CORE
Metadata, citation and similar papers at core.ac.uk
Provided by DSpace at VU

Appl Intell (2007) 27:291–301
DOI 10.1007/s10489-007-0037-3

# A specification language for organisational performance indicators

**Viara Popova · Jan Treur**

Received: 8 August 2006 / Accepted: 16 January 2007 / Published online: 10 February 2007
© Springer Science + Business Media, LLC 2007

**Abstract** A specification language for performance indicators and their relations and requirements is presented and illustrated for a case study in logistics. The language can be used in different forms, varying from informal, semiformal, graphical to formal. A software environment has been developed that supports the specification process and can be used to automatically check whether performance indicators or relations between them or certain requirements over them are satisfied in a given organisational process.

**Keywords** Performance indicator · Organisation · Requirement · Language

## 1 Introduction

In organisational design, redesign or change processes, organisational performance indicators form a crucial source of information; cf. [15]. Within such processes an organisation is (re)designed to fulfil (better) the performance indicators that are considered important. In this manner, within organisational (re)design processes, performance indicators function as requirements for the organisational processes.

Within the domain of software engineering in a similar manner, requirements play an important role. Software is (re)designed to fulfil the requirements that are imposed. The use of requirements within a software engineering process has been studied in more depth during the last decades; it has led to the area called Requirements Engineering; cf. [6, 11, 16]. Formal languages to express requirements, and automated tools have been developed to support the specification process (from informal to formal) and to verify or validate whether they are fulfilled by a designed software component.

In this paper it is investigated how some of the achievements in Requirements Engineering can be exploited in the field of organisational performance indicators. Inspired by requirement specification languages, a formal language to specify performance indicators and their relationships is proposed, and illustrated by various examples. It is shown how this language or subsets thereof can be used in informal, graphical or formal form. Performance indicators expressed in this language can be manipulated by a software environment to obtain specifications or to evaluate performance indicators against given traces of organisational processes.

The organization of the paper is as follows. First, in Section 2, the concept of an organisational performance indicator is briefly introduced. In Section 3 the formal specification language is introduced. It is shown how the proposed language can be used to express the indicators themselves, but also how they relate to each other and in what sense they are desirable. Section 4 is dedicated to the analysis of conflicts between qualified requirements. A technique for detecting potential conflicts from a specification of relationships and qualified requirements is presented for which an implementation is available. Next, in Section 5, a case study

This paper is an extended version of the paper which appeared in: M. Ali and F. Esposito, editors, Proceedings of 18th International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems, IEA/AIE 2005, vol. 3533 of Lecture Notes in Artificial Intelligence, pp. 667–677, Springer Verlag, 2005.

V. Popova · J. Treur (✉)
Department of Artificial Intelligence, Vrije Universiteit Amsterdam, De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
e-mail: treur@few.vu.nl

V. Popova
e-mail: popova@few.vu.nl

of the use of the language for the domain of third-party logistics is presented. Section 6 concludes the paper with a discussion.

## 2 Organisational performance indicators

In order to assess its performance, it is crucial for an organisation to identify its key performance indicators. Performance indicators are metrics that show the state of the company—they can be monitored and analysed to give a clear view on the current functioning. Furthermore they can also be used for defining objectives that need to be achieved by the company or for comparing to competitors or the industry benchmarks.

The set of possible performance indicators is large, diverse and domain- and company-specific. Monitoring and evaluating these indicators is difficult and resource intensive if at all possible. It is therefore important to identify the most important ones—the key performance indicators that reflect the position, focus and objectives of the company. For one company working in a domain where each client tends to appear only once or not very regularly and make small orders (e.g., a small airport drugstore) the number of clients might be a key performance indicator. For another company, however, which relies on having regular clients and a good reputation (e.g., a local cafe) the number of orders per client or the satisfaction of the client might be more important. Historically, companies considered mainly financial indicators. Nowadays it is widely recognised that non-financial and even non-numerical indicators can give valuable information as well [3, 4, 9] (e.g., customer or employee satisfaction, motivation, safety, etc.).

While identifying the key performance indicators and the objectives for desired or expected performance of the company it is beneficial to be aware of how these indicators are related to each other. For example it is not uncommon that some performance indicators turn out to be conflicting—improving one may worsen another [10]. Other types of relationships are possible as well. There exists a large amount of research on identifying and classifying (key) performance indicators for different domains. However, extensive research on how these indicators relate to and influence each other is lacking. Such insight is important in identifying how the key performance indicators and the objectives related to them can be reflected into the concrete planning for the organisation activities in order to achieve these objectives. Moreover, it can be used in organisational change and redesign processes to be undertaken when the performance is considered critically low and measures are needed to improve it by performing more extensive changes to the structure and behaviour of the organisation.

Automated analysis performed in a specialized software environment can provide assistance in these processes. In this paper it is argued that it is necessary to model in a precise manner the performance indicators, their relationships and the objectives explicitly in order to achieve higher variety of analysis tools. A formal language which provides the possibility to specify such information is a basis for this perspective. Such a language inspired by the area of Requirement Engineering is proposed in the next section.

