A Logic-Based Formalization of KPIs for Virtual Enterprises*

Claudia Diamantini, Domenico Potena, and Emanuele Storti

Dipartimento di Ingegneria dell'Informazione,
Università Politecnica delle Marche - Via Brecce Bianche, 60131 Ancona, Italy
{c.diamantini,d.potena,e.storti}@univpm.it

Abstract. Open innovation is gaining increasing interest as a model to foster innovation through collaboration and knowledge sharing among organizations, especially in the context of Virtual Enterprises (VE). One of the main issues to overcome in such distributed settings is the integration of heterogeneous data, and the need to evaluate common Key Performance Indicators (KPI) capable to measure overall performances of the VE. In this paper we propose a conceptualization of KPIs into an ontology, to provide a common vocabulary to semantically annotate data belonging to different organizations. KPIs are described in terms of dimensions and a mathematical formula. In order to support reasoning services over KPIs formulas we refer to a logic-based formalization in Prolog, where formulas are translated as facts, and several predicates are included to support both mathematical functionalities for formula manipulation and higher-level functions especially suited for VE setup.

Keywords: KPI ontology, logic-based formalization of KPIs, logic reasoning, open innovation, virtual enterprises.

1 Introduction

Collaboration, communication and interaction are key elements to open paradigms to innovation and especially to Virtual Enterprises (VE), which are characterized by heterogeneous groups of partners with different policies, skills, know-how and way of doing business. In this open context, the BIVEE project[1] is purposed to develop an ICT framework capable to address the need of knowledge integration and sharing for innovation projects. In BIVEE, to overcome interoperability issues, a semantics-based infrastructure, namely the Production and Innovation Knowledge Repository (PIKR) [1], serves as a knowledge gateway. By enabling a unified access to information and by exploiting a set of reasoning functionalities, it is capable to grant a smart and easier access to resources and ultimately to enhance support for decision making.

* This work has been partly funded by the European Commission through the ICT Project BIVEE: Business Innovation and Virtual Enterprise Environment (No. FoF-ICT-2011.7.3-285746).

[1] http://www.bivee.eu

X. Franch and P. Soffer (Eds.): CAiSE 2013 Workshops, LNBIP 148, pp. 274–285, 2013.

In order to compare performances of the participating organizations, to measure overall performances of the VE and ultimately to enable managers to define information requirements and goals, it becomes crucial to evaluate heterogeneous Key Performance Indicators (KPI) about innovation and production. This problem can be considered a particular instance of the more general integration of federated Data Warehouses (DWH), in which each organization provides a DWH with its own dimensional structure and fact table, and KPIs are the measures for this last. However, besides the typologies of heterogeneity usually addressed in the DWH Literature (i.e., dimensional heterogeneities, see [2] for a survey), in distributed and collaborative environment a further source of heterogeneity is given by the semantics that each organization assumes for the indicators, in terms of both conceptual meaning and mathematical structure of the corresponding formula [3]. For instance, let us consider two collaborating enterprises A and B, providing data about production costs measured through different indicators, namely I_x (for A) and I_y, I_z (for B). Given that there are not indicators in common, integration at VE level would not be feasible. However, if the structure of I_x formula is know and equal to a combination of the other two, e.g. I_y+I_z, then it is possible to compute I_x also for B and compare the two results.

In the context of the PIKR, this work is aimed to present the semantic model for the definition and manipulation of heterogeneous KPIs belonging to collaborating organizations that take part to a VE. The approach is based on the conceptualization of KPIs, including their properties and the structure of their formulas, through an ontological representation that serves as a common reference language for the VE, allowing the annotation of enterprise data by means of a shared terminology. Then, in order to support reasoning functionalities over KPIs, we refer to a logic-based formalization in predicate logic of KPI formulas and mathematical functions implementing axioms, together with operators for equation solving and formula manipulation. By following this approach, indicators referring to the same concept but having a different name or mathematical structure can be reconciled, and new indicators can be defined in the VE, if implicitly computable starting from the provided data. The reasoning services introduced herein are targeted to support the VE setup phase, and include KPI elicitation, the analysis of dependencies among indicators in terms of common components, evaluation of semantical equivalence between formulas, check of consistency and correctness. The management of annotation of enterprise data, as well as the definition of links between business concepts and KPIs are outside the purpose of this work, as they are addressed by specific modules within the BIVEE framework.

