

RESEARCH ARTICLE - METHODOLOGY

A Flexible Framework to Ensure Traceability, Consistency, and Propagation of KPIs Evolution

Eladio Domínguez¹  | Beatriz Pérez²  | Ángel L. Rubio²  | María A. Zapata¹ 

¹Departamento de Informática e Ingeniería de Sistemas, Universidad de Zaragoza, Zaragoza, Spain | ²Departamento de Matemáticas y Computación, Universidad de La Rioja, La Rioja, Spain

Correspondence Beatriz Pérez | (beatriz.perez@unirioja.es)

Received: 29 August 2022 | **Revised:** 11 December 2024 | **Accepted:** 29 January 2025

Keywords: consistency conditions | constraint logic programming | evolution framework | KPIs | propagation | reasoning | traceability

ABSTRACT

Organizations use key performance indicators (KPIs) to assess the effectiveness and efficiency of their procedures and processes. In a world that is constantly evolving and hyperconnected via the internet, it is of great interest to analyze how changes (organizational, legal, technological or other) can lead to modifications in the KPIs involved. However, little attention has been paid to KPI evolution either in the scientific literature or in developed solutions. This paper presents *A Flexible Framework for the Evolution, Consistency and Traceability of KPIs* (AFFECTK) that aims at establishing the basis for suitable KPIs' evolution management. The feasibility of this proposal is demonstrated through a proof-of-concept developed using a reasoning tool based on Constraint Logic Programming. The framework is further evaluated, using real KPI case studies, to assess the functional suitability of our approach.

1 | Introduction

Decision-making is a fundamental constituent of successful management in companies and institutions of all kinds [1]. In order to achieve the objectives of the organization, its managers must make accurate and up-to-date decisions. Among the many factors that hinder this decision-making, we can highlight two that are strongly related to the development of internet-related technologies: the enormous amount of information available on the web and the existence of an ever-changing internal and external context (new regulations, new technological developments, etc.) [2]. To assist in decision-making, organizations use all means at their disposal. Among these means, one of the most important is performance assessment [1], since it can be applied both to individuals and aggregations (departments, working groups, etc.) and to the own processes of the organization. The most common way of assessing performance is through the setting of indicators, usually in the form of key performance indicators or KPIs that “represent a set of measures focusing on those aspects of organizational performance that are the most critical

for the current and future success of the organization” [3] and are “quantitative or qualitative indicators that reflects the state/progress of the company, unit or individual” [4].

Despite being an instrument used in a global way (or precisely because of this), the paradox is that there is no standard model for the specification and use of KPIs. In [5], this problem is addressed by means of an exhaustive review of the literature, obtaining as a result a *taxonomy for KPIs management*, which collects, in a unified way, the diverse properties applied to the notion of KPI in the different approaches used in the last decades. One of the findings of this taxonomy is that adaptation and evolution of KPIs have been studied very poorly and that there is a need for considerable research on the evolution of performance measurement systems, conclusions reinforced by recent works [6]. This is somewhat surprising, since, as we have indicated, one of the most important characteristics of the current socio-economic context is the fact that it is subject to constant change. As circumstances change, organizations must be prepared to adapt accordingly. Thus, since KPIs are

used to evaluate performance, they should also be naturally adapted to the changes that occur over time. These adaptations must meet at least two requirements. KPIs must continue to fulfill their performance assessment function in the least invasive and most automated way possible, without changes leading to disruption of procedures. Additionally, they must be useful for tracing the changes that have occurred, as the form that these changes take can be precisely a key factor in decision-making. As such, adapting KPIs to changes needs to be managed and, if possible, assisted by automated tools.

Based on these premises, and making use of the terminology presented in [7], in this context we could speak of *KPI flexibility* to refer to any aspect related to modifications that can be considered in performance indicators. Similarly based on [7], we distinguish four types of KPI flexibility. Thus, we can speak of *KPI variability* (implies the existence of different KPI variants); *KPI looseness* (KPI are not fully prespecified); *KPI adaptation* (KPI are adapted to cope with emerging events); and *KPI evolution* (the capacity of modifying a KPI permanently to accommodate evolving needs). In view of the scarce number of approaches that consider KPI evolution aspects [5] and encouraged by the undoubted interest in providing organizations with advanced mechanisms for managing KPIs flexibility, in this paper we focus on KPI evolution.

More specifically, we propose as a long-term research goal to build on a *KPI evolution framework* which can be used with whatever particular KPIs management approach. A first version of some of the elements that shape this framework was presented in a previous work [8]. Now, we make a more complete proposal, called *A Flexible Framework for Evolution, Consistency and Traceability of KPI (AFFECTK)*, which we present following the Design Science Research (DSR) methodology [9] (Section 3). This proposal introduces new relevant features with respect to our previous proposal [8], among which we indicate the most noteworthy. First, we present the framework through a three-level (*method*, *operational*, and *application*) working structure. Second, in order to achieve adequate management of KPIs evolution, we enrich our previous proposal based on the fulfillment of three properties: *traceability*, *consistency*, and *propagation*. We have included these three properties as cornerstones for our framework since these aspects have been (separately) considered in the literature as key elements of the evolution of indicators [4, 6, 10–12]. These properties are the guiding thread of two essential components of *AFFECTK*: pattern and metamodel. More specifically, we propose a notion of *KPI evolution pattern* to capture, in a structured and comprehensive way, all the factors that affect the evolution of a KPI (from the most basic data such as the description or the motivation that makes such evolution necessary, to the information concerning the interrelationship between patterns). In addition, we have completed (especially with regard to the *execution view*) the *KPI evolution metamodel*, so that it comprises all the concepts linked to KPI evolution, with specific attention to the aforementioned properties of traceability, consistency conditions and propagation effects. These components have been designed to make *AFFECTK* flexible enough to be adapted to particular KPI evolution strategies of organizations and to different KPIs specification approaches (Section 4).

Another specific contribution of this work (not present in our previous approach [8]) is that we demonstrate the effectiveness

of *AFFECTK* in two ways. First, in order to demonstrate that the framework achieves its purpose, we present a reasoning-based prototype as proof-of-concept of the theoretical framework (Section 5). We have developed this prototype using Formula [13], which is a tool for model finding and design space exploration mainly based on Constraint Logic Programming. Formula provides us with a solution to the formal specification of the KPI evolution strategy of any organization, allowing both constructive automated reasoning to perform correctness proofs on KPI systems and evolution patterns specification as logic program transformations. Second, we perform a more formal evaluation of the framework to measure how well it supports a solution to the problem it was defined for. This evaluation is carried out around four remarkable characteristics: *functional suitability*, *appropriateness*, *correctness* and *completeness* through the analysis of four different case studies (Section 6). Section 7 discusses our proposal and its threats to validity, while conclusions and further work are set out in Section 8.

2 | Related Work

Companies need to evolve and adapt their organizational and technical activities to the ever changing business circumstances [2, 6, 14–17]. This fact makes it necessary for all businesses to evaluate and modify their performance measures to confirm the continued relevance of such indicators, a fact that has been recognized both in research [6, 14–18] and in concrete practical applications [19, 20]. In this context, several works advocate the dynamic nature of measurement systems and suggest that organizations should have systematic processes in place for managing the evolution of their measurement systems [6, 14, 15, 17].

Specifically, the factors that affect performance measurement system evolution are analyzed in [14] without descending to the detail of specific indicators modifications. Additionally, the notion of measure redefinition is considered in [18], but the proposal is somewhat limited, since the different elements of a measure (dimensions and rule of calculation) are simply overridden when redefinitions are performed. Despite the importance of evolution, the literature review about KPI management performed in [5] concludes that very few works consider KPIs customization and evolution management as one of their characteristics.

As for practical applications, the Health Information and Quality Authority states in its report [19] that “a plan to review the KPI at regular intervals with a view to refinement in response to stakeholder demands or improved data availability” is necessary. This report also asserts that, since “health services are continually evolving”, it is fundamental that KPIs give answers to these changes.

Furthermore, change management is not only concerned with the change itself, since other issues, such as traceability, consistency and propagation change must also be taken into account [5]. First, speaking on the broader context of software systems, traceability of various artifacts created or modified during the development of software systems plays an important role in software evolution, providing benefits in areas such as reuse, change management, and maintenance [21]. Traceability information,

in general, can be used for analyzing and supporting system evolution [22]. In the case of KPI evolution, traceability helps maintain information about the different business elements related with the KPIs (goals, processes or the KPIs themselves) and keep a historical trace between the different evolution versions [8]. For example, the relationship between Process Performance Indicators (PPIs) and Business Process (BP) elements is considered in [10] by means of concrete structures that relate PPIs with BP elements, so that each PPI can always be traced back to the BP elements used in its definition. Associations among indicators and processes are also defined in [4, 11, 12]. Additionally, in those works, relationships between indicators are represented with more or less expressiveness (such as aggregation [4, 10–12] and derived measures [10, 12], causality or correlation [4]). However, to our knowledge, other works [10–12] do not keep historical trace of those relationships (between indicators and processes, or between indicators) as indicators change over time. An exception is [4], where concrete traces are kept regarding relationships between indicators as they change over time so that such information can be later used for specifying temporal expressions over Performance Indicators (PIs). In a slightly different line, authors of [23] present the abstract syntax of a specification language of measures that includes connections with other related elements (processes and dashboards). These connections enable the traceability of the modifications that have been made.

