Measuring and Improving the Completeness of Natural Language Requirements

Alessio Ferrari[1], Felice dell'Orletta[2], Giorgio Oronzo Spagnolo[1], and Stefania Gnesi[1]

[1] ISTI-CNR, Pisa, Italy
{alessio.ferrari,giorgio.oronzo.spagnolo,stefania.gnesi}@isti.cnr.it
[2] ILC-CNR, Pisa, Italy
felice.dellorletta@ilc.cnr.it

Abstract. [**Context and motivation**] System requirements specifications are normally written in natural language. These documents are required to be *complete* with respect to the input documents of the requirements definition phase, such as preliminary specifications, transcripts of meetings with the customers, *etc.* In other terms, they shall include all the *relevant concepts* and all the *relevant interactions* among concepts expressed in the input documents. [**Question/Problem**] Means are required to *measure* and *improve* the completeness of the requirements with respect to the input documents. [**Principal idea/results**] To *measure* this completeness, we propose two metrics that take into account the relevant terms of the input documents, and the relevant relationships among terms. Furthermore, to *improve* the completeness, we present a natural language processing tool named Completeness Assistant for Requirements (CAR), which supports the definition of the requirements: the tool helps the requirements engineer in discovering relevant concepts and interactions. [**Contribution**] We have performed a pilot test with CAR, which shows that the tool can help improving the completeness of the requirements with respect to the input documents. The study has also shown that CAR is actually useful in the identification of specific/alternative system behaviours that might be overseen without the tool.

Keywords: Requirements analysis, requirements completeness, requirements quality, natural language processing, terminology extraction, relation extraction.

1 Introduction

The starting point of a requirements definition process is very rarely a blank paper. More often, several input documents are placed on the desk of the requirements engineer, from legacy system documentation to reference standards, from transcripts of meetings with the customers to preliminary specifications. The content of these documents has to be taken into account when writing the requirements [1, 2], since it settles the background on which the future system can start to take its form. Such input documents are normally written in natural language (NL), and suitable natural language processing (NLP) tools can help identifying all the information that is relevant for the requirements. NLP approaches have been proposed in the past to identify significant abstractions that can aid the requirements process (e.g., [3, 4]). However, none of the existing approaches considers the *completeness* of the requirements with respect to the existing documentation. A requirements document that does not include the relevant information of the input documents - i.e., it is *incomplete* - could bring to several problems: if the missing information resides in the transcripts of meetings with the customers, the product might not address the customer's expectations; if some information is overseen from the reference standards, the resulting product might not comply to the norms; when concepts from legacy documentation and preliminary specifications are not taken into account, re-work on the product or on the process artifacts is hard to avoid.

In this paper, we propose a NLP-based approach to *measure* and *improve* the completeness of a requirements specification with respect to the input documents of the requirements definition process. A requirements document is *complete* with respect to the input documents if all the *relevant concepts* and *interactions* among concepts expressed in the input documents are also treated in the requirements. We refer to this type of completeness as *backward functional completeness*. In order to measure such completeness, we provide two metrics that take into account the relevant terms and relevant relations among terms of the input documents. Furthermore, we provide a NLP approach to automatically extract such terms and relations. A prototype tool named Completeness Assistant for Requirements (CAR) has been developed, which suggests relevant information during the requirements definition phase, and automatically computes the degree of completeness of the requirements specification produced.

We evaluate the effectiveness of the approach with a pilot test, which is also used as a reference example in the remainder of the paper. The pilot test concerns the definition of the requirements for an Automatic Train Supervision (ATS) component of a Communications-based Train Control system (CBTC). CBTC systems are signalling and control platforms tailored for metro, standardized by the IEEE Std 1474.1-2004 [5]. These systems provide automatic train protection, train monitoring, and automated train driving. The ATS component of a CBTC is a centralized system that monitors and regulates the movement of the trains. The system automatically routes trains, and sends them speed profiles that shall be followed while moving through the railway network. It is normally equipped with a user interface where the ATS operator can view the position of all the trains, their schedule, and other information.

From the pilot test, we find that the CAR tool actually helps in improving the completeness of the requirements specification with respect to the input documents – in our case, the ATS reference standard. The tool suggests relations about concepts that do not appear evident while reading the input document, and facilitates the identification of specific/alternative behaviours of the ATS system.

The paper is structured as follows. In Sect. 2, we give some background on requirements specifications completeness. In Sect. 3, the research questions addressed by the current paper are presented. In Sect. 4, we introduce two metrics to evaluate the backward functional completeness of a requirements specification. In Sect. 5, the CAR tool is described. Sect. 6 presents the evaluation of the approach through a pilot test. Sect. 7 provides conclusions and future works.

2 Defining and Measuring Completeness

In general, a requirements specification is complete if all the necessary requirements are included [6]. Several works have been presented in the literature to *define* and to *measure* the completeness of a requirements specification. In this paragraph, we review some definitions, which give a framework to understand the concept of *backward functional completeness* provided by the current paper.

Completeness. A largely agreed definition of completeness of a requirements specification can be found in Boehm [7]. The definition states that a complete specification shall exhibit five properties: 1) No To-be-determined (TBD) items 2) No nonexistent references 3) No missing specification items (e.g., missing interface specifications) 4) No missing functions 5) No missing products (i.e., part of the actual software that are not mentioned in the specification).