The rest of this work is organized as follows: next Section proposes a brief overview of the main solutions and models available in the Literature for KPI modeling and management. In Section 3 the conceptualization of KPIs into an ontology is provided, while Section 4 is devoted to present a logic-based formalization of KPI formulas in predicate logic, together with the reasoning services. A case study in an example scenario is introduced in Section 5, while Section 6 provides some final remarks and discusses future extensions.

2 Related Work

Only few work in the Literature have focused to models and methodologies for monitoring efficiency and effectiveness of innovation activities and for innovation project planning, especially in cooperative environments. Many studies propose innovation indicators at macro-level, suitable to analyse innovation performances of countries or market segments (see e.g., [4]) but largely unapt to monitor and improve internal processes and organization structures. In [5] a review of the Literature pertaining to the measurement of innovation management at the firm's level is presented, reporting also "an absence of measures well aligned to the activities of the innovation process". A framework and a set of 26 KPIs for the measurement of innovation availability, efficiency and effectiveness has been proposed in [6]. In such a work, innovation in a firm is seen as a whole, ignoring the process that created it. A framework to semantically define Performance Indicators related to processes is introduced in [7]. However, our focus is not limited to KPIs about processes, as we address also indicators about other resources and perform reasoning on them. To the best of our knowledge, there is a lack of work about reconciliation of heterogeneous KPIs used by different enterprises in a collaborative setting, which requires a peculiar care in the management of KPI integration.

For what concerns a systematization of KPI for enterprise environments, there is a plethora of Performance Indicators definitions and/or Glossaries provided by researchers and research groups, international and national public bodies (e.g., [4]), and in reference models like the Supply-Chain Reference Model (SCOR) [8], the Value Chain Reference Model (VRM) [9] and Six-Sigma [10] (see also [1] for a more comprehensive list of performance measurement approaches and KPI description). All such reference models provide quite diverse indicators, depending on the goal and the domain of interest. However, in order to design the ontology described in Section 3, namely KPIOnto, we need to abstract over their general characteristics so that any indicator can be represented. In this sense, almost all proposals are homogeneous in the kinds of information provided, differing on the degree of precision and formality of the description. For the design of the ontology we focus on the description of indicators in VRM. This choice has three main reasons: (1) the VRM model explicitly addresses networked enterprises, (2) the model provides an accurate description of indicators, and (3) the BIVEE project mainly refers to VRM model to develop its business innovation reference framework [1].

A first proposal of the ontology and the approach described in this work is available in [3], in which a semantic representation of KPIs for health care sector is presented, together with a set of reasoning functionalities aimed at requirements elicitation and DWH design, in particular for data marts materialization, consolidation and integration.