## 3 A formal specification language for performance indicators

The starting point of this research is in the area of requirements engineering as applied within the process of design of software systems. The approach we adopt uses logic as a tool in the analysis (see for example [1, 2, 13]) and more specifically order-sorted predicate logic which employs sorts for naming sets of objects. Such an extension of first order logic by a sort hierarchy increases the clarity and intuitiveness in the description of the domain area.

In the following subsection we introduce the language by defining the sorts, predicates and functions included in it. We start with the simplest constructs on the level of the performance indicators and build on this basis to introduce constructs describing relationships between them and requirements imposed on the indicators.

### 3.1 Performance indicators

First we consider single performance indicators and lists of indicators. The sorts that we define are given in Table 1.

Based on these sorts we define a predicate that allows us to give names to lists of indicators for ease of reference:

IS-DEFINED-AS : INDICATOR-LIST-NAME × INDICATOR-LIST

In order to demonstrate the use of this and other predicates, we use a running example for the rest of this section. The domain area is logistics from the point of view of a logistics service provider. Table 2 gives the indicators included in the example.

**Table 1** Sorts defined on indicators and lists of indicators

| Sort name | Description |
|---|---|
| INDICATOR-NAME | The set of possible names of performance indicators |
| INDICATOR-LIST | The set of possible lists of performance indicators |
| INDICATOR-LIST-NAME | The set of possible names for lists of performance indicators |

**Table 2** An example set of performance indicators

| Indicator name | Description | Indicator name | Description |
|---|---|---|---|
| NC | Number of customers | ISC | Information system costs |
| NNC | Number of new customers | FO | % of failed orders |
| NO | Number of orders | SB | Salaries and benefits |
| ND | Number of deliveries | AP | Attrition of personnel |
| MP | Motivation of personnel | | |

The above defined predicate can be used as follows:

IS-DEFINED-AS(COD, [NC, NO, ND]).

The definitions given in this subsection are fairly simple but they give us the basis for going one level higher and exploring the possible relationships between indicators.

## 3.2 Relationships between performance indicators

Performance indicators are not always independent. Often they are connected through complex relationships such as correlation (the indicators tend to change in a similar way) or causality (the change in one indicator causes the change in another). Often we would like to know whether these relationships are positive or negative, e.g. correlation can be positive (the indicators increase together) or negative (one increases and the other one decreases). Therefore we need a new sort given in Table 3.

Now we are ready to define predicates for the relationships we would be interested in. First we define a predicate for correlation as follows:

CORRELATED: INDICATOR-NAME $\times$ INDICATOR-NAME $\times$ SIGN

Causality relation between two indicators is denoted with the following predicate:

IS-CAUSED-BY: INDICATOR-NAME $\times$ INDICATOR-NAME $\times$ SIGN

Examples: CORRELATED(NC, NO, pos), IS-CAUSED-BY(AP, MP, neg)

In a similar way we can define a predicate for cases where one indicator is included in another by definition, e.g. one indicator is the sum of a number of other indicators:

IS-INCLUDED-IN: INDICATOR-NAME $\times$ INDICATOR-NAME $\times$ SIGN

Example: IS-INCLUDED-IN(NNC, NC, pos)

Another predicate is used for indicating different aggregation levels of the same indicator, e.g. measured by day/month/year (temporal aggregation) or by employee/unit/company (organizational aggregation):

IS-AGGREGATION-OF: INDICATOR-NAME $\times$ INDICATOR-NAME

A set of indicators can be independent (no significant relationship plays a role) or divergent (correlation, causality or inclusion in a negative way) denoted in the following way:

INDEPENDENT: INDICATOR-NAME $\times$ INDICATOR-NAME
DIVERGENT: INDICATOR-NAME $\times$ INDICATOR-NAME

Examples: INDEPENDENT (ISC, FO), $\neg$ DIVERGENT (NC, ISC)

It might also be the case that we can easily replace measuring one indicator with measuring another one if that is necessary—it is expressed as follows:

TRADE-OFF-SET: INDICATOR-NAME $\times$ INDICATOR-NAME

While the meaning of the indicators might be similar it might still be the case that measurement for one can be more expensive to obtain than for the other one. Such a relationship is also important to consider when we choose which particular set of indicators to measure. It is denoted using the predicate:

IS-COSTLIER-THAN: INDICATOR-NAME $\times$ INDICATOR-NAME

The relationships discussed so far can be represented graphically using a conceptual graph (see [17, 18]). Conceptual graphs have two types of nodes: concepts and relations. In our case the first type will represent the indicator names

**Table 3** Additional sorts used in defining relationships between indicators

| Sort name | Description |
|---|---|
| SIGN | The set {pos, neg} of possible signs that will be used in some relationship formulas |

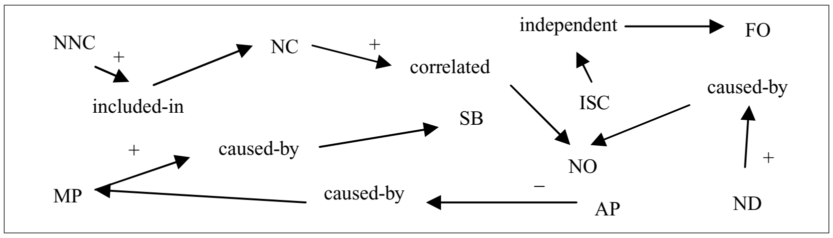

**Fig. 1** The conceptual graph of relationships between the indicators

while the second type represents the relations between them. The nodes are connected by arrows in such a way that the resulting graph is bipartite—an arrow can only connect a concept to a relation or a relation to a concept. Some of the predicates that we defined have an additional attribute of sort SIGN. In order to keep the notation simple we do not represent it as a concept node but as an extra sign associated to the arc: '$+$' for positive relationships and '$-$' for negative ones. Figure 1 is a small example of how such a conceptual graph would look like. We use here the examples given to illustrate the predicates in this section and represent them in the graph.

