlined terms for both objects giving as example the term "warehouse".

Table 2. Similarity between Requirements (II)

NA-02.Warehouse Type	NA-01.Capacity
$tf_{warehouse} = 1/11$ $idf_{warehouse} = log\frac{2}{1+2} \approx -0.176$ $tf * idf \approx -0.016$	$tf_{warehouse} = 2/11$ $idf_{warehouse} = log\frac{2}{1+2} \approx -0.176$ $tf * idf \approx -0.032$

In the same way, we obtain the results for all terms that are underlined in Table 1. Table 3 presents these results.

On the other hand, we need to calculate the vector associated with the concepts we have described in Table 1. This vector has 17 components: one for each keyword (you must remember the criteria explained in this section such as plurals, etc.). Then, the vector's dimensions are:

{capacity, represent, storage, define, facility, is, warehouse, contain, object, different, quantity, are, product, three, be, sell, kind}

Table 3. Similarity between Requirements (III)

NA-02.Warehouse Type *tf * idf*	
Capacity	*-0.016*
Storage	*0*
Facilities	*0*
Is	*0*
Defined	*-0.016*
Object	*0*
Warehouse	*-0.016*
Are	*0*
Three	*0*
Different	*-0.016*
kinds	*0*

NA-01.Capacity *tf * idf*	
Represents	*0*
Capacities	*-0.016*
Defined	*-0.016*
Warehouses	*-0.032*
Contain	*0*
Different	*-0.016*
Quantities	*0*
Products	*0*
Be	*0*
Sold	*0*

Table 4 shows the values of the vector components for each one of the requirements shown in Table 1.

Table 4. Similarity between Objects (IV)

	NA-02.Warehouse Type	NA-01.Capacity
capacity	*-0.016*	*-0.016*
represent	*0*	*0*
storage	*0*	*0*
define	*-0.016*	*-0.016*
facility	*0*	*0*
is	*0*	*0*
warehouse	*-0.016*	*-0.032*
contain	*0*	*0*
object	*0*	*0*
different	*-0.016*	*-0.016*
quantity	*0*	*0*
are	*0*	*0*
product	*0*	*0*
three	*0*	*0*
be	*0*	*0*
sell	*0*	*0*
kind	*0*	*0*

Finally, we calculate the angle between the vectors described in Table 4. For this, we apply the cosine:

$$cos(\alpha) = \frac{V1.V2}{||V1||.||V2||} = \frac{0.00128}{0.032 \times 0.042} \approx 0.952$$

This means that the angle between the two vectors is very small (≈0.3 rad) and the vectors are very similar.

4.2.3 Algorithm of concepts' identification

Before starting the identification algorithm, user must have previously indicated what requirements he/she wants to merge. User must also indicate the two NDT-Profile projects. Then, our tool (NDT-Merge) can run the algorithm. Our procedure for identifying concepts is as follows.

Our tool (NDT-Merge) gets the set of requirements from the first project. It is usually called a query. The whole set of requirements from the second project constitutes the compared objects. Then cosine similarity is applied between the requirement from the first project and each of the requirements from the second project. All the results are saved. Then the second requirement of the first project is compared to the same set and so on until all the requirements from the first project have been used (1). The results are taken all together and sorted. The sorting algorithm takes the pair which got the highest similarity and save it has a correspondence between the two objects: they are describing the same object. Then the second more relevant pair is taken and so on until we get all the requirements from both projects. In fact we

want to have a correspondence for each requirement (2). It can be possible that several requirements got linked to a single one if the number of requirements is not the same in both project (3), this case is visible in the following example. We consider the example shown in Figures 3 and 4.

In the first project, the first requirement to be used as query is "RA-01.Client" (see Figure 3). Next, our tool calculates the cosine similarity with each requirement of the same type in the second project (see Figure 4): RA-02.ProductID, RA-01.Good, RA-03.Storage Facility, RA-04.Client. Thus, each requirement gets at least one relationship with the other requirements. Posteriorly, our tool applies exactly the same algorithm for "RA-02.Product" and "RA-03.Warehouse" in the first project.