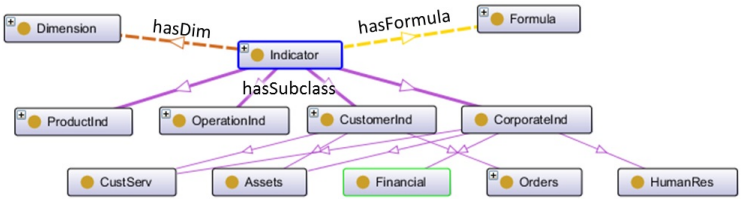

Fig. 1. KPIONTO: main classes

3 KPIOnto: An Ontology of Key Performance Indicators

Following the VRM model, in KPIOnto a taxonomy of KPIs is introduced, based on the Enterprise area of intervention, e.g., Corporate, Customer, Operation or Product. However, multiple-categorization is allowed by the ontology, thus supporting more efficient indexing and searching functionalities. The main categories of the KPIOnto that are relevant in the context of this work are Indicator, Dimension and Formula. Figure 1 shows these classes and relationships among them. The *Indicator* is the pivotal class of the KPIOnto, and its instances (i.e., indicators) describe the metrics enabling performance monitoring. Properties of an indicator are: name, identifier, acronym, definition (i.e., a plain text giving a detailed description of the indicator meaning and its use) and Unit of measurement (i.e., both the symbol and the description of the unit of measurement chosen for the indicator. These properties are given by referring to the Measurement Units Ontology (MUO) [11]).

A dimension is the coordinate/perspective to which the metric refers. For instance, it is useful to analyse deliveries along dimensions like the delivery date, the delivered product, the means of transportation, and so forth. Following the multidimensional model, a dimension is usually structured into a hierarchy of levels, where each level represents a different way of grouping elements of the dimension [12]. For instance, it can be useful to group means of transportation by transport companies, and days by weeks and years. Each level is instantiated in a set of elements known as members of the level, e.g. the company "ACME", the weeks "3rd-2012" and "42nd-2011". In order to describe dimensions in KPIOnto, our approach is similar to the one proposed in [13]. In particular, the top class is *Dimension*, which is composed of a set of subclasses, one for each specific dimension, like Organization Dimension, Time Dimension, and so forth. Levels are represented as disjoint primitive subclasses of the dimension (or the set of dimensions) they belong to, each member is an instance of these classes, and the dimension's hierarchy is represented by a part-of relationship among classes. Hence, for instance, the Organization dimension has a Person level that is part-of a Team or of a Department, which in turn are part-of an Enterprise, and so forth. This ontological description enables usual OLAP operations over indicators, namely roll-up and drill-down. The roll-up operation has a direct correspondence with the part-of relationship, while the drill-down operation implies an opposite path to that described by the part-of.

Finally, the semantics of an indicator cannot be fully given without the representation of its *Formula*, which is the way the indicator is computed [3]. Each formula is characterized by: the related indicator, the aggregation function, the way the formula is presented, the semantics (i.e., the mathematical meaning) of the formula, and references to its components, which are in turn (formulas of) indicators. The formal representation and manipulation of the structure of a formula is essential in Virtual Enterprises to check inconsistencies among independent indicator definitions, reconcile indicators values coming from different sources, and provide the necessary flexibility to indicators management. To this end we need a new kind of ontology, whose peculiarity is the contemporaneous use of logic axioms as well as algebraic formulas to represent the information about indicators. We resort to mathematical standards for describing the way the formula is presented and its semantic (MathML [14]), and for representing operators semantics (OpenMath [15]). Interested readers can refer to [16] for more details about the use of these standards.

4 Reasoning Functionalities

Reasoning about KPIs is mainly based on the capability to manipulate formulas according to strict mathematical axioms, like commutativity, associativity and distributivity of binary operators, and properties of equality needed to solve equations. Therefore, in order to define KPI reasoning functionalities we need to represent both KPIOnto formulas and mathematical axioms in a common logic-based layer. To this end, we refer to the first-order logic and define the functionalities in logic programming (LP); in particular, we use Prolog and we refer to XSB, a logic programming and very efficient deductive database system[2], as reasoning engine. KPIOnto formulas are translated from MathML-Content to LP facts, while we have adapted and extended the Equation Solving System (PRESS) [17] for the representation of mathematical axioms in LP predicates. Indicators' formulas are translated by using two predicates, as follows:

- `formula(Y,Expression,K)` is a LP fact representing the formula related to an instance of the Indicator class in the KPIOnto. Y is the indicator name, Y=Expression is the formula of the indicator written using the infix notation, and K is a constant identifying whether Y is a leaf indicator or not. In the KPIOnto a leaf indicator is an indicator whose formula is not defined on the basis of other indicators.
- `equation(Equation,X)` represents a generic equation in the variable X. When a new indicator has to be added to the knowledge base, at first it is inserted using the equation predicate, and in this form it goes through correctness and consistency checks, as explained below. If the check is positive a new `formula` fact is added to the LP theory, and hence to the ontology.