Besides traceability, since KPIs (and their components) are related among them, a change may also involve other consequent modifications to be performed (*propagation* change), to improve some aspect or maintain consistency of the system (*consistency conditions*). In this sense, the effects of classic create/update/delete operations of KPIs and, in particular, the need of consistency checking on the set of indicators is considered in [24]. As another example, authors in [4] take into account checking the consistency of PI structures when adding, updating, or deleting PIs. Additionally, in [6] validation rules checking is also considered. However, it seems that this work just addresses updating a PI formula, creating, and deleting PIs, together with a closed set of consistency conditions (*uniqueness*, *inequivalence*, and *coherence*). On the other hand, in [10], evolution of PPIs is connected to the changes made to the BPs thanks to the structures used to relate PPIs and their associated BPs elements, which, at the same time, allow coherence to be maintained across both assets. To the best of our knowledge, our AFFECTK framework would be the only one which not only considers these three evolution aspects within the KPI evolution management (traceability, consistency conditions, and propagation changes), but it also contemplates *customization* aspects regarding such three evolution issues aimed at coping with the particular circumstances and desires of each organization concerning its evolution policy (customization aspects such as deciding whether to register the historical trace between KPIs evolution versions or what actions to perform if consistency conditions are not satisfied when evolving a KPI). In this way, our KPI evolution proposal can be tailored to different evolution needs, requirements, and contexts, rendering it very versatile and flexible.

With regard to the type of conceptual artifacts provided in our proposal (metamodels and patterns), its usefulness and benefits are widely recognized [25, 26]. On the one hand, several works

make use of models and metamodels for different evolution concerns, such as structuring issues of evolution [27], tracing aspects [26, 28], or comparing and merging of models' versioning [29]. In the case of KPIs evolution, conceptual modeling can also be used as a development strategy. But, as far as we know, there is no KPI evolution metamodel in the literature. On the other hand, patterns are recognized to be a very useful construct to capture recurring concepts within a particular domain. Aimed at being generic and reusable in a wide variety of scenarios, they are specified in a language- and technology-independent way [30]. But again, we are not aware of any pattern devoted specifically to KPIs evolution.

Finally, regarding the existence of tool support for KPI evolution, although to the best of our knowledge there is no software tool combining the above-mentioned characteristics of traceability, consistency properties, and change propagation, several related work deserve to be mentioned. This is the case of the tool proposed in [6] that, giving support for the evolution of smart cities KPIs, does not take into consideration these three indicated characteristics. Focusing on traceability aspects, the proposal presented in [10] for PPIs also provides the *PPINOT Tool Suite* that, particularly, gives support to historic evolution of PPIs. As another example, the *Business Process Manager* tool of IBM allows users to perform KPI changes over time and includes the notion of *KPI history* [20] that can be used to track such KPI changes. Focusing on the strategy used for maintaining indicators consistency, we remark the following approaches. In [24], authors present the *KPIOnto Editor* for collaborative construction and maintenance of PI definitions based on the KPIOnto ontology. This tool bears some resemblance with our approach using reasoning functionalities (based on Logic Programming-LP) for PIs' formula manipulation and consistency checking supporting create/update/delete operations, but it does explicitly consider neither traceability nor propagation aspects. Although with another goal, authors in [4] use a Temporal Trace Language (TTL) checker tool as an alternative formal method for checking the consistency of a PI structure. In a different line, authors of [12] use reasoning technology for KPIs' evaluation to check whether they deviate from allowed values.

3 | Methodology

We frame our research using Peffers et al.'s Design Science Research (DSR) methodology process [9]. The DSR methodology process structures DSR in information systems by providing six steps which go from problem identification and motivation to communication of the results [9]. In this section, we address our proposal throughout the six steps of the DSR methodology process as shown in Figure 1.

Problem identification and motivation. The reference process typically starts with the identification of a research problem with practical relevance. Thus, in this first step of the process, we defined and motivated the research problem mainly by showing the need for an approach that allows organizations to achieve an adequate management of KPIs evolution, so that KPIs can be naturally adapted to the changes that occur over time. Indeed, we addressed this step in Section 1, where we illustrate that the scarce study of KPIs' adaptation and evolution

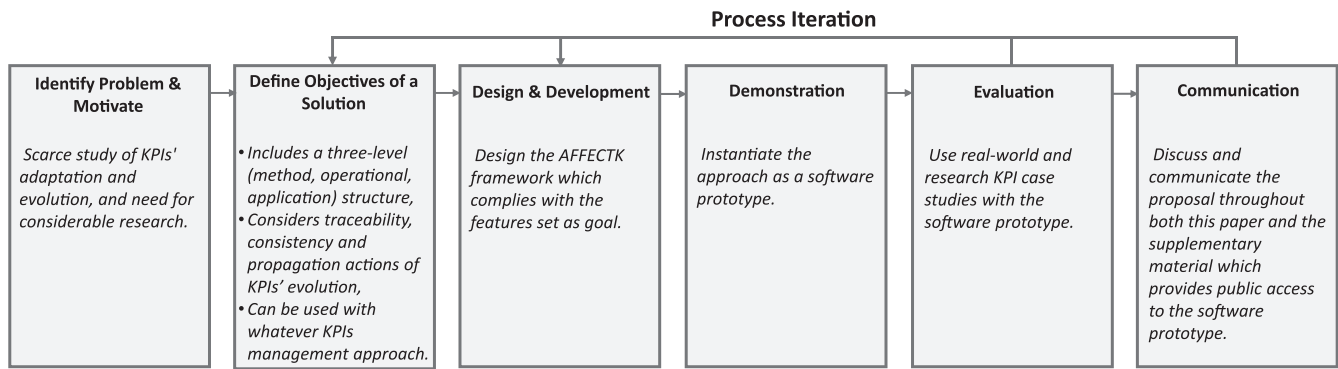


FIGURE 1 | Design science research process based on Peffers et al's [9].

has being identified as a theoretical and practical gap [5] and that there is a need for considerable research on the evolution of performance measurement systems [5, 6].

Definition of the objectives of a solution. The DSR methodology process by Peffers et al. [9] states for the second step that the objectives of a solution must be defined based on the identified problem and the literature. The objectives can be “[...] a description of how a new artifact is expected to support solutions to problems not hitherto addressed”. In our case, this step is addressed in Sections 1 and 2, where we mainly derive a solution objective that addresses the identified problem from the first step, considering knowledge of the state of problems and current solutions, as suggested by [9]. More specifically, our objective consists of establishing a KPI evolution framework which includes relevant characteristics compared to previous works, highlighting a three-level (method, operational, and application) working structure, while considering traceability, consistency, and propagation aspects of KPIs' evolution. These features have been determined to make the framework flexible enough to be adapted to whatever particular KPIs management approach. Based on this main objective, we design and develop our framework in the next research process step.

Design and development. In the third step, the artifact itself is created, which can potentially be “constructs, models, methods, or instantiations (each defined broadly)...[.]” [9]. In our case, we have designed *A Flexible Framework for Evolution, Consistency and Traceability of KPI (AFFECTK)* with the features stated previously. Additionally, the definition of such a framework is supported by the proposal we give for the notion of KPI evolution pattern and the KPI evolution metamodel. A detailed description of the design and development of the framework is presented in Section 4.

Demonstration. In the fourth step, a lightweight evaluation is performed to demonstrate that the artifact achieves its purpose in at least one context [9]. In this evaluation activity, prototypes are frequently used [31]. We thus have demonstrated the use of our framework by implementing a reasoning-based software prototype, which is presented as proof-of-concept of the theoretical framework. More specifically, we have developed our prototype according to a concrete KPI evolution strategy considering specific organizational decisions, thus giving evidence of the applicability of our framework [9]. This step is addressed in Section 5.

Evaluation. In the fifth step, a more formal evaluation is performed by observing and measuring how well the artifact supports a solution to the problem it was defined for [9]. In particular, the developed artifact has to be evaluated to demonstrate its applicability and usefulness [31, 32]. In order to do this, we have tested the prototype by applying it to four real-world and research KPI case studies, which is one of the procedures for artifact evaluation proposed in [31]. Very varied properties are pertinent to be assessed during this evaluation activity. One option is to determine these properties obtaining them from general-purpose quality models for software systems [33]. The most widely adopted quality model is the ISO/IEC 25010—System and Software Quality Requirements and Evaluation (SQuaRE) standard [34]. Among the characteristics proposed in this standard and taking into account that our prototype is a first practical validation of our theoretical framework (without focusing on being an exhaustive or general tool), we have considered the *functional suitability* the most appropriate for evaluating our tool. Details of this evaluation appear in Section 6.