We now define one more predicate over a list of indicators. It will be used to indicate whether the set of indicators is minimal, where by minimal we imply that these three constraints are satisfied: no two indicators are replaceable, none is a different aggregation level of another and none is used in the definition of another:

MINIMAL: INDICATOR-LIST-NAME

Note that while such property of the indicator set is interesting to consider, it does not mean that we are only interested in minimal sets.

## 3.3 Requirements over performance indicators

The previous subsection concentrated on relationship between performance indicators. Going one more level higher we can define our own preferences over the set of indicators—what we prefer to measure and how we should evaluate the results. First we consider the second question by defining qualified expressions.

*Qualified expressions*. Qualified expressions specify what we consider 'a success', i.e. when we consider one measurement of an indicator better than another one. Such specifications can be as simple as 'higher value is preferred over a lower one' or more complex such as 'the value should approximate a certain optimal value while never exceeding a predefined maximal value'.

The sorts that need to be added to our list are given in Table 4.

The sort VARIABLE-EXPRESSION contains expressions defining constraints over a variable as in the following examples:

$v < \text{maxKD}$ (where v is a variable and maxKD is a constant),
$v > \text{minKD} \wedge v \leq \text{maxKD}$ (where minKD is also a constant),
$v \leq \text{minKD} \vee v > \text{maxKD}$,
etc.

The sort INDICATOR-VARIABLE-EXPRESSION on the other hand, contains expressions defining to which indicator the variable refers. Here we use the function:

has-value: INDICATOR $\times$ VARIABLE
$\rightarrow$ INDICATOR-VARIABLE-EXPRESSION

**Table 4** The sorts concerning qualified expressions

| Sort name | Description |
|---|---|
| VARIABLE | The set of possible variables over the values of indicators |
| INTEGER | The set of integers |
| INDICATOR-VARIABLE-EXPRESSION | The set of expressions over an indicator and its corresponding variable (see the definition below) |
| VARIABLE-EXPRESSION | The set of expressions over a variable (see examples below) |
| QUANTIFIER | The set of possible quantifiers (see the definitions below) |
| QUALIFIED-EXPRESSION | The set of possible qualified expressions (see below) |
| QUALIFIED-EXPRESSION-NAME | The set of possible names for qualified expressions |
| QUALIFIED-EXPRESSION-LIST | The set of possible lists of qualified expressions |
| QUALIFIED-EXPRESSION-LIST-NAME | The set of possible names for lists of qualified expressions |

For example the expression has-value(NNC, v) indicates that the variable v refers to the values of the indicator NNC. We now define the following functions that return objects of the type QUANTIFIER:

minimize, maximize: VARIABLE → QUANTIFIER
approximate: VARIABLE × CONSTANT → QUANTIFIER
satisfy: VARIABLE-EXPRESSION → QUANTIFIER

Examples: minimize(v), approximate(v, bestKD), satisfy(v < maxKD)

A qualified expression is identified by a quantifier and an indicator-variable expression. The following function given such a couple returns a qualified expression:

Qualified-expression: QUANTIFIER × INDICATOR-VARIABLE-EXPRESSION → QUALIFIED-EXPRESSION

As an example consider the expression Qualified-expression (min(v), has-value(ISC, v)), which should be read as: 'minimize the value v of the performance indicator ISC'. The following predicates can also be added to our set of predicates:

IS-DEFINED-AS: QUALIFIED-EXPRESSION-NAME × QUALIFIED-EXPRESSION
IS-DEFINED-AS: QUALIFIED-EXPRESSION-LIST-NAME × QUALIFIED-EXPRESSION-LIST

Example: IS-DEFINED-AS (q, Qualified-expression (max(v), has-value(NNC, v)))

*Qualified requirements*. Building on the notion of qualified expressions, we can now define qualified requirements stating our preferences among the possible qualified expressions. We first introduce a number of new sorts (Table 5).

We can now define the following function which returns a qualified requirement:

Requirement: QUALIFICATION × QUALIFIED-EXPRESSION-LIST → QUALIFIED-REQUIREMENT

Example: Requirement(desired, Qualified-expression (max(v), has-value(NC, V)))

This can be read as: 'it is desired to maximize the value v of the performance indicator NC'. For simplicity, we abuse the notation by interchanging a qualified expression and a list of one qualified expression. Another example could look like:

Requirement(preferred-over,
[Qualified-expression (max(v1), has-value(NC, V1)),
Qualified-expression (max(v2), has-value(NNC, V2))])

Here the list indicates that the first qualified expression (the head of the list) is preferred over the rest of the expressions (the tail of the list). Other possible qualifications expressing different degrees of desirability can be: required, highly_desired, weakly_desired, etc.