The temporary results of applying the cosine similarity for the Storage Requirements are shown in the Table 5.

Table 5. Results of Applying the Cosine Similarity

First project	Second project	Score ($0 \leq cosine \leq 1$)
Client	Client	*0.596986*
Product	Good	*0.335637*
Product	Client	*0.223984*
Warehouse	Storage Facility	*0.203202*
Warehouse	Client	*0.184944*
Client	Good	*0.134006*
Warehouse	Good	*0.121600*
Client	ProductID	*0.119864*
Client	Storage Facility	*0.109182*
Warehouse	ProductID	*0.107809*
Product	ProductID	*0.090265*
Product	Storage Facility	*0.074950*

The algorithm allows getting only the highest scores of each requirement. The results for this example are shown in the Table 6.

Table 6. Results of Applying the Cosine Similarity (I)

		Score ($0 \leq cosine \leq 1$)
RA-XX	Client & Client :	*0.596986*
	Product & Good :	*0.335637*
	Warehouse & Storage Facility :	*0.203202*
	Product & ProductID[1] :	*0.090265*
NA-XX	Capacity & Warehouse Type :	*0.222456*
	Capacity & IDType[2] :	*0.057308*

All the objects are linked and the doubled objects will be considered as structural conflicts.

[1] "ProductID" requirement doesn't have any equivalent in the first project. The algorithm links it to the closest object of the same type.

[2] "Capacity" requirement is taken twice because "IDType" requirement have not any equivalent. "Capacity" is the closest requirement.

It is sometimes difficult to get relevant similarity results because the texts may be too short. It may happen that the list of similarities gets low scores in general, when for example the words chosen are synonyms ("products" and "goods" for instance). In its actual version, our algorithm doesn't use any dictionary of synonyms. For this reason, the scores are only based on roots of words.

To improve the results, a second step is realized. With this new step, the algorithm takes into account the name of the requirements and the text of the descriptions. We temporary add the usable name (without the type and attributes) to the description. In general we observed that the description contain terms from the name. Adding it, we improve the *tf* component of a term which is in the title and in the description. Sometimes the term in the name/title is not present in the description. This will improve the *idf* component of this term. As we do this for both projects the similarity cosine should be improved between objects to match.

Table 7 shows the results taking into account this second step. You can check the results have improved from the results shown in Table 6.

Table 7. Results of Applying the Cosine Similarity (II)

		Score ($0 \leq cosine \leq 1$)
RA-XX	Client & Client :	*0.641377*
	Product & Good :	*0.345933*
	Warehouse & Storage Facility :	*0.226203*
	Product & ProductID :	*0.117276*
NA-XX	Capacity & Warehouse Type :	*0.222456*
	Capacity & IDType :	*0.057308*

Then, obtained score are changed and several cases for the descriptions may occur:

- The score of the considered pair is improved following a general improvement of the majority of pairs: the two names contained same words and confirm that the two objects represent the same requirement. It is often the case that analyst chose similar words as name from one group to another to identify an object. This case is shown in Table 7.
- The score is quite the same (in comparison of the rank of values) and shows that the names were different. The result is not improved so the name did not enhance the similarity for the considered pair. The analysts chose different names; this is a structural conflict on an identified pair we may resolve later, choosing one of them.
- The score is seriously raised, and two other pairs containing the two requirements with two other requirements associated respectively had higher scores in the first list. We consider that the names were the most relevant element in the descriptions which were not considered describing the same requirement. Therefore we conclude that the names, which contain same element or are identical, create a semantic conflict.

4.2.4 Conflicts characterization

The difference of the components of the pairs is made in order to get the *structural conflicts*. The components depend on the type of NDT requirement dealt. As said before, the *structural conflicts* can be seen as a difference in any property of two objects when they are identified as a unique concept. These properties are state of the object (approved, etc.), tags, author, description, attributes, and name, among others. We call these characteristics as common characteristics.