In the following subsections we introduce the main functionalities for reasoning about KPIs, whereas in Section 5 some usage examples are provided.

[2] `http://xsb.sourceforge.net/`

4.1 Formula Manipulation

Formula Manipulation is an essential reasoning functionality already available in PRESS, that consists in walking through the graph of formulas to derive relations among indicators and rewriting a formula. All the other reasoning functionalities are based on Formula Manipulation. We consider two main types of predicates: for the simplification and for the resolution of equations. The former are used both to compact and to improve the rendering of a formula, by individuating and collecting equivalent terms (e.g., $a * b$ and $b * a$, or $a * a$ and a^2, or $a * 1/b$ and a/b), and by deleting unnecessary brackets (e.g., [()]=()). The main predicates of this kind are `simplify_term`, which is used to simplify equations, and `simplify_solution` that rewrites the final solution in a more understandable form.

The second type of predicates enables the symbolic resolution of equations by applying mathematical properties (e.g., commutativity, factorization), and properties of equality. The number and kind of manipulations the reasoner is able to perform depend on the mathematical axioms we describe by means of logical predicates. The resolution of the equation *Equation* in a given variable X starts from the predicate `solve_equation`, that is defined as follows:

```
solve_equation(Equation,X,X=Solution) :-
    solve_equation_1(Equation,X,X=SolutionNotSimplified),
    simplify_solution(SolutionNotSimplified,Solution).
```

where *Solution* is the solution of the equation.

The predicate solve_equation_1 handles different kinds of equations, e.g. linear and polynomial ones[3]. For lack of space, here we report only the method for solving linear equations, as follows:

```
solve_equation_1(LeftMember=RightMember,X,Solution) :-
    simplify_term(LeftMember-RightMember,LeftMember1),
    single_occurrence(X,LeftMember1=0),!,
    position(X,LeftMember1=0,[Side|Position]),
    isolate(Position,LeftMember1=0,X=Solution).
```

The resolution of linear equations is based on the isolation method, consisting in manipulating the equation and trying to isolate the variable X on the left side of the equation; the right side is the requested solution. To this end the equation is firstly simplified by means of the `simplify_term` predicate, then the `single_occurrence` predicate is verified to check whether the equation is linear or not. This predicate `single_occurrence(X,LeftMember1=0)` is true if the variable X is only in one term of *LeftMember1*. Note that after the simplification, terms of the same degree are grouped together, so if the equation is linear the term of first degree of X is present only once. The other predicates are verified to actually implement the isolation method. The `position` predicate returns the side and list of positions of the variable X in the equation. For instance, the side

[3] At the present stage of the project, the formulas of all KPIs described in the KPIOnto are linear and polynomial equations.

and position of x in the equation 3*(1/x)-5*y=0 are respectively "1" (left side) and "1,2,2", i.e. the first position with respect to the minus operator, the second argument of the product, and finally the second argument of the division. At this point, predicates for isolation are used:

```
isolate([N|Position],Equation,IsolatedEquation) :-
    isolax(N,Equation,Equation1),
    isolate(Position,Equation1,IsolatedEquation).
isolate([],Equation,Equation).
```

The set of predicates `isolax` are needed to move a term from a side to another, for instance multiplying or adding the same term to both sides. In total, in order to perform the formula manipulation functionalities more than 900 predicates are used in the theory.