Communication. In the last step, we discuss our proposal (Section 7) and, finally, communicate our research and findings with the relevant audience, throughout both this paper and the [Supplementary material](#) [35] which provides public access to the software prototype we have developed.

4 | The AFFECTK Framework for Supporting KPI Evolution

In this section, we propose *A Flexible Framework for Evolution, Consistency and Traceability of KPI (AFFECTK)*, by means of which we aim at providing a solution to the identified problem regarding the support of KPIs evolution management. The main outstanding features of this framework are, on the one hand, that it proposes a working structure that is based on the three well-known organizational levels: *method* level, *operational* level, and *application* level (not provided in our previous proposal [8]) and, on the other hand, that it is defined around three cornerstone properties concerning evolution: *traceability*, *consistency conditions*, and consequent *propagation* change.

Given these scaffolding properties, the constituent conceptual artifacts of the AFFECTK proposal are a set of *KPI evolution patterns* and a *KPI evolution metamodel*. The *KPI evolution patterns* provide a description or template of commonly occurring

evolution changes applied over KPIs as gathered in the literature and tools, especially focusing on key evolution aspects. The *KPI evolution metamodel*, defined upon our patterns, establishes the structure of concepts, relationships, and constraints involved in the definition and management of the evolution of KPIs. These two components, patterns and metamodel, were briefly described in [8], where we presented both the elements that conform the specification of a KPI evolution pattern and a first version of the metamodel. In this paper, we provide a complete description of the patterns and present an enhanced version of the metamodel.

4.1 | Overview of AFECTK

The AFECTK proposal is structured along three levels (see Figure 2). The most general level, the *method level*, provides the conceptual artifacts regarding KPI evolution management to be considered regardless both the specific organization and the elements used for KPIs specification. This level is composed of a set of *KPI evolution patterns*, that capture commonly occurring changes in KPIs, and a *KPI evolution metamodel*, that gathers the structural and execution aspects of the patterns (see Sections 4.2 and 4.3).

Taking as a starting point the components defined in the previous level, the *organizational level* reflects the structures and decisions that each organization (companies, public agencies, etc.) adopts when managing the evolution of its KPIs. If we examine Figure 2 again, we can see that each organization must make certain specific decisions that will end up constituting what we have called the *KPI evolution strategy* for that organization. We assume that the organization in question has already defined a collection of KPIs, according to its own needs. In order to correctly apply our framework, it is convenient for the organization to analyze this collection in the light of the KPIs taxonomy [5]. The use of this taxonomy makes it possible to conceptualize in a structured way the notions that appear in any KPI, which in turn will allow a first major type of decision to be made, concerning

which of the elements appearing in its KPIs will be susceptible to change and evolution. The second major type of decision that the organization will have to make is to choose which evolution patterns can be applied. The organization will be able to choose among those suggested in our framework (see Section 4.2) or propose its own (since the patterns we propose form an open collection). In any case, for each evolution pattern, the organization will have to establish its policy with respect to each of the adaptable options of the pattern (regarding traceability, consistency, and propagation). Finally, by customizing the execution view of the KPI evolution metamodel (see Section 4.3), the organization will be able to define its own specific *KPI evolution strategy*.

As for the *application level*, the organization-specific KPI evolution strategy defined in the previous level will be implemented in a particular tool for evolution management. Such a tool will allow users to manage the different KPI versions used by the organization over time, allowing historical information to be stored. Our framework does not prescribe the type of tool to be used, giving flexibility to each organization to choose the most appropriate form of implementation: from the integration of a specific module in the existing KPI management system, to the use of a stand-alone tool, or any other suitable variant. To demonstrate the feasibility of our proposal, in Section 5, we present, as a proof-of-concept, a reasoning-based prototype built on the formula model finding tool.

4.2 | KPI Evolution Patterns

Our main goal for defining KPI evolution patterns is to capture commonly occurring changes in KPIs. These patterns are conceived as flexible artifacts that, when a particular KPI evolution tool is developed, can be adapted to specific KPI evolution strategies of organizations and to different KPI specification approaches. The set of patterns is an open collection that can be extended to cover further KPI evolution types. As a way of example of KPI evolution patterns, four proposals were elaborated in [8] (Create KPI, Delete KPI, Update Calculation Rule

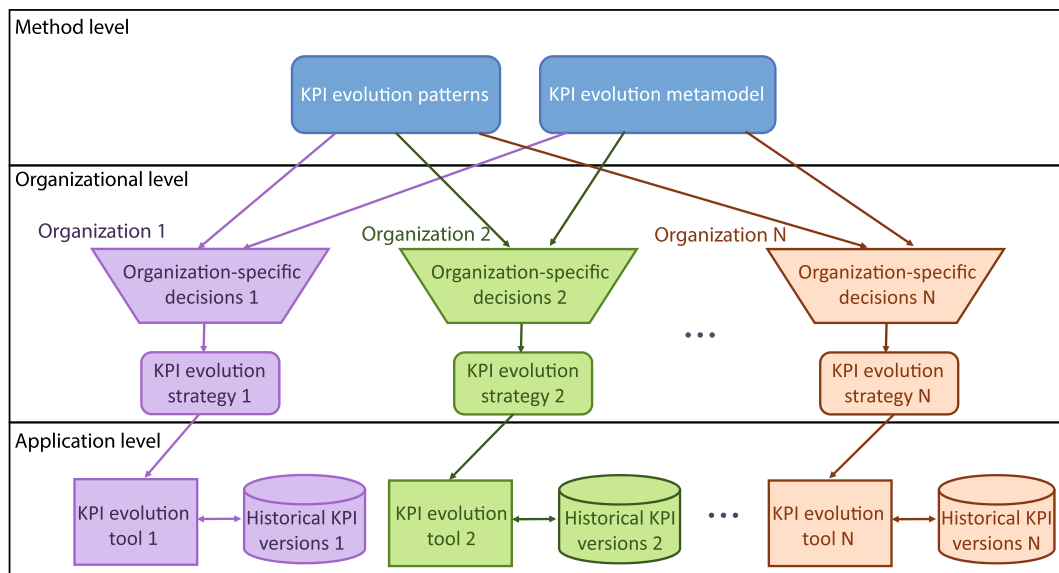


FIGURE 2 | The AFECTK framework.

and Update Target), based on literature on the topic [36, 37] and tools (such as *PPINOT* [10], *KPIOnto* [24], or the *Business Process Manager tool* of IBM [20]). These patterns have been revised and extended in this work, including also the Update Status Options pattern (see [35]). Other possible patterns for addressing more KPI changes could be the following: Update Name [24], Update responsible [10], Create relationship between KPI elements (for example, establishing that a KPI measures a process [4]), or Delete relationship.

In order to facilitate the specification of KPI evolution patterns, a pattern syntax proposal and several guidelines are provided. An example of a KPI evolution pattern following this syntax can be seen in Table 1. This table shows the “Update Calculation Rule” pattern (UCR pattern in the following) that addresses the change of the calculation rule of a KPI, thus evolving it to a new KPI version. There are different situations in which a calculation rule may need to be updated. For example, a modification in the evaluation criteria of an on-line course, penalizing the number of wrong answers, can imply a change in the formula calculating the learning adequacy of students [38]. As another example, the

G4 Sustainability Reporting Guidelines [39] were superseded by the GRI Standards [40], so that the changes introduced in the GRI Standards must be considered in the calculation of KPIs. This pattern gathers information regarding the change of KPIs’ calculation rules and provides a template that serves as reference of the core aspects to consider when managing such an evolution change. As it can be seen in Table 1, the first elements of a pattern specification are the *name*, an *abbreviation*, a *Description* of its functionality, *Examples* to clarify the pattern’s use, and a *Motivation* aimed at identifying the nature of the modifications in the environment which can trigger the change.

The remainder elements constitute the key aspects of each evolution pattern, since they describe different strategies and considerations organizations could adopt for managing the evolution change addressed by the pattern at hand. These sections are related to (1) the *Traceability* between the different KPI versions, (2) the *Consistency conditions* to be checked when evolving KPIs, together with actions to be performed if any of the conditions are not satisfied, and (3) the *Propagation* mechanisms that may be derived as consequence of the main change. In order to

TABLE 1 | Example of a KPI evolution pattern.