Internal/External Completeness. The definition is further conceptualized by Zowghi and Gervasi [8]. The first two properties defined by Bohem [7] are associated to *internal completeness*, and the second three properties to *external completeness*. Internal completeness can be measured by considering solely the information included in the specification. Instead, measuring external completeness requires additional information provided by domain experts, for example in the form of a domain model.

Feasible Semantic Completeness. A more formal definition of external completeness - referred as *semantic completeness* - is given in Lindland et al. [9]. They look at the requirements specification as a *conceptual model* M, and they state that M has achieved semantic completeness if it contains all the statements about the domain D that are correct and relevant (i.e., $D \setminus M = \emptyset$). They observe that total semantic completeness cannot be achieved in practice, and they define the concept of *feasible semantic completeness* as $D \setminus M = S \neq \emptyset$. The set S is composed of correct and relevant statements, but there is no statement in S such as the benefit of including it in the specification exceeds the drawback of including it.

Functional Completeness. A further refinement of the concept, which goes toward the definition of a completeness measure, is provided by España et al. [10]. In line with the observations of Zwoghi and Gervasi [8], the authors argue that, in order to compute the feasible semantic completeness, a reference model M_r shall be defined to conceptualize the domain D. By focusing on functional requirements, they consider the subset $FM_r \subset M_r$, which is a model of the functional requirements. Such a model is composed of functional encapsulations F_r, roughly "functions", and linked communications LC_r, roughly "messages". More formally, $F_r = F_r \cup LC_r$.

A functional requirements specification FM shall be compared against this reference model FM_r to evaluate its completeness. Therefore, the specification FM shall be regarded as a composition of functional encapsulations F and linked communications LC (i.e., $FM = F \cup LC$). The introduced concepts are used to define two aspects of *functional completeness*:

– *functional encapsulation completeness:* all functional requirements specified in the reference model have been specified in the model (i.e., $F_r \setminus F = \emptyset$).
– *linked communication completeness:* all linked communications specified in the reference model have been specified in the model (i.e., $LC_r \setminus LC = \emptyset$).

In order to provide metrics associated to these aspects, the authors define the degree of functional encapsulation completeness as $degFEC = |F|/|F_r|$, and the degree of linked communication completeness as $degLCC = |LC|/|LC_r|$. In practice, computing these metrics requires the definition of a reference model for the functional requirements in terms of functions and linked communications.

3 Motivation

Besides the one applied by España et al. [10], several other measures for functional requirements completeness have been proposed in the literature (e.g., [11–15]). Nevertheless, the majority of such metrics deal with functional completeness defined with respect to the future implementation of the system[1]. Indeed, domain models [10], ontologies [15], identification of components [14], identification of system states [12], or expert analysis [11] are required to compute this kind of completeness. In other terms, domain experts are called to foresee a possible implementation of the system, possibly through a reference functional model FM_r. According to this vision, we refer to this kind of completeness as *forward functional completeness.* Instead, in our work we wish to focus on the completeness of the requirements with respect to the available input documents of the requirements definition process. The input documents might be transcripts of meeting with customers, preliminary specifications, reference implementation standards, or any other information specifically regarding the system under development. We refer to the completeness of a functional requirements specification with respect to the input documents as *backward functional completeness.*

Backward functional completeness is achieved by a functional requirements specification when (1) all the *relevant concepts* expressed in the input documents are treated in the requirements specification; (2) all the *relevant interactions* among concepts expressed in the input documents are treated in the requirements specification.

[1] One exception is [13], where completeness is evaluated against higher-level requirements.

Consider for example the input document of our pilot test [5]. The document contains the sentence *"An ATS system shall have the capability to automatically track, maintain records of, and display on the ATS user interface the locations, [...], the train schedule and [...]"*. Besides the other content, such a sentence tells that the ATS user interface is supposed to display the schedule of the trains. Therefore, the requirement specification is expected to include the concepts of "ATS user interface" and "train schedule". Furthermore, requirements shall be provided that define the *interaction* among the two concepts (i.e., the fact that the ATS user interface shall display the train schedule).

Achieving backward functional completeness ensures that no relevant information contained in the input documents is left out from the specification. Measuring this type of completeness can give higher confidence on the quality of the specification. Therefore, a metric is required to measure this kind of completeness. Furthermore, we are also interested in establishing whether a positive correlation holds between such completeness and the completeness of the specification with respect to the system to be (i.e., the *forward functional completeness*).

Bearing these observations in mind, we define three research questions, which are addressed by the current paper: **RQ1.** How to *measure* the backward functional completeness of a requirements specification document? **RQ2.** How to *improve* the backward functional completeness of a requirements specification document? **RQ3.** Does the backward functional completeness help in improving the *forward functional completeness* of the specification?

The first question is answered by computing two completeness metrics that consider the number of relevant terms that are used in the input documents, and the number of relevant relations among terms (Sect. 4). Roughly, a document is more complete than another if more relevant terms and more relevant relations are included in the document. The second question is answered through a prototype tool that suggests relevant terms to be included in the requirements, and that considers the relations among terms (Sect. 5). The third question is answered through a pilot test, where we have evaluated the forward functional completeness of the requirements produced with the proposed tool, and without the proposed tool (Sect. 6).