4.2 Equivalence Checking

During KPI elicitation and ontology population, it is useful to individuate and to manage duplications. In our scenario, two KPIs are duplicated if their formulas are equivalent. If duplicates are found, one can choose to erase them leaving only one definition for each KPI in the ontology or to allow multiple definitions by introducing a sort of *same_as* property among KPIs formulas. The second choice is useful to explicitly represent alternatives, to link a formula to the enterprise that actually uses it, and to reduce the time to make inference.

Given two formulas F and G, they are equivalent if F can be rewritten as G, and vice versa, by exploiting Formula Manipulation functionality. The equivalence check is implemented by means of the following predicate:

```
equivalence(Equation,X,Y) :-
    expand_equation(Equation,ExpandedEquation),
    solve_equation(ExpandedEquation,X,X=Solution),
    formula(Y,S,_), expand_equation(Y=S,Y=ES),
    solve_equation(Y=ES,Y,Y=Solution2), Solution=Solution2.
```

where Y is the formula in the ontology that is equivalent to the *Equation* in the variable X.

In order to avoid ambiguity, the check of equivalence is made at leaves level, where indicators are defined without referring to other indicators. Hence, an equation at this level can not be expanded further, and there is no other way to write it. To this end, the `expand_equation` predicate recursively replaces each term in *Equation* with its formula, until we obtain an *ExpandedEquation* that is formed only by leaves indicators. After this transformation the variable X could also be at the right side, then the *solve_equation* is called to rewrite the equation. The same is done for each formula in KPIOnto. If after these rewritings a formula Y exists such that it is equal to the equation, then the `equivalence` predicate is satisfied and the equivalent formula is returned.

4.3 Consistency Checking

When new indicators are added to the ontology or existing indicators are updated or deleted, it is needed to check that such changes do not contradict the ontology. To this end *Consistency Checking* is introduced, that is based on the idea that equivalence relations must be preserved both between indicators and formulas. As a consequence, a formula F_1 for a new indicator I_1 is consistent with the ontology if, whenever there exists a formula F for an indicator I already defined in the KPIOnto such that F_1 can be rewritten as F, then the equivalence $I \equiv I_1$ does not contradict the ontology. Otherwise we have an inconsistency.

In the Prolog theory, we introduced the `incoherence` predicate as follows:

```
incoherence(Equation,X,N) :-
    expand_equation(Equation,ExpandedEquation),
    solve_equation(ExpandedEquation,X,X=Solution),
    formula(Y,S,_,N), expand_equation(Y=S,Y=ES),
    solve_equation(Y=ES,X,X=Solution2), Solution \=Solution2.
incoherence(Equation,X,incosistent) :-
    expand_equation(Equation,ExpandedEquation),
    solve_equation(ExpandedEquation,X,X=0).
incoherence(Equation,X,N) :-
    \+expand_equation(Equation,ExpandedEquation),
    formula(X,_,branch_node,N),!.
```

Given an equation (X=*Equation*), the predicate returns whether the equation is consistent with the ontology and, if any, the reason of the inconsistency. The `incoherence` predicate is based on the definition of equivalence, as described above. Given the set of formulas in the ontology that can be expanded and rewritten as functions of X, each element that is different from the given *Equation* (after the appropriate expansion) leads to inconsistency and is returned. We have an inconsistency also when, after a manipulation, the new equation assumes the form X=*0*, or when the new equation cannot be expanded but a non-leaf formula for X exists. In other cases the consistency is verified.

4.4 Extraction of Common Indicators

The last reasoning functionality we present in this work is the extraction of common indicators. Given a set of indicators $\phi = \{I_1, I_2, ..., I_n\}$, common indicators of ϕ (denoted by ci(ϕ)) is the minimal set of atomic indicators needed to compute all formulas of ϕ. A more general notion refers to the extraction of the k-Nearest Neighbours of a set of indicators ϕ (denoted by NN(ϕ,k)), that is, the set of indicators at a distance k from all indicators in ϕ, where the distance is the minimal number of dependency arcs separating two indicators. Note that ci(ϕ)=NN(ϕ,$+\infty$). Common indicators and k-Nearest Neighbours are useful to determine the possible sets of information that have to be provided by enterprises in order to calculate the requested KPI or, in a bottom-up fashion, to recognize what kind of indicators can be derived, given the available information