We define further the following two predicates:

IS-DEFINED-AS: QUALIFIED-REQUIREMENT-NAME × QUALIFIED-REQUIREMENT
IS-DEFINED-AS: QUALIFIED-REQUIREMENT-LIST-NAME × QUALIFIED-REQUIREMENT-LIST

Example: IS-DEFINED-AS (r, Requirement(desired, Qualified-expression(min(v1), has-value(AP, v1))))

## 3.4 Levels of formalisation of requirements

Within the area of Requirements Engineering, which served as a source of inspiration for the work reported here, methods have been described to aid the modeller

**Table 5** The sorts concerning qualified requirements

| Sort name | Description |
|---|---|
| QUALIFICATION | The set of possible qualifications that can be used in a qualified requirement |
| QUALIFICATION-NAME | The set of possible names for qualifications |
| QUALIFIED-REQUIREMENT | The set of possible qualified requirements |
| QUALIFIED-REQUIREMENT-NAME | The set of possible names for qualified requirements |
| QUALIFIED-REQUIREMENT-LIST | The set of possible lists of qualified requirements |
| QUALIFIED-REQUIREMENT-LIST-NAME | The set of possible names for lists of qualified requirements |

in formalisation of requirements. For example, in [8] it is extensively described how requirements initially formulated informally in natural language and/or graphical elements can be restructured into a more standard structured natural language format, which then can be reformulated more easily in a formal language. Using inspiration from the methods in [8], similar conversion can be applied in the case of qualified requirements as defined in this paper. As an illustration, consider the following simple example of a requirement on safety expressed in natural language:

The safety regulations for production process must not be violated.

This can be reformulated into a more structured form as follows:

It is required that the performance indicator Safety regulations violations is equal to zero.

A formalisation can be made by using formal ontologies for the concepts, and by formalising the relationships, which in the case of qualified requirements can result in such a formulation:

Requirement(required, Qualified-expression(satisfy (v= 0), has-value(SRV, v))).

For a more extensive discussion about the transition from informal to formal, see [8].

## 4 Conflicts analysis between qualified requirements

The language presented in Section 3 gives a formal basis for performing analysis on the performance indicators, their relationships and the qualified requirements defined on them. One type of analysis that will be described in this section is detection of conflicts between requirements. Intuitively, a conflict indicates that the two requirements potentially cannot be satisfied together and is represented by the predicate:

CONFLICTING: QUALIFIED-REQUIREMENT-NAME
× QUALIFIED-REQUIREMENT-NAME.

This can happen for instance when, due to correlation, causality or aggregation relationship, certain movement of one indicator is associated with certain movement of the other, however the corresponding requirements prescribe the opposite of this relation. An example would be two indicators that are positively correlated but the requirements specify one to be maximized and the other one to be minimized. Such relation over the set of requirement is important because often in practice conflicting needs arise and we must take special care in dealing with this.

A simple example for such a situation can be given from the set of indicators listed in Table 2. The company management knows that the salaries and benefits contribute to the total costs and therefore reduce the profit. Thus the following requirement can be considered:

IS-DEFINED-AS(r1, Requirement(desired, Qualified-expression (min(v1), has-value(SB, v1))))

At the same time the management wants to minimize the attrition of employees as that increases the costs for teaching new employees and decreases the average productivity. Therefore another requirement can be considered:

IS-DEFINED-AS(r2, Requirement(desired, Qualified-expression (min(v1), has-value(AP, v1))))

But decreasing the salaries will lead to increase in the attrition of personnel; therefore the two requirements are conflicting: CONFLICTING (r1, r2).

More formally, we define conflicts in the following way. We consider three types of conflicts which are defined separately. The first type, *self-conflicts*, appear when two or more qualified requirements are defined over the same performance indicator and one of them requires maximization while another requires minimization. More formally it can be defined by the following rule:

$\forall$ (i: INDICATOR-NAME; v1: INTEGER; r1, r2 : QUALIFIED-REQUIREMENT-NAME)
IS-DEFINED-AS(r1, Requirement(desired, Qualified-expression (min(v1), has_value(i1,v1)))) $\wedge$
IS-DEFINED-AS(r2, Requirement(desired, Qualified-expression (max(v1), has_value(i1,v1)))))
$\Rightarrow$ CONFLICTING(r1, r2)

The second type of conflicts are *primary conflicts* which appear when either two performance indicators are related by a positive relation (positive causality, correlation, inclusion or aggregation) and opposite qualified requirements are defined on them or when they are related by a negative relation (negative causality or correlation) and the same type of qualified requirements are defined on them. This can be expressed formally by the following rules. The first one states that requirements based on positively related indicators such that one is required to be maximized and the other one to be minimized will be conflicting.