4 Metrics for Backward Functional Completeness

Measuring the backward functional completeness of a requirements specification requires the definition of specific metrics (**Research Question 1**). Here, we define two metrics. The first one, named *degree of concept completeness*, measures how many relevant concepts that are expressed in the input documents are treated also in the specification. The second one, named *degree of interaction completeness*, measures how many relevant interactions that are expressed in the input documents are treated also in the specification.

More formally, we define the two metrics as follows. Let T be the set of relevant concepts expressed in the input documents, and let $Q \subseteq T$ be the set of such concepts expressed in the requirements specification. We define the *degree* *of concept completeness* of a requirements document $\mathcal{D}$ with respect to a set of input documents $\mathcal{I}$ as $degCC(\mathcal{D},\mathcal{I}) = |Q|/|T|$.

Now, let U be the set of relevant interactions among concepts expressed in the input documents, and let $R \subseteq U$ be the set of relevant interactions among concepts expressed in the requirements specification. We define the *degree of interaction completeness* of a requirements document $\mathcal{D}$ with respect to a set of input documents $\mathcal{I}$ as $degIC(\mathcal{D},\mathcal{I}) = |R|/|U|$.

Given a requirements document and the corresponding input documents, we would like to compute the two metrics in an automated manner.

We argue that the relevant concepts expressed in the input documents can be approximated with the relevant *terms* included in such documents. Furthermore, relevant interactions among concepts can be approximated with the relevant *relations* among terms. Therefore, we define a NLP approach to automatically identify relevant terms and relations among terms in the input documents.

4.1 Identification of Relevant Terms

The proposed method for the identification of relevant terms is based on a novel natural language processing approach, named *contrastive analysis* [16], for the extraction of *domain-specific terms* from natural language documents. In this context, a *term* is a conceptually independent linguistic unit, which can be composed by a single word or by multiple words. For example, consider the document that we have used in our pilot test [5]. In such document, "Automatic Train Supervision" is a term, while "Supervision" is not a term, since in the textual documents considered in our study it often appears coupled with the same words (i.e., "train", "route"), and therefore it cannot be considered as conceptually independent.

The *contrastive analysis* technology aims at detecting those terms in a document that are *specific* for the domain of the document under consideration [16, 17]. Roughly, contrastive analysis considers the terms extracted from domain-generic documents (e.g., newspapers), and the terms extracted from the domain-specific document to be analysed. If a term in the domain-specific document highly occurs also in the domain-generic documents, such a term is considered as domain-generic. On the other hand, if the term is not frequent in the domain-generic documents, the term is considered as domain-specific.

In our work, the documents from which we want to extract domain-specific terms are the input documents of the requirements definition phase. The proposed method requires two steps. First, conceptually independent expressions (i.e., *terms*) are identified (*Identification of Terms*). Then, *Contrastive Analysis* is applied to select the terms that are domain-specific.

Identification of Terms. Given a set $\mathcal{I} = \{I_1, \ldots, I_n\}$ of input documents, we aggregate the documents in a single input document I. From this document, which collects the content of all the input documents, we identify a ranked list of *terms*. To this end, we perform the following steps.

1. POS Tagging: first, Part of Speech (POS) Tagging is performed with an english version of the tool described in [17]. With POS Tagging, each word is associated with its grammatical category (*noun*, *verb*, *adjective*, etc.).

2. Linguistic Filters: after POS tagging, we select all those words or groups of words (referred in the following as *multi-words*) that follow a set of specific POS patterns (i.e., sequences of POS), that we consider relevant in our context. For example, we will not be interested in those multi-words that end with a preposition, while we are interested in multi-words with a format like <*adjective*, *noun*, *noun*> (such as "Automatic Train Supervision").

3. C-NC Value: terms are finally identified and ranked by computing a "termhood" metric, called C-NC value [16]. This metric establishes how much a word or a multi-word is likely to be conceptually independent from the context in which it appears. The computation of the metric is rather complex, and the explanation of such computation is beyond the scope of this paper. The interested reader can refer to [16] for further details. Here we give an idea of the spirit of the metric. Roughly, a word/multi-word is *conceptually dependent* if it often occurs with the same words (i.e., it is *nested*). Instead a word/multi-word is *conceptually independent* if it occurs in different context (i.e., it is normally accompanied with different words). Hence, a higher C-NC rank is assigned to those words/multi-word that are conceptually independent, while lower values are assigned to words/multi-words that require additional words to be meaningful in the context in which they are uttered.

After this analysis, we have a ranked list of k words/multi-words that can be considered *terms*, together with their ranking according to the C-NC metric, and their frequency (i.e., number of occurrences) in I. The more a word/multi-word is likely to be a *term*, the higher the ranking.

