

A Specification Language for Organisational Performance Indicators

Viara Popova and Jan Treur

Department of Artificial Intelligence, Vrije Universiteit Amsterdam,
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
{popova, treur}@few.vu.nl

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.

1 Introduction

In organisational design, redesign or change processes, organisational performance indicators form a crucial source of information; cf. [6]. Within such processes an organisation is (re)designed to fulfill (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 fulfill 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 an area called requirements engineering; cf. [3][4][7]. Formal languages to express requirements have been developed, 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 language is introduced. In Section 2 it is shown how the proposed language can be used to express indicators themselves, but also how they relate to each other and in what

sense they are desirable. Next, in Section 3, case studies of the use of the language for the logistics domain are presented. Section 4 is a discussion.

2 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 [2][5][1]) 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.

2.1 Performance Indicators

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

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-AME	The set of possible names for lists of performance indicators

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 \times 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 2. An example set of performance indicators

Indicator name	Indicator	Indicator name	Indicator
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 explore the possible relationships between indicators.

2.2 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 below.

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

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 × INDICATOR-NAME × SIGN

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

IS-CAUSED-BY : INDICATOR-NAME × INDICATOR-NAME × 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 × INDICATOR-NAME × 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 × INDICATOR-NAME

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

INDEPENDENT : INDICATOR-NAME × INDICATOR-NAME

CONFLICTING : INDICATOR-NAME × INDICATOR-NAME

Examples: INDEPENDENT (ISC, FO), ¬ CONFLICTING (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 × 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 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 × INDICATOR-NAME

The relationships discussed so far can be represented graphically using a conceptual graph (see [8][9]). Conceptual graphs have two types of nodes: concepts and relations. In our case the first type will represent the indicator names 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.

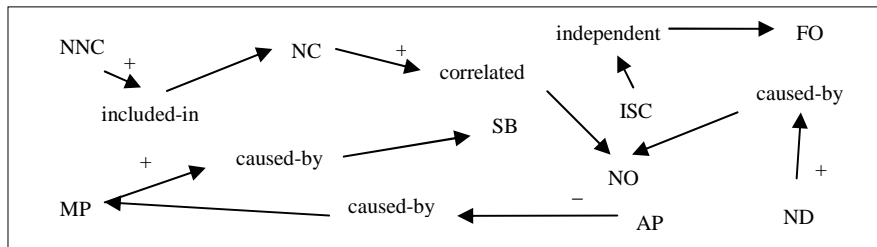


Fig. 1. The conceptual graph of relationships between the indicators

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.

2.3 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.

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

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),
 $v > \text{minKD} \ \& \ v \leq \text{maxKD}$ (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:

$\text{has-value: INDICATOR} \times \text{VARIABLE} \rightarrow \text{INDICATOR-VARIABLE-EXPRESSION}$

For example the expression $\text{has-value}(\text{NNC}, v)$ 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:

$\text{minimize, maximize: VARIABLE} \rightarrow \text{QUANTIFIER}$
 $\text{approximate: VARIABLE} \times \text{CONSTANT} \rightarrow \text{QUANTIFIER}$
 $\text{satisfy: VARIABLE-EXPRESSION} \rightarrow \text{QUANTIFIER}$

Examples: $\text{minimize}(v)$, $\text{approximate}(v, \text{bestKD})$, $\text{satisfy}(v < \text{maxKD})$

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

$\text{Qualified-expression: QUANTIFIER} \times \text{INDICATOR-VARIABLE-EXPRESSION} \rightarrow \text{QUALIFIED-EXPRESSION}$

As an example consider the expression $\text{Qualified-expression}(\text{min}(v), \text{has-value}(\text{ISC}, v))$, 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:

$\text{IS-DEFINED-AS: QUALIFIED-EXPRESSION-NAME} \times \text{QUALIFIED-EXPRESSION}$
 $\text{IS-DEFINED-AS: QUALIFIED-EXPRESSION-LIST-NAME} \times \text{QUALIFIED-EXPRESSION-LIST}$

Example: $\text{IS-DEFINED-AS}(q, \text{Qualified-expression}(\text{max}(v), \text{has-value}(\text{NNC}, v)))$

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. 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

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

Requirement: QUALIFICATION \times QUALIFIED-EXPRESSION-LIST \rightarrow 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).

We define further a number of predicates:

IS-DEFINED-AS : QUALIFIED-REQUIREMENT-NAME \times QUALIFIED-REQUIREMENT

IS-DEFINED-AS : QUALIFIED-REQUIREMENT-LIST-NAME \times QUALIFIED-REQUIREMENT-LIST

CONFLICTING : QUALIFIED-REQUIREMENT-NAME \times QUALIFIED-REQUIREMENT-NAME

Intuitively, CONFLICTING indicates that the two requirements cannot be satisfied together. More precisely that can happen 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 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).

We can now express rules such as this one – requirements over positively related indicators, where one is maximized and the other minimized, are conflicting:

```

 $\forall$  ( i1, i2 : INDICATOR-NAME; L : INDICATOR-LIST-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)  $\vee$  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))))
 $\Rightarrow$  CONFLICTING (r1, r2)

```

3 A Case Study from the Area of Logistics

In this section we take a case study from the area of 3rd-party logistics (3PL) and apply the approach presented in the previous section. 3PL companies are specialized in providing logistics services to other companies. Important performance aspects typically include efficiency in transportation (e.g. reduction of transportation costs, improvement of route planning, equipment and labour utilization, etc.), customer satisfaction, employees satisfaction (in order to reduce the attrition of drivers), etc. Our case study includes performance indicators relevant for most of these aspects.