Pattern UCR: Update Calculation Rule	
Description	The calculation rule of a KPI is updated
Examples	- A modification in the evaluation criteria of an on-line course, penalizing the number of wrong answers, can imply a change in the KPI “Session Learning Adequacy per Training Step” (SLATS)³⁸ adding, for example, a factor of 1.5 to the number of reloads. - A company supersedes the G4 Sustainability Guidelines³⁹ for the GRI Standards⁴⁰, so that the changes introduced in the GRI Standards must be considered in the calculation of KPIs.
Motivation	Several reasons can motivate a change in the calculation rule of a KPI, for example, a change in the goals of the company, a change in the business processes or a change in other KPIs involved in the calculation rule.
Traceability	[T1] Store the fact that the update calculation rule pattern has been applied, linked to the new calculation rule. [yes] → [T2] Store the information related with the reason that provokes the calculation rule modification (goal, process or KPI change, ...) [yes] → [T3] Keep the previous calculation rule in order to know the evolution over time. [yes] → [T4] Keep the previous values of the KPI together with the previous calculation rule.
Consistency conditions	<i>General consistence conditions</i> to be checked to make sure that the system would remain consistent if the KPI’s calculation rule is updated: [C1.1] <i>uniqueness</i>: to check that the definition of an indicator is unique in the system, that is, there is no identical calculation rule²⁴. [C1.2] <i>inequivalence</i>: to make sure that a calculation rule is not equivalent to a calculation rule already in the system²⁴. [C1.3] <i>coherence</i>: to check that equivalence relations are preserved between system’s indicators and their constituent parts/members. More specifically, it must be checked that the calculation rule does not contradict any other already defined calculation rule²⁴, that is, any other dependent KPI. Additionally, it must be verified that the new calculation rule does not contradict targets and status options of the modified KPI. <i>Actions</i>: If any of the conditions are not satisfied, various policies could be implemented. We can distinguish several alternatives for action: [C2.1] If the <i>coherence</i> check is not satisfied because the new calculation rule is not coherent with the target or status options of the KPI, modify the target or the status options as consequently (related to propagation [P1.1]). [C2.2] If the <i>coherence</i> check is not satisfied because the KPI to be modified is part of the calculation rules of other KPIs, for each of those KPIs, decide: • keep the calculation rule as it is, including the modified KPI. • redefine the calculation rule using the previous calculation rule of the modified KPI (related to propagation [P1.2]). [C2.3] If the <i>coherence</i> check is not satisfied because of the existence of circular dependencies, avoid the evolution to be performed. In any other inconsistency case not considered previously, not to modify the calculation rule of the KPI (being this option the most common one).
Propagation	[P1.1] Modify other KPI attributes, such as target or status options, as appropriate, making use of patterns Update Target (Pattern UT) and Update Status Options (Pattern USO). [P1.2] Modify the calculation rules of dependent KPIs (KPIs whose calculation rules include the updated KPI), as appropriate, making use of pattern Update Calculation Rule (Pattern UCR). [trace and any propagation] → [P2] Store the propagation relationships between the pattern application facts stored because of the calculation rule modification.
Related patterns (Influences)	[P1.1] Update Target (Pattern UT) [P1.1] Update Status Options (Pattern USO) [P1.2] Update Calculation Rule (Pattern UCR)

make explicit the relations of propagation between patterns, the *Related patterns* influenced by the pattern that is being specified are also detailed.

Traceability. Traceability, in general, refers to keeping track of information to a certain degree. This aspect can also contribute to provide essential support for evolutionary change of the KPI systems within an organization. More specifically, traceability information could be used to support the analysis of implications and integration of changes that occur in KPI systems; to enable system acceptance by allowing users to better understand the system, contributing to a clear and consistent system documentation; and to support KPI system inspection, by indicating compromises made along time. In order to ensure that the anticipated benefits of these tasks can be realized, it is necessary not only to have an up-to-date set of traceability relations between the artifacts involved in the KPIs system evolution but also to maintain such traceability relations over time as the KPI system evolves in order to prevent their decay.

Although the aspects that can be considered in each pattern concerning traceability can be many and varied, we consider some general aspects that particularly point out:

- **[T1] Storage of the pattern application fact.** This aspect indicates the tracing of the pattern application itself, that is, the fact that the pattern has been applied. Also, when the application of the pattern gives rise to a new KPI element version, then the pattern application fact is linked to it.
- **[T2] Storage of the information related to the change that provokes the application of the pattern (goal, process, ...).** This aspect is useful for having a trace of the reasons that motivate the different KPI changes over time.
- **[T3] Maintenance of a historical trace between the different evolution versions.** This aspect allows the traceability of KPIs' changes over time to be handled. We can identify two situations. First, in *modification* patterns, each time the evolution pattern is applied, concrete evolution traces from the previous KPI version to the current one will be registered. Thus, this aspect entails keeping track of both previous and current versions of the specification of the KPI. Second, in the particular case of KPI *deletion* patterns, the previous KPI version would not be directly removed but treated as an "out-of-date" version. As a result, the registered traceability links will serve as a way to navigate between KPI versions, allowing the identification of the evolution history over time. This aspect does not apply for KPI *creation* patterns.
- **[T4] Maintenance of a historical trace between the different KPI values.** This aspect allows tracing the values of the KPI versions over time, so that a comparison among values, knowing the version of the KPI to which they correspond, is possible.

Consistency conditions. Ensuring consistency among KPIs during the successive evolution changes that take place over time is of major importance for managing reliable KPI systems. More specifically, to guarantee that consistency is preserved when performing an evolution change, suitable conditions must be verified. Thus, a requisite for consistency management when

addressing KPIs' evolution is to establish the conditions that must be preserved each time an evolution change takes place, which, at the same time, can vary among patterns. For example, when applying the "Create KPI" pattern, it must be ensured that no other KPI with the same name exists in the system; however, checking this condition when applying the UCR pattern makes no sense. Once established, the conditions to be satisfied in each type of evolution change, if a concrete KPI change does not preserve any of them, various policies could be implemented [24], being the most common one to avoid the evolution to be performed.

Taking this into account, we propose two types of aspects regarding consistency conditions:

- **[C1] Consistency conditions that can be compromised by the application of the pattern.** Each evolution pattern has to specify a set of consistency conditions that guarantee the system's consistency when performing the evolution change. For example, the UCR pattern (Table 1) defines the following general consistency properties: *[C1.1] uniqueness* check, to avoid to insert twice the same indicator's definition in the system; *[C1.2] inequivalence* check, to avoid duplicated indicators (i.e., to ensure that there are no indicators whose calculation rules are equivalent); and *[C1.3] coherence* check, to make sure that, when changing an indicator, equivalence relations are preserved between system's indicators and calculation rules [24]. These constraints can also be applied to other evolution patterns.
- **[C2] Actions to be performed if any of the consistency conditions is not satisfied.** In case a KPI evolution change does not preserve any of the fixed consistency conditions, different actions could be performed in order to maintain the logical consistency of the KPIs system [24]. While, as mentioned above, the most common action is avoiding the evolution to be performed, alternative policies can be proposed. When the alternative policies are related with some propagation mechanism, these mechanisms would be specified in the pattern. For example, the UCR pattern (Table 1) establishes in the [C2.1] action that if the coherence check is not satisfied because the new calculation rule is not coherent with the target of the KPI, then the propagation action of modifying the target can be performed in order to maintain the KPIs system consistency. These derived actions will be related with the corresponding options of the *Propagation* section.

Propagation. Since KPIs, and their components, are logically related to each other, an initially proposed change may involve a large number of modifications to various artifacts (either those they are built upon or other related elements). In our context, propagation is related with the incremental update mechanisms that could be necessary either (1) for maintaining/re-establishing the KPI system consistency among the different involved elements of KPIs (which could be related with actions described in option [C2]) or (2) for improving aspects, such as readability or compactness. Managing change propagation becomes complex and challenging [41] since, besides the impact from the main change, it also carries the accumulated impact of the given propagated change.

Similarly to traceability and consistency, we have identified some aspects that can be specified regarding propagation aspects, which are the following:

- **[P1] Propagated actions derived as consequence of the application of a pattern.** This aspect assesses, per each pattern, the changes that can be triggered as an effect of the pattern application. Since the propagation changes can be managed by other evolution patterns, they must be specified here. For instance, in the UCR Pattern (Table 1), two optional propagation actions are included. First, *[P1.1]* establishes that when the calculation rule of a KPI is modified, the specification of other KPI attributes, such as target or status options, can be analyzed and modified accordingly, making use of the “Update Target” pattern and the “Update Status Options” pattern, respectively. Second, *[P1.2]* establishes that the KPIs’ calculation rules that include the updated KPI can be analyzed and modified, making use of the same UCR pattern.

We propose to make it explicit the overall patterns that could be triggered as an effect of the application of the pattern at hand. For example, the patterns that could be applied to carry out propagation changes in the UCR pattern are specified in the *Related patterns (influences)* block (Table 1).

- **[P2] Storage of the propagation relationships between the applied patterns.** According to this aspect, the links between propagated patterns will be registered so that we can keep track of the propagation history. In this way, we register not only the information regarding the application of the involved patterns (the main pattern and propagated ones) but also the existing connections among them, thus registering the very complete history of patterns involved in the evolution change. In the software systems context, keeping track of the subsequent changes to other artifacts potentially affected by a change is widely supported in the literature [21], since it enables estimating the impact of changes. It can be similarly applied to the KPI evolution context.