Contrastive Analysis. The previous step leads to a ranked list of k terms where all the terms might be *domain-generic* or *domain-specific*. With the contrastive analysis step, terms are re-ranked according to their domain-specificity. To this end, the proposed approach takes as input: 1) the ranked list of terms extracted from the document I; 2) a second list of terms extracted from a set of documents that we will name the *contrastive corpora*. The contrastive corpora is a set of documents containing domain-generic terminology. In particular, we have considered the Penn Treebank corpus, which collects articles from the Wall Street Journal. The reasonable assumption here is that a term that frequently occurs in the Wall Street Journal is not likely to be a domain-specific term of the domain of a technical requirements specification. The new rank $TRank(t)$ for a term t extracted from the document I is computed according to the function [16]:

$$TRank(t) = \log(f(t)) \cdot (\frac{f(t)}{\frac{F_c(t)}{N_c}})$$

where $f(t)$ is the frequency of the term t extracted from I, $F_c(t)$ is the sum of the frequencies of t in the contrastive corpora, and N_c is the sum of the frequencies of all the terms extracted from I in the contrastive corpora. Roughly, if a term is less frequent in the contrastive corpora, it is considered as a *domain-specific* *term*, and it is ranked higher. Consider again our pilot test. After the contrastive analysis, a term such as "train" – which is highly frequent in the document (57 occurrences), but is also frequent in the contrastive corpora – is ranked lower than "ATS user interface". Indeed, this term has 8 occurrences in the document, but is uncommon in the contrastive corpora.

After this analysis, we have a list of terms, together with their ranking according the function $TRank$, and their frequency in I. The more a term is likely to be domain-specific, the higher the ranking. From the list, we select the terms that received the higher ranking. The choice shall be made according to a *domain relevance threshold* τ. If $TRank(t) \geq \tau$ the term will be selected as *relevant*. The value of τ is defined over normalized values, where the rank of each term is divided by the maximum value of $TRank$. The selection of τ shall be performed by a domain expert after reviewing the lists of terms extracted. Normally, a value of $\tau = 0.99$ allows selecting most of the relevant terms.

Assuming that the set of selected terms $\bar{T}$ provides an approximation of the relevant concepts of the input documents T, we can approximate the degree of concept completeness as $degCC(\mathcal{D}, \mathcal{I}) \approx |\bar{Q}|/|\bar{T}|$, where $\bar{T} = \{t \subset I : TRank(t) \geq \tau\}$, and $\bar{Q} = \mathcal{D} \cap \bar{T}$. For example, in our case study, we have $|\bar{T}| = 67$ relevant terms extracted from the input documents (see Table 1 for examples). In the first experiment, the requirements produced by subject A included $|\bar{Q}| = 46$ of such terms. Therefore $degCC(\mathcal{D}, \mathcal{I}) \approx 68.7\%$.

4.2 Identification of Relevant Relations

In order to identify relevant relations among terms, we first select all the terms t extracted in the previous step, regardless of their ranking. Then, we search for possible relations among such terms. We state that there is a relation $u = (t_j, t_h)$ between two terms t_j, t_h if such terms appear in the same sentence or in neighboring sentences. In our case, we select the previous and the following sentence. In order to give a rank to such relation, we use the Log-likelihood metric for binomial distributions as defined in [18]. The explanation of such metric is beyond the scope of this paper. Here, we give an idea of the spirit of the metric. Roughly, a relation holds between two terms if such terms frequently appear together. Moreover, the relation is stronger if the two terms do not often occur with other terms. In other words, there is a sort of *exclusive relation* among the two terms. For each couple of terms t_j, t_h occurring in neighboring sentences of the input document I, we associate a rank according to the Log-likelihood metric, which represents the degree of their relation $u = (t_j, t_h)$:

$$RRank(u) = \text{Log-likelihood}(t_j, t_h)$$

In our pilot test, the term "re-routing of trains" has a relation with "movement of trains" and with "ATS user interface". However, the relation is stronger (i.e., more exclusive) with the former ($RRank = 14.88$ *vs* $RRank = 8.85$), since the latter often occurs with other terms. Indeed, the ATS user interface is required to show several information, besides those concerning re-routing of the trains.

After this analysis, we have a list of relations, together with their ranking according the function *RRank*. From the list, we select the terms that received the higher ranking. The choice shall be made according to a *relation degree threshold* ρ. If $RRank(u) \geq \rho$, the relation will be selected as *relevant.* The selection of ρ shall be performed by a domain expert after reviewing the lists of relations extracted with the proposed method. Normally, a Log-likelihood above 10.83 is recommended to select only relevant relations. However, lower thresholds can be chosen, if more relations are required.