Most of the properties are also very specific to the type of object. For instance, Interaction Requirements have sub objects like combo boxes, textboxes, checkboxes, and buttons, among others. Moreover, these sub objects have their own characteristics.

In its actual version, our algorithm only deals with simple objects or common characteristics of the objects.

For the attributes for example, we apply the same algorithm than before. The sets of "requirements" are the requirements of each pair obtained before. Then, we compare all the attributes to get pairs.

In our example, in the pair of storage requirements "RA-01.Client" (see Figure 3) we may get the conflict on "First name" and "Name" which are identified as same concepts by their descriptions. The conflict is then a *structural conflict* about the name we have to choose. Another example would be between "Product" and "Good". The attributes "label" and "name" may generate a conflict as described above, the other attributes are different so we generate a conflict in the requirement level and in the resolution, and we add all of them in the new requirement.

NDT requirement have also special fields for links. These relations allow to link requirements from one diagram to others, which can be of different type, in the same one or in a different one. This is then analysed to get link conflicts when the lists are different.

4.3 Conflict resolution

The resolver module gets the conflicts and creates a new NDT-Profile project. The new project comes with some reports about the detection. We describe how the software may deal the resolution which is still in development.

The Resolver Module will work with the *structural conflicts* to erase them. We define two cases. First, we add artefacts which are absent in one model and present in the other one to the new project – they were added as *structural conflicts*. We take an optimistic position understanding that the best solution is to include the construction as an improvement when it is not present. This idea comes from the fact that new requirement artefact may improve other requirements functionality. Then, for artefacts type, or configuration incompatibility, we need to analyse deeper, putting some priority rules to choose one of the objects or to even replace by another type of object. Some rules concerning the interaction diagram in NDT are given in [9] and they will be extended in our project.

- Read-write over Read-only widgets. It may happen that the structural comparison exposes a contrast between read-only widget (or disabled TextField) and a TextField. In this case, we choose the most flexible one: use a TextField to enable showing and editing data.
- Fixed data values range over wide values range. Two widgets may deal with the same data but differ in the manipulated range; masked text inputs and restricted set of options are examples. In this case, restrictive widget such as Combobox, RadioButton or masked TextFields are prioritized over less restrictive widgets.
- Container vs. atomic widgets: When having one VisualizationPrototype specifying a Container that defines an aggregation of data against a non container widget such as a TextField, Containers must be preserved because they establish a detailed information structure specification.

Note that each NDT model must be specifically analyzed, however, a common process for all the NDT objects consists in letting the analyst choose between the two configurations of NDT common attributes (Names, Notes...) or propose him to make a merge version.

The *structural conflicts* may be dealt among their complexity. If it is a name problem, e.g. they are different, we can arbitrary choose one of them if the name is a unique key in the project.

The *link conflict* resolution is made in a first version by putting into the link list of an object only the requirements which are present in both requirements.

The *semantic conflict* can be dealt by just identifying it in a report and not changing the name because the action of choosing names can be complex and not easily automated.

5. Conclusions & Future Works

In a software project, one of the most relevant phases in the lifecycle is the requirements phase, which conditions the development through all the aspects of the project, mainly economic. Either the diversity of data the system has to manage or the diversity of users show the complexity analysts must face up. In big projects, managers usually share tasks among different teams working on separated aspects often occurring in the same phases. Then it is particularly important to conciliate results. Conflicts in the requirements phase must be solved to get a working set of models for next phases of development. Nevertheless, this task frequently depends on the analyst's experience or it is performed manually, without a specific and normalized support to develop it.