$\forall$(i1, i2: INDICATOR-NAME; v1, v2: INTEGER; r1, r2: QUALIFIED-REQUIREMENT-NAME)
(CORRELATED(i1, i2, pos) $\vee$ IS-INCLUDED-IN(i1, i2, pos) $\vee$

CAUSED-BY(i1, i2, pos) ∨IS-AGGREGATION-OF(i1, i2))∧
IS-DEFINED-AS(r1, Requirement (desired, Qualified-expression (max (v1), has_value(i1,v1)))) ∧
IS-DEFINED-AS(r2, Requirement (desired, Qualified-expression (min (v2), has_value(i2,v2)))))
⇒ CONFLICTING (r1, r2)

Similarly, negatively related indicators required to 'move in the same direction' will also generate conflicting qualified requirements which is expressed in the following two rules:

∀ (i1, i2: INDICATOR-NAME; v1, v2: INTEGER; r1, r2: QUALIFIED-REQUIREMENT-NAME)
(CORRELATED(i1, i2, neg) ∨ IS-INCLUDED-IN(i1, i2, neg) ∨ CAUSED-BY(i1, i2, neg)) ∧
IS-DEFINED-AS(r1, Requirement (desired, Qualified-expression (min (v1), has_value(i1,v1)))) ∧
IS-DEFINED-AS(r2, Requirement (desired, Qualified-expression (min (v2), has_value(i2,v2)))))
⇒ CONFLICTING(r1, r2)

∀ (i1, i2 : INDICATOR-NAME; v1, v2:INTEGER; r1, r2: QUALIFIED-REQUIREMENT-NAME)
(CORRELATED(i1, i2, neg) ∨ IS-INCLUDED-IN(i1, i2, neg) ∨ CAUSED-BY(i1, i2, neg)) ∧
IS-DEFINED-AS (r1, Requirement (desired, Qualified-expression (max (v1), has_value(i1,v1)))) ∧
IS-DEFINED-AS (r2, Requirement (desired, Qualified -expression (max (v2), has_value(i2,v2)))))
⇒ CONFLICTING (r1, r2)

Furthermore *secondary conflicts* appear when a directed path exists in the conceptual graph between two performance indicators such that either the sign of the path is positive and the defined qualitative requirements have opposite directions or the sign of the path is negative and the defined qualitative requirements have the same direction. The sign of a path is defined to be positive if all relations in the path are positive, i.e., causality or correlation with a positive sign, included_in or aggregation_of. The sign of a path is defined to be negative if all but one relation in the path are positive and the remaining relation is causality or correlation with a negative sign. Formally this can be defined by the following rules where PATH_BETWEEN: INDICATOR-NAME × INDICATOR-NAME × SIGN is true if there exists a path from i1 to i2 having the specified sign:

∀ (i1, i2: INDICATOR-NAME; v1, v2: INTEGER; r1, r2: QUALIFIED-REQUIREMENT-NAME)
(IS-DEfiNED-AS(r1, Requirement (desired, Qualified-expression (max (v1), has_value(i1,v1)))) ∧
IS-DEFINED-AS(r2, Requirement (desired, Qualified-expression (max (v2), has_value(i2,v2)))) ∧
PATH_BETWEEN(i1, i2, neg))
⇒ CONFLICTING (r1, r2)

∀ (i1, i2: INDICATOR-NAME; v1, v2: INTEGER; r1, r2: QUALIFIED-REQUIREMENT-NAME)
(IS-DEFINED-AS(r1, Requirement (desired, Qualified-expression (min (v1), has_value(i1,v1)))) ∧
IS-DEFINED-AS (r2, Requirement (desired, Qualified-expression (min (v2), has_value(i2,v2)))) ∧
PATH_BETWEEN(i1, i2, neg))
⇒ CONFLICTING (r1, r2)

∀ (i1, i2 : INDICATOR-NAME; v1, v2: INTEGER; r1, r2: QUALIFIED-REQUIREMENT-NAME)
(IS-DEFINED-AS(r1, Requirement (desired, Qualified -expression (max (v1), has_value(i1,v1)))) ∧
IS-DEFINED-AS (r2, Requirement (desired, Qualified: expression (min (v2), has_value(i2,v2)))) ∧
PATH_BETWEEN(i1, i2, pos))
⇒ CONFLICTING (r1, r2):

∀ (i1, i2 INDICATOR-NAME; v1, v2 : INTEGER;r1, r2: QUALIFIED-REQUIREMENT-NAME)
(IS-DEFINED-AS (r1, Requirement (desired, Qualified: expression (min (v1), has_value(i1,v1)))) ∧
IS-DEFINED-AS (r2, Requirement (desired, Qualified: expression (max (v2), has_value(i2,v2)))) ∧
PATH_BETWEEN(i1, i2, pos))
⇒ CONFLICTING (r1, r2)

Note that the definition of the sign for a path does not consider all possible combinations of relationships. The reason for that is that in the rest of the situations no conclusion can be given on whether the qualified requirements are potentially conflicting. For example if a path of length 2 contains two negative relations this does not mean that they 'cancel' each other and the path sign is considered positive. In such situation the sign is considered undetermined.

Conflicts, as defined above, and especially secondary conflicts, are difficult to find manually or just by looking at the conceptual graph. Therefore we have built a software tool that can detect all conflicts given a conceptual graph and a set of qualified requirements defined on a subset of the set of performance indicators in it. The program first checks for self-conflicts. Then it creates a copy of the graph structure on which it propagates the relations whenever possible in the following way: if there exists a positive relation from indicator i1 to i2 and from i2 to i3 then a new relation is created from i1 to i3 with a positive sign. Similarly, a positive and a negative relation result in a new negative relation. In this way all paths (according to the definition given earlier) are discovered with their signs. Finally the new structure is matched with the defined qualified requirements and all discovered primary conflicts are outputted. These conflicts correspond to all primary and secondary conflicts in the original graph.