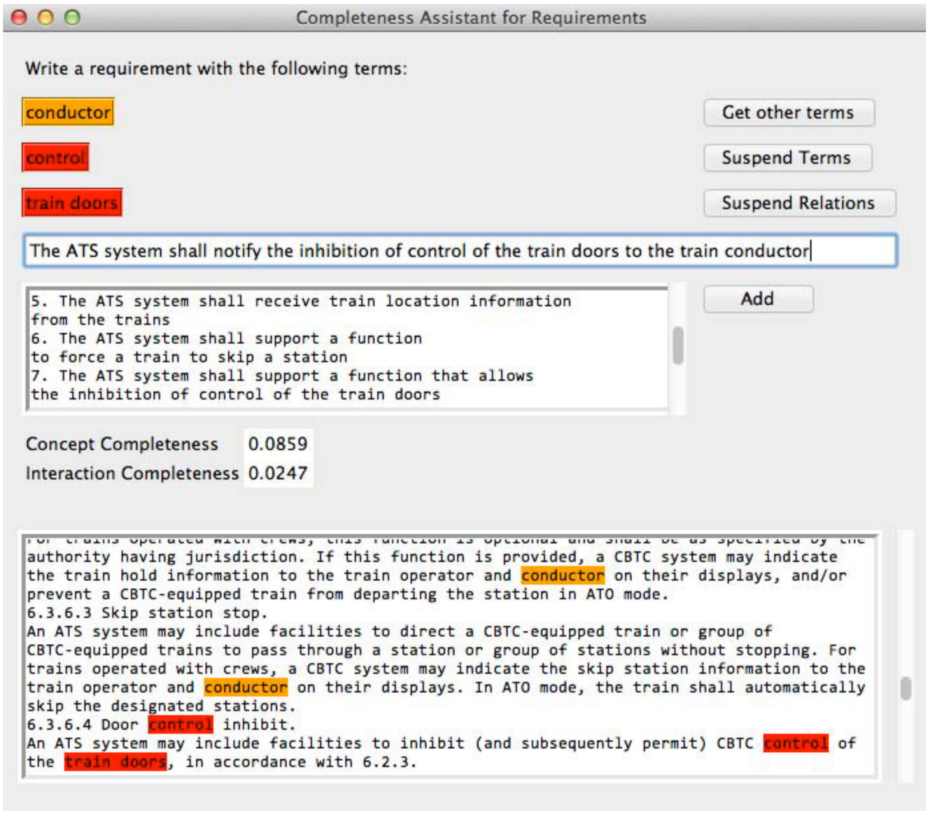
Assuming that the set of selected relations $\bar{U}$ provides an approximation of the relevant interactions U in the input documents, we can approximate the degree of interaction completeness as $degIC(\mathcal{D}, I) \approx |\bar{R}|/|\bar{U}|$, where $\bar{U} = \{u \in \bar{T} \times \bar{T} : RRank(u) \geq \rho\}$, and $\bar{Q} = (\mathcal{D} \times \mathcal{D}) \cap \bar{U}$. For example, in our case study, we have $|\bar{U}| = 316$ relations extracted from the input documents (see Table 2 for examples). In the first experiment, the requirements produced by subject A included $|\bar{R}| = 54$ of such relations. Therefore $degIC(\mathcal{D}, I) \approx 17.1\%$.

5 A Word-Game to Support Requirements Definition

We would like to provide means to improve the backward functional completeness of a requirements specification (**Research Question 2**). We argue that the backward functional completeness of a requirements specification is normally hampered by two problems: (1) *missing concepts:* the person who writes the requirements might forget to consider relevant concepts of the problem, either because she postpones their analysis, or because they are unclear and hard to specify, or because the input documents include too many concepts to consider them all; (2) *missing concept interaction:* when one writes a requirement, she might be concentrated on the specific function that she is defining, and oversee possible interactions among elements.

We have implemented a prototype tool named Completeness Assistant for Requirements (CAR), which addresses these problems by automatically suggesting possible relevant terms and possible relevant relations among terms to be used in the requirements. The relevant terms and relations are extracted from the input documents (e.g., transcripts of meeting with the customers, reference standards, preliminary requirements) according to the approach explained in Sect. 4. Therefore, the tool starts with a set $\bar{T}$ of relevant terms, and a set $\bar{U}$ of relevant relations. Furthermore, the degree of concept completeness and the degree of interaction completeness is computed at run-time while the requirements manager writes down the requirements.

Fig. 1 shows the interface of CAR. The figure is used as a reference example to explain the working principles of the tool. The example, adapted from our pilot test, concerns the definition of the requirements for an Automatic Train Supervision (ATS) system. An ATS system is a component of a metro control system that takes care of monitoring and routing trains. Furthermore, an ATS provides capabilities to remotely issue commands to the trains. The input document, in the example, is a reference international standard [5], which is used



Write a requirement with the following terms:

conductor

control

train doors

The ATS system shall notify the inhibition of control of the train doors to the train conductor

```
5. The ATS system shall receive train location information
from the trains
6. The ATS system shall support a function
to force a train to skip a station
7. The ATS system shall support a function that allows
the inhibition of control of the train doors
```

Concept Completeness	0.0859
Interaction Completeness	0.0247

```
authority having jurisdiction. If this function is provided, a CBTC system may indicate
the train hold information to the train operator and conductor on their displays, and/or
prevent a CBTC-equipped train from departing the station in ATO mode.
6.3.6.3 Skip station stop.
An ATS system may include facilities to direct a CBTC-equipped train or group of
CBTC-equipped trains to pass through a station or group of stations without stopping. For
trains operated with crews, a CBTC system may indicate the skip station information to the
train operator and conductor on their displays. In ATO mode, the train shall automatically
skip the designated stations.
6.3.6.4 Door control inhibit.
An ATS system may include facilities to inhibit (and subsequently permit) CBTC control of
the train doors, in accordance with 6.2.3.
```

Fig. 1. User interface of the tool

as a starting point to write the requirements for the ATS system. In general, the tool can work with any kind of natural language input document, such as interviews, transcripts of meetings with the customers, etc.