In this paper we propose to extend NDT-Suite with a new tool (NDT-Merge) that aims to help analysts in this task saving time. The process, using NDT methodology for the systematic detection of requirements inconsistencies, extends it to the conflicts resolution that already exists in some methodologies like WebSpec. The objective is to propose a tool capable of solving conflicts for any types of requirements of NDT and their models. In this paper we mainly focus on the model of interaction requirements which organises the functional requirements through the construction of the future interface prototype, because these mechanisms bring into play many specific aspects of NDT and include generic processes used for the whole merging step.

ACKNOWLEDGMENTS

This research has been supported by the Tempros project (TIN2010-20057-C03-02) and the NDTQ-Framework project of the Junta de Andalucía, Spain (TIC-5789).

REFERENCES

[1] Kotonya, G., Sommerville, I. 1996. Requirements engineering with viewpoints. *Software Engineering Journal*, vol.11, no.1, pp.5-18.

[2] De Lucia, A., Qusef, A. 2010. Requirements Engineering in Agile Software Development. *In Journal of Emerging Technologies in Web Intelligence*, Vol. 2, no. 3, pp 212-220.

[3] McConnell, S. 1996. Rapid Development: Taming Wild Software Schedules. *Microsoft Press*. ISBN 1-55615-900-5.

[4] Leffingwell, D. 1997. Calculating the Return on Investment From More Effective Requirements Managament. *American Programmer*, Vol. 10, no. 4, pp 13-16.

[5] Escalona, M.J., Aragón, G. 2008. NDT: A Model-Driven Approach for Web requirements, *IEEE Transactions on Software Engineering.* Vol. 34, no. 3. pp 370-390.

[6] NDT-Suite. 2012. Available at www.iwt2.org. Accessed in August 2012.

[7] Escalona, M.J. 2007. Equipo de Coordinación. Mosaico. El sistema de Información para la Gestión del Patrimonio Histórico Andaluz. *Proceedings of XI International Congress on Project Engineering*.

[8] IEEE. 1998. IEEE Recommended Practice for Software Requirements Specifications. *IEEE Std* pp 830-1998.

[9] Urbieta M., Escalona M.J., Robles E., Rossi G. 2011. Detecting Conflicts and Inconsistencies in Web Application Requirements. *ICWE Workshops 2011*, pp 278-288.

[10] Robles, E., Garrigós, I., Grigera, J., Winckler, M. 2010. Capture and Evolution of Web Requirements Using WebSpec. *ICWE Workshops 2010*, pp 173-188.

[11] Silva, J.R., Do Santos, E.A. 2004. *Applying Petri Nets to requirements validation. ABCM Symposium. Series in Mechatronics.* Vol. 1. pp. 508-517.

[12] Ceri, S., Fraternali, P. Bongio, A., Brambilla, M., Comai, S., Matera, M. 2002. Designing Data-Intensive Web Applications. M*organ Kaufmann Publishers Inc.*, San Francisco, CA, USA.

[13] Brito, I. S., Vieira, F., Moreira, A., Ribeiro, R. A. 2007. Handling conflicts in aspectual requirements compositions. *In Transactions on aspect-oriented software development III, LNCS*, Vol. 4620. Springer-Verlag, Berlin, Heidelberg, pp 144-166.

[14] Altmanninger, K. 2007. Models in Conflict - Towards a Semantically Enhanced Version Control System for Models. *MoDELS Workshops 2007*, pp 293-304.

[15] Sommerville, I.: Software Engineering. Addisson Wesley (2002). Van Der Straeten, R., Mens, T., Simmonds, J., Jonckers, V. 2003. Using Description Logic to Maintain Consistency between UML Models. *UML 2003*, pp 326-340.

[16] Salton G., Buckley C. 1988. Term-Weighting approaches in automatic text retrieval. *Department of Computer Science, Cornell University*, Ithaca, NY 14853, USA.

[17] Rifaieh R. et Al. 2006. A Matching Algorithm for Electronic Data Interchange. *Proceeding TES'05 Proceedings of the 6th international conference on Technologies for E-Services*, Springer-Verlag Berlin, Heidelberg 2006, pp 34-47. ISBN: 3-540-31067-3 978-3-540-31067-9.