at enterprise level. Hence, they are essential tools at design and measuring time. On the basis of these definitions, we have implemented the `meaToInd` predicate which, given a set of indicators (called measures), returns the derivable KPIs and `indToMea` predicate that, given a set of KPIs, returns the sets of measures needed to compute all the KPIs. These two predicates extend the definition of k-Nearest Neighbours, by returning the set ϕ for any values of the distance k.

```
indToMea(Ent,L,Lfin):-
    hasEntInd(Ent,[L],[],Lfin2), sort(Lfin2,Lfin).
```

where *Ent* is the name of the enterprise, *L* is the set of KPIs and *Lfin* the set of needed measures.

The predicate is satisfied for each set *Lfin* of indicators provided by the enterprise, such that a formula for *L* can be derived from the ontology and it is defined on the basis of this set. `hasEntInd` checks, by using Formula Manipulation predicates, that an indicator is provided by the enterprise.

```
meaToInd(Ent,Lin,V):-
    hasMeaEnt(Ent,Lin), formula(V,F,Kind,Fname),
    \+member(V,Lin), indToMea(Ent,V,L), subset(L,Lin).
```

where *Lin* is the set of measures and *V* the set of indicators derivable by *Lin*.

As first step, the predicate `hasMeaEnt` is called to check whether *Ent* provides all indicators in *Lin*. Now, by using `indtoMea`, the predicate returns each indicator *V* that can be computed by a subset of *Lin*. If *Ent* is not specified, then the predicate is checked for all the enterprises.

5 Case Study

Let us consider two partners, ACME and ACME2, willing to build up a Virtual Enterprise. They agree on the KPIs to be used to monitor the VE, defined by the following formulas, whose structural relations are depicted in Figure 2:
- $PersonnelCosts = NumHours * HourlyCost * (Overhead + 1)$
- $Costs = TravelCosts + PersonnelCosts$
- $PersonnelTrainingCosts = HourlyCost * NumTrainingHours$
- $TeachCost = NumTrainingHours * HourRate$
- $InvestmentInEmployeeDevelopment = PersonnelTrainingCosts + TeachCost$

At one point, ACME realizes that would need other KPIs for monitoring the VE and proposes to introduce two new indicators (1) *IG_ROI* and (2) *TotCostsEmpTrain*, currently not defined, which are aimed to measure respectively the Return On Investment related to those ideas generated by the organization and the total internal costs invested for training of employers:

$ROI\_IG = ExpectedMarketImpact/Costs$

$TotCostsEmpTrain = TeachCost + NumTrainingHours * HourlyCost$

After having translated such KPI formulas in Prolog facts, the Equivalence and Consistency Checking functionalities are used to verify whether formulas