The tool is a sort of word-game. The main steps of the game are summarized below:

1. The tool suggests to write a requirements with three terms. The first term (conductor , in Fig. 1) is extracted from the set of relevant terms, while the other two terms (control , train doors) are extracted from the set of relevant relations. The three terms are also highlighted in the original document, which is loaded to the bottom frame of the interface. In the current version of the tool, the extraction is random. Nevertheless, smarter approaches can be devised that choose the terms by taking into account their relevance, their position, or the previously written requirements.
2. The user writes a requirement, possibly using the three terms suggested. An example requirement that employs the three terms is *"The ATS system shall notify the inhibition of* control *of the* train doors *to the train* conductor*"*.

Then, the user adds the requirement to the central panel by pressing the button **Add**. It is worth noting that a requirement like the one presented above could not be deduced by simply reading the text of the input document. It is actually an additional behaviour inspired by the suggested terms. Indeed, a relation between the "conductor" and the "train doors" was not specified in the original input document, as one can see from the fragment displayed in Fig. 1.

3. The system checks if the user used any relevant term or relevant relations, and consequently increases the degree of **Concept Completeness** and the degree of **Interaction Completeness**. These values are computed as $|\bar{Q}|/|\bar{T}|$ and $|\bar{R}|/|\bar{U}|$, respectively, as explained in Sect. 4.1 and 4.2. When relevant concepts are found within the requirement, these are added to the set $\bar{Q}$. When relevant relations are found, these are added to the set $\bar{R}$. The current values of the metrics are shown below the panel that lists the requirements.
4. The system automatically suggest other terms to be used in the following requirement.

If a relevant term or relation is suggested twice, and the user does not employ it in the requirement, such term/relation is marked as *not relevant.* Therefore, the completeness scores are adjusted consequently (i.e., $|\bar{T}|$ or $|\bar{U}|$ are decreased).

In some cases, the user might not be interested in writing a requirement that includes all the suggested terms. In other cases, the user might want to focus on the suggested terms/relations to write more than one requirement. With the normal behaviour of the tool, new terms/relations would be automatically suggested in these cases after pressing the button **Add**. As explained, if such terms/relations are not used, they are marked as *not relevant*, and will not be presented anymore among the suggestions. Therefore, we added the **Suspend Terms** and **Suspend Relations** buttons, to suspend the automated suggestion of terms and relations, and prevent the tool from marking them as not relevant.

If new relations among terms are reported in a requirement, these new relations shall be added to the relevant relations $\bar{U}$. In our case, the relations between "conductor" and the other two terms are added to $\bar{U}$. Similarly, if some terms are used that were not identified as relevant in the initial analysis, such terms shall be stored among the relevant terms $\bar{T}$. These situations do not influence the computation of the backward completeness (also $|\bar{Q}|$ and $|\bar{R}|$ increase like $|\bar{T}|$ and $|\bar{U}|$). Nevertheless, we argue that storing and reviewing the new concepts and relations can help understanding if the requirements specification provides additional information with respect to the input documents.

6 Pilot Test

We have performed a pilot test to assess the effectiveness of the proposed approach, and to evaluate the correlation between the backward functional completeness and the forward functional completeness (**Research Question 3**) of a requirements specification.

In the pilot test, the first and third author, referred as subject A and subject B, were required to write requirements for an ATS system, according to the generic requirements provided by the standard IEEE Std 1474.1-2004 [5].

The requirements have been written with the support of the tool, and without the support of the tool. The goal was to compare the degree of backward functional completeness and the degree of forward functional completeness achieved in the two cases.

More specifically, the pilot test required four steps, which are described below.

1. Input document reading: the chapter concerning the ATS of the IEEE Std 1474.1-2004 [5] - about 5 pages long - was used as input document for the requirements definition task. Subject A and B were asked to read the input document to have a first understanding of the general needs of the system.
2. Tool set-up: from the input document, 67 relevant terms and 316 relevant relations have been automatically extracted. To this end, a threshold of 99% and a threshold of 10 were chosen as domain relevance threshold τ, and relation degree threshold ρ, respectively. In Table 1 and 2, we provide representative examples of relevant terms and relevant relations extracted from the document. These terms and relations have been fed into the tool to support the definition of the requirements.
3. Requirements definition Phase 1: subject A and B were asked to write the requirements. Subject A operated with the support of the tool, and subject B operated without the tool. The requirements definition lasted one hour.
4. Requirements definition Phase 2: subject A and B were asked again to write the requirements. Subject B operated with the tool, and subject A operated without the tool until they produced the same amount of requirements produced in the previous step (i.e., if a subject produced n requirements in Phase 1, he should have produced n requirements also in the Phase 2). Given a subject, this choice allows comparing the completeness scores achieved in the two phases on the same amount of requirements.

The subjects chosen for the test - first and third author - were involved in the definition of the principles of CAR, while the approach for term/relation extraction was defined and implemented by the second author only. Therefore, we argue that the expectations of the two test subjects on the success of the solution had a limited influence on the result of the test. Indeed, they did not know which types of terms/relations would be considered relevant by the tool, and could not influence the test by avoiding the usage of relevant terms/relations when the tool was not used. This is especially true for Subject B, who performed his first experiment without the tool. But it is also true for Subject A, since during the first experiment he viewed only a limited part of the terms/relations extracted by the tool (i.e., the suggested terms/relations).