When the conflicts are discovered it is the responsibility of the designer/analyst to decide what needs to be done. A conflict does not necessarily mean that the graph and/or qualified requirements are incorrect, although that is a possibility that might be considered if the results are counterintuitive. Conflicts indicate that it might be problematic to satisfy both requirements at the same time. It is up to the designer to find out whether this would really lead to a problem or not. For example, it might be known that decreasing the salaries of the employees will reduce the costs but will also reduce the employees' motivation which will lead to lower quality, high attrition of personnel, etc., thus, also increasing the costs. Therefore there will be conflicts between the requirements to minimize salaries and costs and to maximize the personnel's motivation. Further analysis might show that the expected increase in costs due to lack of motivation is much lower than the decrease achieved by decreasing the salaries. If, however, that is not the case, it might be decided to drop the requirement to minimize the salaries or an additional factor might be chosen to increase the motivation to an acceptable level (e.g., measures to improve the work atmosphere, additional non-financial benefits, etc.). In such analysis, the defined preferences over requirements are used as indicators about the relative importance and taken into account in the final decision.

## 5 A case study from the area of logistics

In this Section we discuss a case study from the area of third-party logistics (3PL) and apply the approach presented in the previous Section. 3PL companies are specialized in providing logistics services to other companies [5, 7, 19]. Logistics service providing companies often operate under great pressure in an industry with small margins where customers increasingly expect shorter delivery times and more accurate services. It is therefore necessary for the management to continuously look for opportunities to improve the company's profitability—this can be achieved for example by scaling up or expanding the activities to a wider region [14].

Planning and control, both daily and on the long-term, play a crucial role in the operations of a logistics service provider. A good insight in the performance information and how it should be used for steering the planning is therefore also important. However the activities on defining, monitoring and analysing performance indicators are usually done in an ad-hoc manner.

Important performance aspects for the area of third-party logistics typically include efficiency in transportation (e.g., reduction of transportation costs, improvement of route planning, equipment and labour utilization, etc.), customer satisfaction, employee satisfaction (in order to reduce the attrition of drivers), etc. For a literature review on performance measurement in logistics, a classification framework and a list of performance indicators see [12].

Our case study is inspired by a Netherlands based medium sized logistics service provider which operates 40 trucks in its container business. Planning there is still a manual task performed daily by three full-time planners, however, they are partly supported by information and communication technology. They utilize a platform which enables them to track and trace trucks and carriers—based on their GPS location—every single minute all throughout Europe. The planning as such uses a computer application as well, but this is not more than a list of orders to execute; the order-to-truck-assignment is done by the planner. The (manual) assigning is based on simple (unwritten) heuristic rules such as: if a new order is available at the place where the previous order ended, take this order—therefore reducing the amount of empty kilometres. The planners do utilize other performance indicators as well, such as the employees' satisfaction. Furthermore, we found support for the use of indicators such as customer happiness, which is of true importance in the planning process and company reputation. The planners use several objectives and rules for their planning, many of which are not well defined and documented.

This analysis was performed in the context of the DEAL project (DEAL stands for Distributed Engine for Advanced Logistics). The aims of this project, funded by the Ministry of Economic Affairs and a group of companies, are to develop an agent-based software system for road-distribution planning.

In the following subsections we present the results of the case study. We first introduce the set of indicators and formulate how they are related to each other. Then we define the set of possible (meaningful) requirements over the list of indicators and analyze them concentrating on detecting conflicts.

### 5.1 Performance indicators

The list of indicators is given in Table 6. It is based on real-life indicator sets used in logistics and is augmented by several additional indicators used in 3rd-party logistics. Furthermore, we added a couple of indicators that usually remain implicit in real-life performance measurement and have to do with employees' satisfaction and safety. Most of the indicators are typically numeric (costs, km, etc.), however, also non-numeric ones are included (employee motivation and safety). They can be modelled in different ways as long as the possible values are ordered in a consistent way.

### 5.2 Relationships

Looking closer at the indicators we see that many are not independent. The list below gives the most important relationships that we take into account.

**Table 6** The list of performance indicators considered in the case study

| Indicator name | Description | Indicator name | Description |
|---|---|---|---|
| TC | Total costs | TK | Total number of km |
| KD | Km/day | NT | Total number of trips |
| UV | Number of used vehicles | TO | Total number of orders |
| SO | % of served orders | R | Revenue |
| VO | % of violated orders | TP | Total profit TP = R − TC |
| TD | Trips per day | NA | Number of accidents |
| TT | Trips per truck | TS | Total amount for salaries |
| ST | Shops per truck | EM | Employee motivation (average) |
| NC | Number of clients | S | Safety |
| VP | % violations over the original plan | EP | Employee productivity (average) |