4.3 | KPI Evolution Metamodel

The KPI evolution metamodel of AFFECTK includes two complementary views: a *structural view* and an *execution view*. While the *structural view* deals with the structural aspects of the evolution as defined by the KPI evolution patterns (see top of Figure 3), the *execution view* concerns the application of such patterns, registering mainly the way in which concrete KPI instances undergo modifications as patterns are instantiated (see bottom of Figure 3). A simpler initial version of this metamodel was proposed in [8]. In this paper, we revisit this metamodel and enhance it by introducing new elements and attributes related with the patterns and their application, the most remarkable changes being in the *execution view*.

In order to make the paper self-contained, we briefly summarize the main ideas behind each view and then go on explaining the new aspects we propose for the *execution view*. Before that, we

would like to remark that, trying to be as general as possible, we will explain our evolution metamodel using the terminology of the taxonomy presented in [5].

Structural view. The instances of the *structural view* metamodel represent the considered KPI evolution patterns. The metamodel includes a `Pattern` metaclass which contains the basic definition information of each KPI evolution pattern (such as the *name*, *abbreviation*, etc.). The KPI entities involved in a KPI evolution pattern are represented in the metamodel by the `KPIInvolvedElement` metaclass, associated with the `Pattern` metaclass. *Traceability* or *Consistency* aspects are represented by the association metaclasses `TraceOperation` and `ConsistencyCondition`, respectively, between the `Pattern` and `KPIInvolvedElement` metaclasses (see Figure 3). Finally, *Propagation* aspects are represented by the `PropagationOperation` metaclass and the *influences* and *influencedBy* association roles. The *optionallyApplied?* attribute, included in these metamodel elements, provides flexibility when developing a specific KPI evolution tool, representing whether the corresponding aspect will be always applied (false value) or whether the person performing a KPI evolution change by means of the tool is free to apply it or not (true value).

Execution view. This view (see bottom of Figure 3) represents aspects related to the application of the specific evolution patterns established in the *structural view*. More specifically, the instances of the *execution view* metamodel will represent the application of the concrete evolution patterns to the KPIs. In particular, we link this view with the previous one by means of the relationship between the `Pattern` metaclass and the `PatternApplication` metaclass. The `PatternApplication` metaclass includes concrete attributes representing specific information regarding the application of the pattern itself, such as “who” performs the change (*committer*) [29], “when” such changes are performed (*commitDate*) [29], any remarkable comment the committer considers necessary to make (*comments*), as well as the concrete motivation (*motivation*) to perform the change. This view mainly defines entities representing application instances of the ones defined in the *structural view* (`AppliedPropagationOperation`, `AppliedTraceOperation` and `AppliedConsistencyCondition`, together with the `AppliedKPIInvolvedElement`). In particular, when a concrete evolution pattern is applied to a KPI element, concrete evolution traces from a source element version (*AsIs*) to another target version (*ToBe*) of such an element can be registered (both versions being instances of the `AppliedKPIInvolvedElement` metaclass).

Considering the previous UCR pattern, the application of this pattern to a concrete KPI will be represented as an instance of the `PatternApplication` metaclass. This instance will also be linked to an instance of the `AppliedTraceOperation` metaclass, which, in turn, will be associated with the `AppliedKPIInvolvedElement` instance corresponding to the new “calculation rule” of the modified KPI.

We have mainly enhanced the metamodel we proposed in [8] by introducing new elements in the *execution view* (Figure 3).

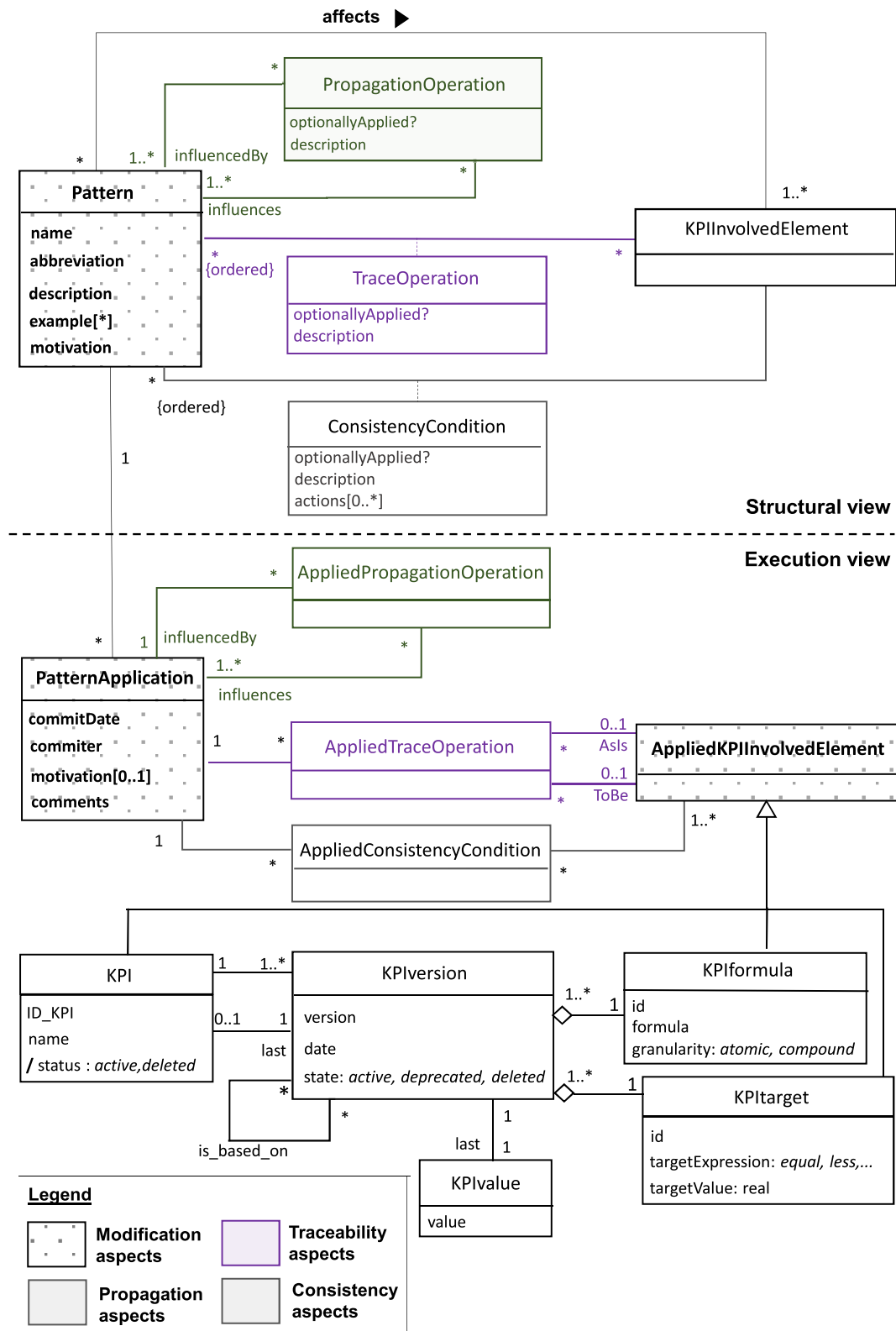


FIGURE 3 | KPI evolution metamodel.

More specifically, we have included in the metamodel a specialization hierarchy of the **AppliedKPIInvolvedElement** metaclass, together with information concerning KPIs' versions along time, attributes, and associations (see the bottom part of the *execution view* in Figure 3). The ultimate goal for this part of the metamodel is to represent the different

elements and attributes that can be used to represent the KPIs as long as the evolution patterns are executed. From now on, we will use the term *KPI metamodel* to refer to this part of the execution view. Let us note that the **AFFECTK** framework is still valid even if a different metamodel is considered for the KPIs.

As shown in Figure 3, each KPI (KPI) is associated to the set of KPI versions (KPIVersion) that represents the historical trace of the different versions it has gone through over time. Each version has several elements associated to it: a formula, a target, and its last value (KPIformula, KPItarget, and KPIvalue). Furthermore, the relationships among KPIs derived from the way they are calculated is represented in the metamodel by means of the `is_based_on` relationship.

Finally, we explain the different states a version can be in each moment (see `state` in KPIVersion of Figure 3). When a KPI *creation* pattern is applied, in addition to the KPI, a linked KPI version is created in the state `active`. When a *modification* pattern is applied on a KPI, then a new version is created in the state `active` and the state of the previous version is changed to `deprecated`. If it is decided to eliminate a KPI by applying a *deletion* pattern, then a new version is created in the state `deleted` (and the state of the previous version is changed to `deprecated`). In order to know the last version of each KPI, an association between the KPI and KPIVersion metaclasses is considered, where this latter one plays the role of `last`. We note that these versions' states will also configure the statuses a KPI can go through during its life as evolution patterns are applied, considering that the current status of a KPI is the state of its last version. This fact is represented in the metamodel by means of the derived attribute `status` of the KPI metaclass.