6.1 Quantitative Evaluation

We evaluated the results of the test by computing the backward functional completeness of the produced requirements for the two subjects. Then, we computed the forward functional completeness according to the metrics provided by España

Table 1. Examples of relevant terms

Term	**TRank (%)**	**Freqency**
CBTC	100.0	44
ATS	$99.99999 + 0.99769 \times 10^{-6}$	43
ATS system	$99.99999 + 0.8456 \times 10^{-6}$	19
ATS user interface	$99.99999 + 0.29614 \times 10^{-6}$	8
train location	$99.99999 + 0.1231 \times 10^{-6}$	7
train	$99.99999 + 0.1185 \times 10^{-6}$	57
conductor	$99.99997 + 0.73215 \times 10^{-6}$	8
station	$99.99979 + 0.57378 \times 10^{-6}$	12

Table 2. Examples of relevant relations

Relation	**RRank**	**Freqency**
(conductor, ATS system)	35.1402383629	6
(ATS user interface, position of trains)	17.9938334306	2
(station, train at station)	16.1777267317	2
(speed regulation function, service brake rates)	14.8834871304	1
(train fault reporting, train health data)	14.8834871304	1
(re-routing of trains, movement of trains)	14.8834871304	1
(equipment, supplier)	13.1023727742	2
(ATS user interface, movement authorities)	12.4872415276	2
(station departure time, train service)	12.1108984081	1

et al. [10]. The degree of functional encapsulation completeness $degFEC$, and the degree of linked communication completeness $degLCC$ require the definition of a reference model for the system. In our case, we have employed a preliminary system specification where functions and linked communications were listed. The reference model defines 21 functions and 10 linked communications for the ATS system. The document was edited in the context of the Trace-IT project, a project for technology-transfer, which involves ISTI-CNR and a medium-sized railway signalling company. It is worth noting that the reference model was provided *before* the definition of the method presented in this paper. Table 3 summarizes the results of the test.

Backward functional completeness. We see that, for both subjects, the backward functional completeness, estimated with $degCC$ and $degIC$, is higher when the tool is employed ($\Delta degCC = 12.7\%$ and $\Delta degIC = 8.6\%$ in average). Therefore, in our pilot test, the usage of the tool actually helped in improving the backward functional completeness of the requirements specification. Furthermore, we argue that if a larger amount of input documents would be employed, the benefit given by the usage of the tool would be even more evident. The CAR tool helps in the navigation of the input documents. Without tool support, coherent navigation would be hardly practicable in the case of many documents. Moreover, with a larger amount of information, the statistics that bring to the set of relevant terms/relations would be more accurate, and the consequent suggestions given by the tool would be more meaningful.

Table 3. Results of the pilot tests

Subject	Num. Reqs	Tool	*degCC*	*degIC*	*degFEC*	*degLCC*
A	36	Yes	68.7%	17.1%	47.6%	40%
		No	52.3%	12.8%	61.9%	50%
B	21	Yes	67.2%	24.5%	47.6%	50%
		No	58.2%	11.6%	33.3 %	50%

Forward Functional Completeness. Conflicting results have been found concerning the effectiveness of the approach with respect to forward functional completeness, estimated through $degFEC$ and $degLCC$. Indeed, we see that subject A achieved a lower value for both metrics when using the tool with respect to the values obtained when the tool was not employed ($\Delta degFEC = -14.3\%$, $\Delta degLCC = -10\%$). Instead, subject B achieved a higher value for $degFEC$ when using the tool ($\Delta degFEC = 14.3\%$), while equivalent values for $degLCC$ were obtained in Phase 1 and 2. Therefore, from our test, we cannot identify a positive correlation between the degree of backward functional completeness and the degree of forward functional completeness. Instead, we argue that the results obtained might be related to the *order* that was followed by the two subjects in performing the tasks. Subject A performed the experiment with CAR *before* writing the requirements without the tool, while for subject B was the other way around. Both subjects achieved a higher degree of completeness during Phase 2. Basically, a higher degree of completeness was obtained when the subjects acquired a higher confidence with the topic of the requirements, since they already defined requirements for the system in Phase 1.

6.2 Qualitative Evaluation

We have performed a qualitative analysis of the produced requirements to understand which were the main differences between the requirements produced with CAR and those produced without the tool. Interesting results have been found. We have identified two main differences: 1) requirements produced with CAR tend to be more specific, while requirements produced without the tool are more high-level; 2) requirements produced with CAR tend to identify alternative behaviors of the system. Representative examples of requirements produced *without* the support of the tool by subject A are:

- R_1. *The ATS system shall send the desired speed profile to the trains*
- R_2. *The ATS system shall have the capability to define temporary speed restrictions for the trains*
- R_3. *The ATS system shall implement the functionality of train routing*