| | |
|---|---|
| RL1: IS-CAUSED-BY (TC, TK, pos) | RL2: IS-CAUSED-BY (TC, UV, pos) |
| RL3: CORRELATED (VO, SO, neg) | RL4: CORRELATED (TC, NT, pos) |
| RL5: CORRELATED (ST, TT, pos) | RL6: INDEPENDENT (SO, VP) |
| RL7: IS-CAUSED-BY (TC, VP, pos) | RL8: IS-INCLUDED-IN (R, TP, pos) |
| RL9: IS-INCLUDED-IN (TC, TP, neg) | RL10: IS-CAUSED-BY (R, TO, pos) |
| RL11: IS-CAUSED-BY (EP, EM, pos) | RL12: IS-CAUSED-BY (EM, KD, neg) |
| RL13: IS-INCLUDED-IN (TS, TC, pos) | RL14: IS-CAUSED-BY (EM, TS, pos) |
| RL15: CORRELATED (R, TK, pos) | RL16: IS-CAUSED-BY (TO, NC, pos) |
| RL17: IS-CAUSED-BY (R, NC, pos) | RL18: CORRELATED (NT, TO, pos) |
| RL19: IS-CAUSED-BY (EM, S, pos) | RL20: IS-CAUSED-BY (S, NA, neg) |
| RL21: IS-CAUSED-BY (TC, NA, pos) | RL22: IS-AGGREGATION-OF (TK, KD) |
| RL23: IS-AGGREGATION-OF (NT, TT) | RL24: IS-AGGREGATION-OF (NT, TD) |

These relationships can be expressed graphically using conceptual graphs as discussed earlier. Figure 2 gives the graph for our case study.

## 5.3 Requirements

We can now formulate qualified requirements over the set of indicators. Most of the requirements are in a similar form as the ones given in the examples in Section 3.3. RQ9 and RQ10 however are a bit more complex. RQ9 states that the value of the indicator KD should approximate a given constant called bestKD. RQ10 on the other hand, states that KD should not exceed another given constant maxKD. The intuition here is that the number of kilometers per day should approximate some pre-calculated optimal point but at the same time there exists a maximal value that does not allow the drivers to drive too much for health and safety reasons. Therefore the optimal point should be approximated in such a way that we do not exceed the maximal point. The list of qualified requirements is given below.

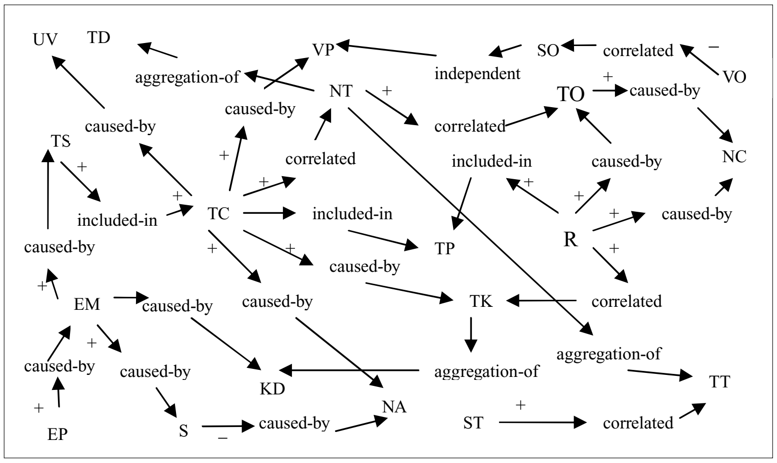

**Fig. 2** The conceptual graph for the case study

RQ1: Requirement (desired, Qualified-expression (min(v), has-value(TC, v)))

RQ2: Requirement (desired, Qualified-expression (min(v), has-value(VO, v)))

RQ3: Requirement (desired, Qualified-expression (min(v), has-value(VP, v)))

RQ4: Requirement (desired, Qualified-expression (max(v), has-value(R, v)))

RQ5: Requirement (desired, Qualified-expression (max(v), has-value(TP, v)))

RQ6: Requirement (desired, Qualified-expression (max(v), has-value(R, v)))

RQ7: Requirement (desired, Qualified-expression (max(v), has-value(EP, v)))

RQ8: Requirement (desired, Qualified-expression (max(v), has-value(TO, v)))

RQ9: Requirement (desired, Qualified-expression (max(v), has-value(KD,v)))

RQ10: Requirement (desired, Qualified-expression (min(v), has-value(KD,v)))

RQ11: Requirement (desired, Qualified-expression (max(v), has-value(NC, v)))

RQ12: Requirement (desired, Qualified-expression (max(v), has-value(S, v)))

RQ13: Requirement (preferred-over, Qualified-expression (min(v1), has-value(VO, v1)),
  Qualified-expression (max(v2), has-value(SO, v2)))

RQ14: Requirement (preferred-over, Qualified-expression (max(v1), has-value(NC, v1)),
  Qualified-expression (max(v2), has-value(TO, v2)))

### 5.4 Analysis of the case study

Using the developed software tool on the concept graph (as in Fig. 2) and the list of formulated qualified requirements, we discover some conflicts. The indicator TC (total costs) is caused by TK (total number of km), which on the other hand is correlated with R (revenue). In our requirements we have indicated that TC should be minimized (RQ1). It is also indicated that R should be maximized (RQ4). Due to the correlation, maximizing R will lead to maximizing TK. Due to the causal relationship, maximizing TK leads to maximizing TC, which disagrees with RQ4. This can be expressed in the following way:

RL1 $\wedge$ RL15 $\Rightarrow$ CONFLICTING (RQ1, RQ6)