5 | Proof-of-Concept Implementation

Aimed at providing evidence of the feasibility of AFFECTK, as a first validation of our theoretical framework, we have implemented a reasoning-based prototype for KPI evolution management in order to demonstrate that the framework achieves its purpose [9]. This prototype, located at the *application level*, has been developed according to the *KPI evolution strategy* we have considered at the *organizational level*. Throughout this section, we will present the specific organizational decisions we have adopted to implement this prototype. We note that in this study, we intend to give evidence of the applicability of our framework, without focusing on providing an exhaustive or general tool that includes all the presented evolution capabilities and that can be integrated within any KPI management tool.

Our prototype implementation is based on two key premises that characterize our overall proposal. First, the elements considered by an organization for specifying its KPIs are given by its *KPI metamodel* (see Figure 3), while the set of KPIs considered by the organization (KPI system) is represented by means of an instance model conforming to such a metamodel. Second, the application of a KPI evolution pattern takes a source KPI system model and evolves/transforms it to a target KPI system model that includes the evolution change. These two aspects have motivated us to consider the Formula tool [13, 42] to implement our prototype. More specifically, Formula is a model finding and design space exploration tool that uses (1) *Algebraic Data Types (ADT)* and *Constraint Logic Programming (CLP)*, for formally specifying domain-specific languages (DSLs) including models, metamodels, and conformance [43], and (2) *transformations*, for representing model editing operations. Another

reason for choosing Formula is the extensive research work that have been recently performed on the verification of system models using this tool, in particular, to reason about UML modeling foundations [44, 45], yielding positive results. As we will see later, these appealing aspects provide us with a formal and practical framework for specifying KPI systems, verifying consistency conditions in such systems and managing KPIs evolution transformations. This functionality has been materialized in a prototype adequate enough to ensure the feasibility of the proposal presented in this paper. In contrast to other works where Formula is used to find instances conforming UML models (starting from either zero or partial knowledge of such instances), here we cover other Formula functionalities that our specific context requires. More specifically, starting from already predefined instances (the source KPI system considered by the organization), we need to manage edition operations (evolution transformations), ensuring consistency of both the source KPI system and the KPI system resulted after the evolution change. We also note that to the best of our knowledge, it is the first work which proposes Formula, or even CLP, not only to specify and check the consistency of KPI systems, but also to support their evolution aspects.

Next, we explain the key implementation aspects of our prototype, but a more complete and detailed explanation can be found on the supplementary material [35]. We first describe how we represent in Formula the metamodel and model aspects of our AFFECTK framework considering a particular KPI evolution strategy and go on presenting how we have implemented the evolution transformations.

Formalization of involved metamodel and models.

Formula supports different types of specifications for both formalizing design problem abstractions and defining instances of these abstractions: *domains* and *models*, respectively. More specifically, a Formula *domain* allows us to formalize an abstraction of each metamodel, to be used by Formula to reason about it. Formula *domains* specify algebraic data types (ADT) and a logic program describing properties of the abstraction; while ADTs constitute the key syntactic elements of Formula, logic program expressions, ensuring the remaining constraints, are provided by means of Formula *rules* and *queries* that are specified based on the defined data types [42]. A Formula *model*, for its part, is a finite set of data type instances built from constructors of the associated domain, which satisfies all its constraints [13]. Once a model is combined with its domain, then queries can be evaluated.

Taking this into account, we represent the constructors of our proposal, that is, the *execution view* metamodel, by means of a Formula *domain*, while each concrete KPI system model is represented by means of a Formula *model*. More specifically, the defined Formula *domain* includes suitable ADTs to represent the *execution view* metamodel's structural elements distinguishing among both KPIs aspects or *KPI Metamodel* (such as KPI, KPIFormula, KPITarget, and `is_based_on` ADTs) and evolution aspects (such as `AppliedTraceOperation`, `AppliedConsistencyCondition`, and `AppliedPropagationOperation` ADTs). *Rules* have been used to specify implications and generate knowledge (e.g., to infer transitive relationships among KPIs, generating new

instances of the `relatedKPIs` ADT or to calculate KPIs' values, generating instances of the `calculatedValues` ADT). *Queries* have been used to encode requirements or correctness properties to be checked in a model (e.g., consistency conditions such as coherence and uniqueness, among others). Finally, having represented the *execution view* metamodel in Formula, the KPI system of an organization is implemented by means of a *model* with the corresponding set of data type instances built from constructors of the *execution view* metamodel domain. At this point, Formula can be used to detect whether the source KPI system to be evolved is initially *satisfiable*, that is, if it verifies all the constraints established in the domain. Otherwise, Formula informs that the model is *unsatisfiable*, indicating also what Formula queries are not satisfied. Such queries can give the user a clue about what aspects have motivated such a problem, so that the user can make subsequent analysis. Starting from a satisfiable KPI system, we can then set in motion the evolution process by applying the evolution transformation patterns.

Formalization of Evolution Transformation Patterns. We have implemented the patterns: Create KPI, Delete KPI, Update Calculation Rule, and Update Target, by means of four different Formula transformations. The strategy followed to implement such transformations has been that consistency conditions and propagation aspects are always applied (the `optionallyApplied?` attribute of both `ConsistencyCondition` and `PropagationOperation` is set to `false`), while the person in charge of applying such transformations is allowed to make decisions regarding traceability aspects (the `optionallyApplied?` attribute of `TraceOperation` is `true`).

Briefly speaking, a Formula transformation is encoded as a logic program so that, based on one or several input models, it computes an output model according to such logic program [13]. In our transformations, such input (aliased as `IN`) and output (aliased as `out`) models conform the *execution view* metamodel domain. In addition to the input model, our transformations are provided with another two types of parameters as extra information: (1) the ones needed by the own transformation to perform the evolution task (such as the name of the KPI to be deleted in the Delete KPI pattern) and (2) the ones regarding specific decisions the person in charge can take when applying the transformation. Regarding the logic included in each transformation, we distinguish the next four different blocks:

- **Basic instructions.** This block performs two main tasks. In the first one, previous information of the source KPI system, related to KPIs other than the evolved KPI, are copied to the output KPI system model without undergoing any change. Additionally, this block performs the evolution change itself considering neither traceability, consistency conditions nor propagation aspects of such evolution change (aspects addressed in the remainder blocks).
- **Traceability.** The implemented transformations leave it up users to decide whether or not to apply any of the available traceability options, in such a way that they can create suitable instances of both the `PatternApplication` and the `AppliedTraceOperation` data types, registering

the `ToBe` information (*[T1]*), store the corresponding motivation change (*[T2]*), maintain in the target KPI system model the previous version of the KPI and all the elements that constituted such previous version, registering the `AsIs` information (*[T3]*), and generate instances of `basicValue` or `calculatedValue` (depending on the type of KPI) associated to previous versions (*[T4]*).

- **Consistency conditions.** The defined transformations use Formula queries to implement consistency conditions such as *coherence*, *uniqueness*, *nonRepeatedName* (so that KPIs with same names cannot be created), *activeState* (this latter one ensuring that transformations are applied to active KPI versions), and other structural conditions. The *inequivalence* constraint has not been implemented since Formula does not allow inferring formulas from other formulas. All queries defined in a transformation are included in a general query which is checked when executing the evolution transformation so that we can know whether the target evolved KPI model satisfies them. When a concrete query implementing a consistency condition is not satisfied, an `AppliedConsistencyCondition` instance is generated to register such a fact.
- **Propagation.** Formula does not allow us to automatically trigger the execution of a transformation from another transformation, and thus performing patterns propagation, but we have followed an alternative strategy in our prototype. This strategy consists on simulating propagation aspects by registering pending propagation actions, so that the person in charge can assess their future application. In particular, a `PatternApplication` element is defined so that it contains a `state` attribute which registers the real state of a pattern application distinguishing among `executed`, for patterns already applied, and, in the contrary, `InStudyOptional/InStudyCompulsory`, in order to inform the user about what patterns are convenient or necessary, respectively, to be applied caused by the propagation of other pattern's application. Choosing between one value or another (`InStudyCompulsory` or `InStudyOptional`) depends on whether such propagation is needed to maintain the KPI system consistent or it is just for improving the system readability or compactness, respectively.