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.

3.1 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 modeled in different ways as long as the possible values are ordered in a consistent way.

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

Indicator name	Indicator	Indicator name	Indicator
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)

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.

- RQ1: Requirement (desired, Qualified-expression (min(v), has-value(TC, v)))
- RQ2: Requirement (desired, Qualified-expression (max(v), has-value(SO, v)))
- RQ3: Requirement (desired, Qualified-expression (min(v), has-value(VO, v)))
- RQ4: Requirement (desired, Qualified-expression (max(v), has-value(ST, v)))
- RQ5: Requirement (desired, Qualified-expression (min(v), has-value(VP, v)))
- RQ6: Requirement (desired, Qualified-expression (max(v), has-value(R, v)))
- RQ7: Requirement (desired, Qualified-expression (max(v), has-value(TP, v)))
- RQ8: Requirement (desired, Qualified-expression (max(v), has-value(EM, v)))
- RQ9: Requirement (desired, Qualified-expression (max(v), has-value(EP, v)))
- RQ10: Requirement (desired, Qualified-expression (min(v), has-value(TS, v)))
- RQ11: Requirement (desired, Qualified-expression (max(v), has-value(TO, v)))
- RQ12: Requirement (desired, Qualified-expression (approximate(v, bestKD), has-value(KD,v)))
- RQ13: Requirement (desired, Qualified-expression (satisfy(v ≤ maxKD), has-value(KD,v)))
- RQ14: Requirement (desired, Qualified-expression (max(v), has-value(NC, v)))
- RQ15: Requirement (desired, Qualified-expression (max(v), has-value(S, v)))
- RQ16: Requirement (desired, Qualified-expression (min(v), has-value(NA, v)))
- RQ17: Requirement (preferred-over, Qualified-expression (min(v1), has-value(VO, v1)),
Qualified-expression (max(v2), has-value(SO, v2)))
- RQ18: Requirement (preferred-over, Qualified-expression (max(v1), has-value(NC, v1)),
Qualified-expression (max(v2), has-value(TO, v2)))

3.4 Analysis of the Case Study

Looking at figure 2 and the list of formulated qualified requirements, we detect some inconsistencies. 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 (RQ6). 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 RQ6. This can be expressed in the following way:

$$RL1 \ \& \ RL15 \Rightarrow \text{CONFLICTING} (RQ1, RQ6)$$

In a similar way we consider ST (shops per truck), TT (trips per truck), NT (total number of trips) and TC (total costs). ST is positively correlated with TT while NT is aggregation of TT. Therefore maximizing ST (as in RQ4) will lead to maximizing TT which results in maximizing NT. However NT is positively correlated with TC and RQ1 requires TC to be minimized.

$$RL5 \ \& \ RL23 \ \& \ RL4 \Rightarrow \text{CONFLICTING} (RQ1, RQ4)$$

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 RQ11:

$$RL4 \ \& \ RL18 \Rightarrow \text{CONFLICTING} (RQ1, RQ11)$$

The last conflict we detect arises from RQ8 and RQ10. RQ8 requires EM (employee motivation) to be maximized. EM is positively caused by TS, therefore changing TS will change EM in the same direction. RQ10 requires TS to be minimized which will lead to minimizing EM – conflict with RQ8.

$$RL14 \Rightarrow \text{CONFLICTING} (RQ8, RQ10)$$

4 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 sub-area 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 satisfy) are satisfied in a given organisational process. The relevant complexity issues of the checking process are still a topic for future research.

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. This is a subject for further research. 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, C.M., and Treur, J.: Analysis of Design Process Dynamics. In: Lopez de Mantaras, R., Saitta, L. (eds.): Proc. of the 16th European Conference on Artificial Intelligence, ECAI'04 (2004) 293—297
2. Brazier F.M.T., Langen P.H.G. van, Treur J.: A logical theory of design. In: Gero, J.S. (ed.): Advances in Formal Design Methods for CAD, Proc. of the Second International Workshop on Formal Methods in Design. Chapman & Hall, New York (1996) 243—266
3. Davis, A. M.: Software requirements: Objects, Functions, and States, Prentice Hall, New Jersey (1993)
4. Kontonya, G., and Sommerville, I.: Requirements Engineering: Processes and Techniques. John Wiley & Sons, New York (1998)
5. Langen, P.H.G. van: The anatomy of design: foundations, models and application. PhD thesis, Vrije Universiteit Amsterdam (2002)

6. Neely, A., Gregory, M., Platts, K.: Performance measurement system design: A literature review and research agenda. *International Journal of Operations & Production Management*, 15 (1995) 80—116
7. Sommerville, I., and Sawyer P.: *Requirements Engineering: a good practice guide*. John Wiley & Sons, Chicester, England (1997)
8. Sowa, J.F.: *Conceptual Structures: Information Processing in Mind and Machine*, Addison-Wesley, Reading, Mass. (1984)
9. Sowa, J.F., Dietz, D.: *Knowledge Representation: Logical, Philosophical, and Computational Foundations*, Brooks/Cole (1999)