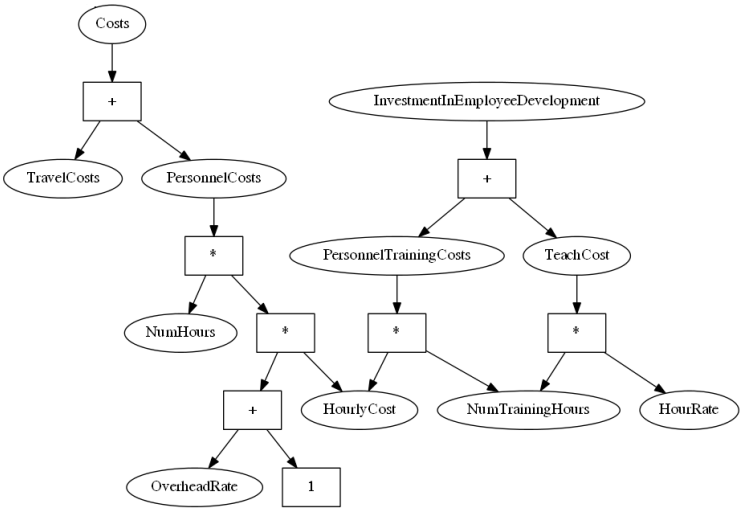

Fig. 2. Formula tree for the example

are duplications of existing ones and whether they do not make the ontology inconsistent. Given that the *ROI_IG* formula is new and does not contradict any previously defined formula, it is added to the Prolog theory. On the contrary, by using the formula about *PersonnelTrainingCosts* and after some formula manipulation, the reasoner discovers that *TotCostsEmpTrain* is structurally equivalent to *InvestmentInEmployeeDevelopment*. Hence, ACME can refer to this last one to annotate its data without introducing a new indicator.

Consistency Checking is exploited also when an enterprise tries to update the definition of a KPI. For instance, let us consider that ACME requires to update *NumTrainingHours*, committing a manifest error when typing the formula:

$$NumTrainingHours = PersonnelTrainingCosts + HourlyCost$$

In this case, ACME discovers that the change directly contradicts the definition of *PersonnelTrainingCost* together with every other higher-level indicators depending on *PersonnelTrainingCost*, i.e. in this case *InvestmentInEmployeeDevelopment*.

By using the `meaToInd` predicate, we can introduce another way to support the choice of the KPIs to use at VE level. In particular, starting from those indicators that each enterprise chooses to share in the VE, we are able to discover the common set of KPIs the VE is able to globally compute. In the present case study, we assume that ACME and ACME2 respectively share in the VE data about the following set of indicators:

ACME={*ROI_IG*, *InvestmentInEmployeeDevelopment*, *HourlyCost*, *NumHours*, *OverheadRate*}

ACME2={*HourRate*, *NumHours*, *NumTrainingHours*, *OverheadRate*, *PersonnelTrainingCosts*, *ExpectedMarketImpact*, *TravelCosts*}

The Extraction of Common Indicator functionality returns the set of all the computable KPIs for the VE: {*OverheadRate*, *NumHours*, *HourlyCost*, *Person-*

nelCosts, *ROI_IG*}. We'd like to note that the last three of such indicators are directly computable only by ACME. However, ACME2 provides *NumTrainingHours* and *PersonnelTrainingCosts*, from which the reasoner is able to derive *HourlyCosts* (by inverting the formula of *PersonnelTrainingCosts*), and then *PersonnelCosts* and also *ROI_IG*.

In order to actually compute KPIs values during the monitoring of the VE, and also in case of the introduction of a new indicator, not previously defined, the functionality provided by the *indToMea* predicate can be exploited. It is used to verify whether it is possible to evaluate the required new indicator starting from the indicators provided by the enterprises. As a by-product, it retrieves the list of indicators that, for each enterprise, are needed for actually computing the requested KPIs. For instance, if the value of *ROI_IG* is required, the reasoner reports as output the following set of indicators:

ACME={*ROI_IG*}

ACME2={*NumHours*, *NumTrainingHours*, *OverheadRate*, *PersonnelTrainingCosts*, *ExpectedMarketImpact*, *TravelCosts*}

If a KPI can not be computed from the provided indicators, then the reasoner is able to return the minimal list of additional indicators to be provided.