Figure 4 shows a screenshot of the Formula Visual Studio integration where a concrete KPI system has been checked for satisfiability before being evolved. More concretely, the central side of the figure shows an excerpt of the implementation of the proposal. The right-hand side of the figure shows, on the top, the implementation units together with the transformations, and on the bottom, the milliseconds it takes to verify on the KPI system model each constraint defined in the domains. The top left-hand side of this figure shows the Formula results obtained after checking the source KPI system. More specifically, it indicates what constraints are satisfied (green filled circle) and what are violated (red filled circle). Additionally, both the KPI system elements and the facts automatically generated by the Formula reasoning tool (such as `calculatedValues` or `relatedKPIs` instances) are also included on the bottom left-hand side. Formula also comes with a command line interface more suitable for performing transformations. By means of this interface,

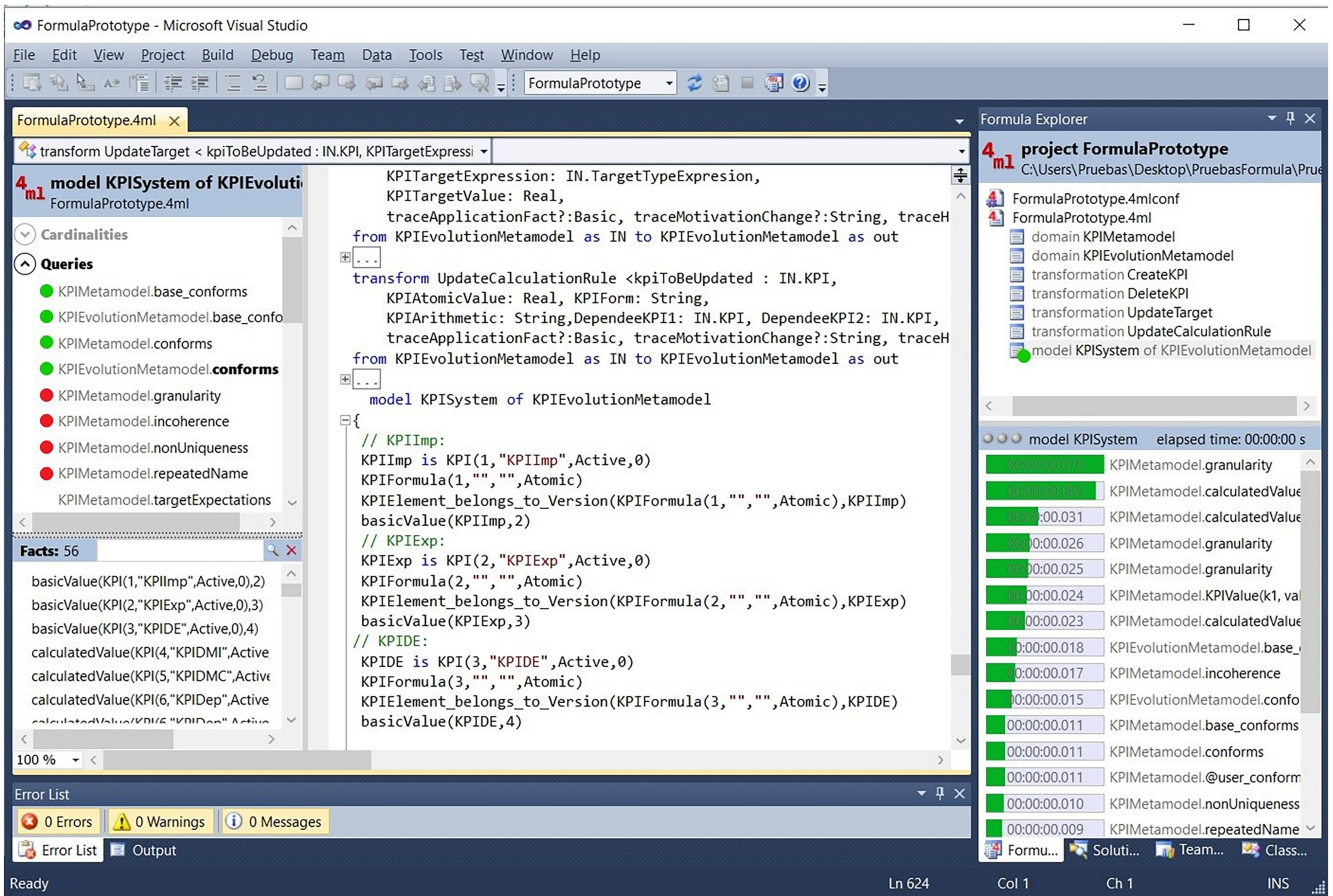


FIGURE 4 | Formula prototype screenshot.

Formula outputs the satisfied/violated constraints (i.e., satisfied consistency conditions and detected inconsistencies) together with a report including not only the KPI system that resulted from the application of the evolution transformation but also the instances generated concerning traceability, violated consistency conditions, and propagation actions.

6 | Evaluation

As described previously, we have performed a more formal evaluation of the framework to measure how well it supports a solution to the problem it was defined for [9]. As it is mentioned in [5], practical evaluations must be performed in order to validate the pertinence of a KPI management approach, being very varied the evaluated characteristics (applicability to different scenarios, effectiveness, flexibility...). In our case, we have evaluated our AFFECTK framework by means of several actions with the goal of assessing several characteristics of both the theoretical framework and its practical application.

More specifically, our reasoning-based prototype tool has served as a proof-of-concept that shows the *applicability* and *effectiveness* of our theoretical framework. The resulting tool provides a way to manage KPI evolution changes by means of Formula transformations, also reporting KPI evolution structures regarding traceability, consistency conditions, and propagation actions. Although being implemented according to a specific KPI

evolution strategy, the prototype exposes the *flexibility* of the proposal to be tailored to organizations' evolution needs, which we consider as one of the proposal's strong points.

Additionally, we have evaluated the *functional suitability* of our proposal by using our prototype for managing KPIs evolution. The functional suitability is one of the characteristics included in the ISO/IEC 25010 - System and Software Quality Requirements and Evaluation (SQuARE) standard [34]. We have considered that, among the characteristics proposed in the product quality model of SQuARE, this is the most timely for evaluating our tool taking into account that, as we have said previously, the prototype is a first practical validation of our theoretical framework (without focusing on being an exhaustive or general tool). The *functional suitability* is further subdivided into three quality subcharacteristics: *functional appropriateness*, *functional correctness*, and *functional completeness*. We have evaluated these properties by applying the prototype to four KPI systems. More specifically, we have selected these indicators from creditable sources (research projects [46], health programs [47], and research papers [6, 24]), mainly because they are varied in the application context and nature, so that they also serve as a test of the *applicability* of the proposal to *different scenarios*. Some additional relevant information related to the evaluation performed can be found on the [Supplementary material](#) [35].

Table 2 depicts remarkable data regarding the selected KPI systems. The first case study (CS1) consists of a set of 60 KPIs

TABLE 2 | Case studies.

CS	Source	Reference	Context/type of Indicators	Atomic KPIs	Compound KPIs	Total KPIs
CS1	Research project	[46]	Urban areas/circular economy and waste prevention	14	46	60
CS2	Health program	[47]	Health/mental health	8	3	11
CS3	Research paper	[24]	Collaborative enterprise/general enterprise	6	5	11
CS4	Research paper	[6]	Governance/smart city governance	11	1	12

presented in a report developed within the UrbanWINS-Innovative Strategic Plans for Urban Waste Reduction and Management project [46]. These indicators are related to circular economy and waste prevention and management. The second case study (CS2) has been obtained from the IAPT (Improving Access to Psychological Therapies) program [47]. This program was established in 2008 in order to improve access to Psychological Therapies for people with Depression and Anxiety Disorders. Since then, the program has defined a family of several indicators which have evolved along time. These indicators are composed by KPIs and Headline indicators as an agreed mechanism for measuring patients' progress; while KPIs are at the tactical level of management and control performance, Headline indicators are strategic indicators used to support change decisions [47] (considered as atomic and compound, respectively, in Table 2). This family of indicators has already been used in other research papers such as [48, 49]. The third case study (CS3) refers to a KPI system formed by 11 indicators obtained from the paper [24], where authors use this system as case study to illustrate their framework for collaborative construction and maintenance of dictionaries of PIs. Finally, the fourth case study (CS4) is a real-world KPI system proposed in [6] for evaluating the degree of smartness and sustainability of smart cities. Especially remarkable is the fact that authors claim the necessity of supporting evolution of KPIs as consequence of the evolution of cities' needs and context. Next, we describe our experiments, as well as the remarkable findings drawn from their analysis.