These requirements are quite generic, and do not add too much content compared to the input document. Instead, more specific requirements are produced with the tool. For example, the following requirement was produced when the tool suggested the term "emergency brake application" and the relations <"response", "wet rail">: *The ATS system shall adjust the speed profile of the trains in* response *to* wet rail *conditions in order to avoid* emergency brake application". Such requirement can be regarded as a specialization of R_1 and R_2, since it explains the specific condition (i.e., the wet rail) that requires temporary speed restrictions. The following requirement is an example of an alternative behavior identified with the support of the tool. In this case, the relations suggested was <"re-routing", "service disruptions">: *The ATS system shall be capable of supporting* re-routing *of trains in response to* service disruptions*"*. This requirement shows an alternative behavior (i.e., re-routing) of the routing functionality identified by requirement R_3. According to this preliminary analysis, we argue that the proposed tool can play a complementary role during requirements definition. Indeed, it can be used as a support tool to identify specific cases, and alternative behaviors that tend to be overseen in requirements definition approaches based solely on the analysis of the input documents.

7 Conclusions

In this paper, the novel concept of *backward functional completeness* of a requirements specification has been defined as the completeness of a specification with respect to the input documents of the requirements definition process. Metrics to measure such completeness have been provided, as well as a NLP-based tool named CAR to improve it. Further development of the principles of CAR are currently under analysis. We would like to give a *type* to the relations that are extracted from the input documents. For example, "ATS user interface" and "train schedule" are related in our input document, and their relation is of type "display". Furthermore, we would like to explore different approaches for choosing the terms to be suggested to the user of CAR. Such approaches should also take into account the structure of the input documents, the structure of the requirements specification itself, and the requirements previously written by the user. Other similarity metrics, such as the *cosine similarity*, are currently under analysis to evaluate the relations among the terms.

After improving the principles of CAR, we plan to assess the tool with both academic and industrial case studies. In particular, we plan to consider systems of different domains, as well as different types of input documents, in order to identify possible refinements and domain-specific optimizations of the approach.

Acknowledgements. This work was partially supported by the PAR FAS 2007-2013 (TRACE-IT) project.

References

1. Rayson, P., Garside, R., Sawyer, P.: Recovering legacy requirements. In: Proc. of REFSQ 1999, pp. 49–54 (1999)
2. Pohl, K., Böckle, G., Van Der Linden, F.: Software product line engineering: foundations, principles, and techniques. Springer (2005)
3. Goldin, L., Berry, D.M.: Abstfinder, a prototype natural language text abstraction finder for use in requirements elicitation. Autom. Softw. Eng. 4(4), 375–412 (1997)
4. Ambriola, V., Gervasi, V.: On the systematic analysis of natural language requirements with CIRCE. Autom. Softw. Eng. 13(1), 107–167 (2006)
5. IEEE: IEEE Standard for Communications Based Train Control (CBTC) Performance and Functional Requirements. IEEE Std 1474.1-2004 (Revision of IEEE Std 1474.1-1999) (2004)
6. Lauesen, S.: Software Requirements: Styles and Techniques. Addison-Wesley (2002)
7. Boehm, B.: Verifying and validating software requirements and design specifications. IEEE Software 1(1), 75–88 (1984)
8. Zowghi, D., Gervasi, V.: The three cs of requirements: Consistency, completeness, and correctness. In: Proc. of REFSQ 2002, pp. 155–164 (2002)
9. Lindland, O., Sindre, G., Solvberg, A.: Understanding quality in conceptual modeling. IEEE Software 11(2), 42–49 (1994)
10. España, S., Condori-Fernandez, N., Gonzalez, A., Pastor, O.: Evaluating the completeness and granularity of functional requirements specifications: A controlled experiment. In: Proc. of RE 2009, pp. 161–170 (2009)
11. Yadav, S.B., Bravoco, R.R., Chatfield, A.T., Rajkumar, T.M.: Comparison of analysis techniques for information requirement determination. Commun. ACM 31(9), 1090–1097 (1988)
12. Davis, A., Overmyer, S., Jordan, K., Caruso, J., Dandashi, F., Dinh, A., Kincaid, G., Ledeboer, G., Reynolds, P., Sitaram, P., Ta, A., Theofanos, M.: Identifying and measuring quality in a software requirements specification. In: Proc. of SMS 1993, pp. 141–152 (1993)
13. Costello, R.J., Liu, D.B.: Metrics for requirements engineering. J. Syst. Softw. 29(1), 39–63 (1995)
14. Menzel, I., Mueller, M., Gross, A., Dörr, J.: An experimental comparison regarding the completeness of functional requirements specifications. In: Proc. of RE 2010, pp. 15–24 (2010)
15. Kaiya, H., Saeki, M.: Ontology based requirements analysis: lightweight semantic processing approach. In: Fifth International Conference on Quality Software (QSIC 2005), pp. 223–230 (2005)
16. Bonin, F., Dell'Orletta, F., Montemagni, S., Venturi, G.: A contrastive approach to multi-word extraction from domain-specific corpora. In: Proc. of LREC 2010, pp. 19–21 (2010)
17. Dell'Orletta, F.: Ensemble system for part-of-speech tagging. In: Proc. of Evalita 2009, Evaluation of NLP and Speech Tools for Italian (2009)
18. Dunning, T.: Accurate methods for the statistics of surprise and coincidence. Comput. Linguist. 19(1), 61–74 (1993)