6 Conclusions

In this work we introduced a semantic model for the definition of KPIs from both a conceptual and a structural perspective. The KPIOnto ontology is exploited in the BIVEE project as common conceptual reference for annotation of KPIs about real enterprise data, whereas a logic-based formalization in Prolog of the structure of KPI formulas is useful to support various reasoning functionalities. Although those introduced in this work are especially suited for the setup of a VE, others have been described in a previous work for execution of OLAP-like and complex queries [16]. The proposed functionalities are currently exposed through web service interfaces, together with other reasoning services of the PIKR, i.e. the semantic layer of the BIVEE project. In this way those BIVEE modules appointed to offer high-level functions and graphical user interfaces to business managers can use the services in a loosely-coupled fashion. Here we focused only on linear equations and common aggregation functions (i.e., minimum, maximum, average, sum or count), because the more than 200 KPIs provided by the enterprises within the BIVEE project do not require more complex representations. However, more predicates are already available to deal with generic non-linear equations, that can be added to the theory thus increasing both expressiveness and computational complexity. In order to keep under control this last, which is often critic in logic programming, we are both analytically and empirically evaluating the theory, in order to locate bottlenecks, useful to implement optimizations strategies by refactoring LP predicates. As future work we also plan to conduct tests on data coming from real use cases, to fully integrate both already existing and new functionalities with all the other modules of the project, and to extensively populate the ontology.

References

1. Diamantini, C., Potena, D., Proietti, M., Smith, F., Storti, E., Taglino, F.: A semantic framework for knowledge management in virtual innovation factories. Int. Journal of Information System Modeling and Design (IJISMD) (in press)
2. Tseng, F.S., Chen, C.W.: Integrating heterogeneous data warehouses using xml technologies. J. Inf. Sci. 31(3), 209–229 (2005)
3. Diamantini, C., Potena, D.: Thinking structurally helps business intelligence design. In: D'Atri, A., Ferrara, M., George, J.F., Spagnoletti, P. (eds.) Information Technology and Innovation Trends in Organizations, pp. 109–116. Physica-Verlag HD (2011)
4. OECD: Glossary for oecd composite leading indicators, `http://www.oecd.org/document/1/0,3746,en_2649_34349_32694145_1_1_1_1,00.html`
5. Adams, R., Bessant, J., Phelps, R.: Innovation management measurement: A review. International Journal of Management Reviews 8(1), 21–47 (2006)
6. Zheng, H., Chanaron, J.J., You, J., Chen, X.: Designing a key performance indicator system for technological innovation audit at firm's level: A framework and an empirical study. In: IEEE International Conference on Industrial Engineering and Engineering Management, IEEM 2009, pp. 1–5 (December 2009)
7. del-Río-Ortega, A., Resinas, M., Ruiz-Cortés, A.: Defining process performance indicators: An ontological approach. In: Meersman, R., Dillon, T.S., Herrero, P. (eds.) OTM 2010. LNCS, vol. 6426, pp. 555–572. Springer, Heidelberg (2010)
8. Supply-Chain Council: Supply chain operation reference (scor) model, overview-version 10.0., `http://supply-chain.org/`
9. Value-Chain Group: Value reference model (vrm), `http://www.value-chain.org/en/cms/1960`
10. Tennant, G.: Six Sigma: SPC and TQM in manufacturing and services. Gower Publishing, Ltd. (2001)
11. Measurement Units Ontology: `http://forge.morfeoproject.org/wiki_en/index.php/units_of_measurement_ontology`
12. Kimball, R., Ross, M.: The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling, 2nd edn. John Wiley & Sons, Inc., New York (2002)
13. Neumayr, B., Schrefl, M.: Multi-level conceptual modeling and OWL. In: Heuser, C.A., Pernul, G. (eds.) ER 2009 Workshops. LNCS, vol. 5833, pp. 189–199. Springer, Heidelberg (2009)
14. W3C Math Working Group: Mathematical Markup Language (MathML) Version 2.0., 2nd edn., `http://www.w3.org/Math/`
15. OpenMath Society: The OpenMath Standard Ver. 2.0., `http://www.openmath.org/`
16. Diamantini, C., Potena, D.: Semantic enrichment of strategic datacubes. In: Proc. of the ACM 11th Int. Workshop on Data Warehousing and OLAP, pp. 81–88. ACM (2008)
17. Sterling, L., Bundy, A., Byrd, L., O'Keefe, R., Silver, B.: Solving symbolic equations with press. J. Symb. Comput. 7(1), 71–84 (1989)