[18] Salton, G. and McGill M. J. 1986. Introduction to modern information retrieval.

[19] Enterprise Architect. 2012. Available at www.sparxsystems.com. Accessed in August 2012.

[20] Escalona M.J., Gutiérrez J.J., Pérez-Pérez M., Molina A., Martínez-Force E., Domínguez-Mayo F.J. 2011. Measuring the Quality of Model-Driven Projects with NDT-Quality. *Information System Development. Springer Science Business Media, LLC 2009*; USA. Vol. 1, Chapter 26, pp. 307-317, ISBN/ISSN: 978-1-4419-7355-9.

[21] García-García J.A., Cutilla C.R., Escalona M.J., Alba M., Torres J. 2011. NDT-Driver, a java tool to support QVT transformations for NDT. *The Twentieth International Conference on Information Systems Development (ISD 2011)*. ISBN: 978-1-4614-4950-8.

[22] García García J. A., Escalona M.J., Cutilla C.R., Alba M. 2011. NDT-Glossary. *Proceedings of the 13th International Conference on Enterprise Information System (Iceis 2011).* International Conference on Enterprise Information Systems (Iceis) (13). no. 13. Beijing, China. Insticc Press. 2011. Pag. 170-175.

Annex

We give the descriptions of the requirements and their attributes

1st project	2nd project
RA-01.Client "This represents a client. The client is a particular person buying the company's products." • ID "This is the national identification number of the client" • Surname "This field represents the client's surname." • First name "This field represents the client's first name."	**RA-01.Good** "The goods are the products sold by the company. They are stored in warehouses before clients buy them." • Name "This is the name given to that category of good." • Product ID "This is the product identification number. There are several types of products, some are made by the company and others are bought."
RA-02.Product "This represents a product the company may sell. The products are material goods stored before being sold by the company." • Label "This represents the label given to that kind of product by the company." • Price "This is the price in euros of the product."	**RA-02.ProductID** "This is the product ID. There are types of ID according to the product provenance." • Number "This is the ID number." • IDType "This represents the ID type."
RA-03.Warehouse "This represents a place where the products are stored before being distributed to smaller facilities." • ID "This is the identification number used by the company for the warehouses." • Capacity "The capacity of the warehouse. This can take three different values defined by the Capacity requirement." • Address "This is the warehouse's place."	**RA-03.Storage Facility** "This represents the facility's data where are stored the goods. There are different kinds of them." • Address "The address of the warehouse." • Warehouse Type "This represents the type of warehouse. It depends on the size."
NA-01.Capacity "This represents the capacities defined for warehouses. Warehouses contain different quantities of products to be sold." • Name "This represents the capacity of the warehouse using the internal cathegories system. It may be : A, B, C. A is a small warehouse where a thousand products can be stored. B is usually between twice and five times bigger. This category of warehouse is usually present in big cities. The C capacity is used to represent the main warehouses which are a hundred times bigger than the A one. The company only owns a couple of C warehouses." • Description "This field gives a description to each type of Capacity."	**RA-04.Client** "This represents a client of the company. The client can only be a particular person who buys goods." • Surname "The client's surname." • Name "This is the client's name." • ID "This is the national client's ID."
	NA-01.IDType "This is the type of ID a product or good can have. It depends on several criterious." • Type "This represents the ID types. There are two different types, X and Y, X define the products which are distributed under new ownership and Y the own made ones." • Description "This allows to explain the type of product ID."
	NA-02.Warehouse Type "The capacity of the Storage Facilities is defined in this object. Warehouses are from three different kinds." • Type "The type of the warehouse depends on the size and capacity. The 1000G (1000 goods capacity), the 2-5000G (from 2000 to 5000 products), the 100MG represents a main warehouse which is presents only in main cities (100000 Goods)." • Description "This describes the capacities of the storage facility."