Another conflict involving TC can be detected in the path TC $\rightarrow$ NT $\rightarrow$ TO. TC is positively correlated with NT which is positively correlated with TO (total number of orders). Therefore there is a conflict between RQ1 and RQ8:

RL4 $\wedge$ RL18 $\Rightarrow$ CONFLICTING (RQ1, RQ8)

Similarly conflicts exist between RQ1 and RQ11, RQ3 and RQ4, RQ3 and RQ8:

RL4 $\wedge$ RL18 $\wedge$ RL16 $\Rightarrow$ CONFLICTING (RQ1, RQ11)
RL7 $\wedge$ RL4 $\wedge$ RL18 $\Rightarrow$ CONFLICTING (RQ3, RQ4)
RL7 $\wedge$ RL4 $\wedge$ RL18 $\wedge$ RL10 $\Rightarrow$ CONFLICTING (RQ3, RQ8)

## 6 Conclusions

Organisational performance indicators are crucial concepts in strategic management of an organisation, and in particular in the preparation of organisational change processes. They can occur in a variety of forms and complexity. In addition, often it is necessary to consider relations between performance indicators, and to express qualifications and requirements over them. Given these considerations, it is not trivial to express them in a uniform way in a well-defined specification language.

A similar situation is addressed in the area of Requirements Engineering which has developed as a substantial subarea of Software Engineering. Also in the area of AI and Design similar issues are addressed. Inspired by these areas, a specification language for performance indicators and their relations and requirements has been defined and presented in this paper. The language can be used in different forms, varying from informal, semiformal, graphical to formal. The semantics of the language was left out from the scope of this paper and will be a subject of further research. A software environment has been developed that supports the specification process and can be used to automatically check whether performance indicators or relations between them or certain requirements over them (those with quantifier called satisfy) are satisfied in a given organisational process. For other types of requirements over performance indicators it may not be easy to automate the checking process. For example, that a certain performance indicator is minimal for a given organisational process requires comparison to alternative possible organisational processes. If a set of alternative processes is given, the software environment can handle the checking on minimality of one of these processes compared to the other ones. But in general such a set is hard to specify in an exhaustive manner. An alternative route is to make a mathematical analysis of this minimality criterion and to formalize this analysis in the language so that it can be performed automatically. Another direction for future investigation might be to provide assistance in the process of discovering missing or redundant requirements. The set of requirements is company-specific but it might be possible to provide some insight through scenario elicitation.

## References

1. Bosse T, Jonker CM, Treur J (2004) Analysis of design process dynamics. In: Lopez de Mantaras R, Saitta L (eds) Proceedings of the 16th European conference on artificial intelligence, ECAI'04, pp 293–297
2. Brazier FMT, van Langen PHG, Treur J (1996) A logical theory of design. In: Gero JS (ed) Advances in formal design methods for CAD, Proceedings of the second international workshop on formal methods in design. Chapman & Hall, New York, pp 243–266
3. Brewer PC, Speh TW (2000) Using the balanced scorecard to measure supply chain performance. J Bus Log 21(1):75–93
4. Chan FTS (2003) Performance measurement in a supply chain. Int J Adv Manuf Technol 21(7):534–548
5. Christopher M (1998) Logistics and supply chain management: Strategies for reducing cost and improving service, 2nd edn. Financial Times/Prentice-Hall, London
6. Davis AM (1993) Software requirements: Objects, functions, and states. Prentice Hall, New Jersey
7. Delfmann W, Albers S, Gehring M (2002) The impact of electronic commerce on logistics service providers. Int J Phys Distr Log Manag 32:203–222
8. Herlea Damian DE, Jonker CM, Treur J, Wijngaards NJE (2005) Integration of behavioural requirements specification within compositional knowledge engineering. Knowl-Based Syst J 18:353–365
9. Ittner CD, Larcker DF (2003) Coming up short on nonfinancial performance measurement. Harv Bus Rev 81(11):88–96
10. Kleijnen JPC, Smits MT (2003) Performance metrics in supply chain management. J Oper Res Soc 54(5):507–514
11. Kontonya G, Sommerville I (1998) Requirements engineering: processes and techniques. John Wiley & Sons, New York
12. Krauth E, Moonen H, Popova V, Schut M (2005) Performance measurement and control in logistics service providing. The Icfaian J Manag Res IV(7):7–19
13. van Langen PHG (2002) The anatomy of design: foundations, models and application. PhD thesis, Vrije Universiteit Amsterdam
14. Lemoine W, Dagnaes L (2003) Globalisation strategies and business organisation of a network of logistics service providers. Int J Phys Distr Log Manag 33(3):209–228
15. Neely A, Gregory M, Platts K (1995) Performance measurement system design: A literature review and research agenda. Int J Oper Prod Manag 15:80–116
16. Sommerville I, Sawyer P (1997) Requirements engineering: a good practice guide. John Wiley & Sons, Chicester, England
17. Sowa JF (1984) Conceptual structures: information processing in mind and machine. Addison-Wesley, Reading, Mass
18. Sowa JF, Dietz D (1999) Knowledge representation: logical, philosophical, and computational foundations, Brooks/Cole
19. Vaidyanathan G (January 2005) A framework for evaluating third-party logistics. Comm ACM 48(1):89–94