Functional suitability. This characteristic represents the capability of a product to provide functions that meet stated and implied needs of intended users [34]. We have evaluated this property by applying the prototype to several indicators defined in the report of CS1 and proving that the evolution information produced by our tool is detailed enough and useful to track the performed changes. More specifically, the report of CS1 defines a set of 60 indicators distinguishing between 35 *waste indicators* and 25 *circular economy indicators*, organized into 14 atomic and 46 compound ones. In this experiment, we have started from several of the circular economy indicators defined in the report and performed different changes in order to check that the produced evolution information (regarding tracing, consistency conditions, and propagation aspects) is useful to track the performed changes. In particular, we have started from six indicators shown in Table 3 which consist of three atomic (Imp, Ext and DE) and three compound indicators (DMI, DMC and Dep). Later,

we have made two main sequential evolution changes, consisting of the creation and the deletion of KPIs, by applying the corresponding Formula transformations. In these transformations, we have supposed that the person in charge has decided to apply all traceability aspects (recall that consistency conditions and propagation aspects are always applied in the transformations implemented in the prototype). Finally, we have analyzed (1) the Formula output regarding both satisfied consistency conditions and detected inconsistencies and (2) the report produced by our transformations regarding traceability, detected inconsistencies, and propagation aspects. Next, we explain such evolution changes in more detail.

- First, we have supposed the need to create a new indicator called *Physical trade balance* (PTB) defined as the difference between physical imports and physical exports, that is $Imp - Exp$ (see Table 3). Thus, we have applied the `CreateKPI` transformation, with the corresponding arguments (such as the new indicator's identifier, if it is atomic or compound, the indicators it is based on, or the reason for the change). As it was expected, Formula has shown that the indicator does not contradict any consistency condition from those implemented (*nonRepeatedName*, *uniqueness*, and *coherence*). Additionally, and according to the `CreateKPI` pattern, the transformation execution informs the user (by means of suitable instances of `AppliedPropagationOperation`) about the propagation actions she/he could take, choosing between: (i) modifying the calculation rules of other KPIs as appropriate, making use of the new KPI, and/or (ii) deleting or creating other KPIs, considering the creation of the new KPI. In this case, since such propagation actions are not needed to maintain the consistency of the KPI system, they are identified as `InStudyOptional`. Based on such suggested propagations and the considered KPI system, we have decided to redefine the indicator DMC so that it includes PTB ($DMC = DE + PTB$), by applying the `UpdateCalculationRule` transformation. Again, Formula has shown that the propagated action does not contradict any inconsistency condition and proposes an additional compulsory propagation action to check whether the new calculation rule is coherent with the target or status options of the DMC indicator (see [35] for more details).
- Second, we have supposed that the indicator DMI is no longer needed in the system and we have applied the `DeleteKPI` transformation. In this case,

TABLE 3 | Indicators considered in CS1.

Indicator	Calculation	Description
Imp	Mass amount of goods entering the geographic area	<i>Imports.</i> Goods entering the economy/territorial unit.
Exp	Mass amount of goods leaving the geographic area	<i>Exports.</i> Goods leaving an economy/territorial unit.
DE	Mass amount of raw material extracted from the natural environment	<i>Domestic extraction.</i> DE is the annual amount of raw material (except for water and air) extracted from the natural environment.
DMI	DE + Imp	<i>Direct material input.</i> Measures the direct input of materials for use into the economy.
DMC	DMI - Exp	<i>Domestic material consumption.</i> Measures the total amount of material directly used in an economy (excluding indirect flows).
Dep	Imp/DMI	<i>Dependency on other systems.</i> This indicates the share of DMI that is from imports.
PTB	Imp - Exp	<i>Physical trade balance.</i> A physical trade surplus indicates a net import of materials, whereas a physical trade deficit indicates a net export.

Formula has informed us that the *coherence* condition is not satisfied since there is an indicator (Dep) that depends on DMI (it generates the corresponding `AppliedConsistencyCondition`). As a consequence, the transformation report indicates (by means of the generated `AppliedPropagationOperation` instances) to choose among different options to maintain the system consistency: (i) modifying the calculation rules of the KPIs that include the KPI to be deleted, so that they do not contain the KPI or (ii) deleting the KPIs that include the KPI to be deleted (actions identified as `InStudyCompulsory`). Additionally, it suggests to assess whether the person in charge wants to delete or create other KPIs, considering the deletion of the KPI (action identified as `InStudyOptional`). In particular, we have decided to delete the indicator Dep as propagated change, so that we have applied the `DeleteKPI` transformation over this indicator leading to neither new inconsistencies nor additional compulsory propagation actions.

Findings. The information reported by our framework is detailed enough to track all performed changes (both main and propagated evolution changes), as well as violated consistency conditions. Furthermore, we have found that the more aspects organizations and/or the person in charge decide to register, the more detailed the reported data. It is therefore important for organizations to be clear from the outset about the level of detail they want to obtain when reporting on the effects of an evolution process.

Assumptions and limitations. In this experiment, we have made several assumptions. We have proposed some evolution changes to illustrate functional suitability. That is, we assume that the organization at hand could have considered, as part of its evolution policy, the possibility of performing at least the evolution changes considered in the experiment. More specifically,

we have first considered evolution changes involving compound indicators (rather than atomic) to enrich the experiment by producing more complex evolution information (such as dependencies among indicators or propagation actions). Second, regarding the evolution changes themselves, while the creation of the indicator PTB constitutes a natural evolution change in the life cycle of the KPIs' system (all indicators are created at least once), we have assumed both the deletion of the indicator DMI (in order to simulate the occurrence of an inconsistency in the system) and the propagated actions derived from both evolution changes, thus giving us the possibility to register richer evolution information. Another assumption we have made is related to the evolution aspects to be registered. According to the above findings, the more information the organization decides to record, the better it will be able to track the performed changes. Therefore, if we had chosen to record less information, the report produced by our transformations would have been less detailed. The other way around, the increase in the amount of information recorded will generally require more resources (both managerial and computational), which may be considered a limitation of the proposal. A limitation shared by all the experiments we have performed in this evaluation is that we have only considered the evolution changes supported by the developed prototype (see next experiment for details). Another limitation that has already been described in Section 5 (since it is closely related to the use of the Formula tool) refers to the fact that the tool does not allow to check whether the *inequivalence* condition is met (included, e.g., in the Create KPI or the Update Calculation Rule patterns) or to automatically trigger the execution of a transformation from another transformation (that is the reason why we have had to manually perform propagation aspects). This limitation of the tool is also shared by the experiments performed in the remaining case studies.

Functional appropriateness. This subcharacteristic is defined as the capability of a product to provide functions that

facilitate the accomplishment of specified tasks and objectives [34]. To this respect, we have evaluated the feasibility of the proposal for managing evolution by performing the real evolution changes that the IAPT's indicators (CS2) have undergone along the nine versions published by the IAPT program [47]. The KPI system is composed of a total of eight atomic indicators which remain the same from the first version. Regarding the compound indicators, the IAPT service started with three indicators. One of such indicators was deleted in the fifth version. Regarding the two compound remaining ones, in the fifth version, they were deleted and two new ones were created with different names than the previous ones but with the same formula. That is, the absence of KPI evolution management obligated them to create new indicators with different names, instead of evolving the previous ones. In order to be faithful with the evolution changes undergone on these compound IAPT's indicators, in our experiments, we have performed the corresponding deletion/creation actions. However, we want to note that, in this situations where new indicators are created instead of performing the evolution of existing ones, we bet for both performing the change and registering traceability information related to the evolved change, thus keeping track of the information that explains the evolution between indicators. In particular, starting from the initial version of the system including eight atomic and three compound indicators, we have performed the evolution changes corresponding to the fifth version of the system (the deletion of a compound indicator and the corresponding deletion/creation actions to give rise to the two new compound ones). To sum up, in this experiment we have (1) created eight atomic and three compound indicators included in the first version of IAPT's indicators (obtaining, as expected, a consistent KPI system as stated by Formula), (2) deleted three compound indicators, and (3) created two new ones, successfully obtaining, as a result, the last version of the KPI system. Thus, given that the *KPI evolution strategy* implemented by the prototype considers evolution patterns for creating and deleting KPIs, we have been able to address the evolution changes the indicators of the IAPT program have undergone along time.

Findings. We highlight the importance for the organizations of having a well-thought-out evolution policy, so that they can manage evolution changes as required by emerging circumstances (such as modifications of the business strategy, legislative updates, or technological adaptations [2]). In case this policy is not completely defined or not defined with sufficient precision, our framework can help (especially thanks to the existence of patterns) to analyze and determine it.

Assumptions and limitations. Since the reasoning-based prototype tool we have developed implements the Create KPI and Delete KPI transformation patterns (in addition to the Update Calculation Rule and Update Target patterns), we have been able to manage the evolution changes performed in the IAPT's indicators along the nine versions published by the IAPT program [47]. Had the IAPT program considered, as part of its evolution policy, the possibility of performing evolution changes other than the supported by the tool, the tool would have had to be extended to support them. As in this case study, the CS3 and CS4 cases present specific KPIs' evolution changes, so the evaluation performed with these case studies shares this limitation.

Functional correctness. This subcharacteristic refers to the capability of a product to provide accurate results when used by intended users [34]. Within our context, the main aspect related to accuracy is the assurance of KPIs consistent evolution. For this reason, we have focused on ensuring KPIs' consistency along evolution changes by establishing consistency conditions and further consequent propagated changes. More specifically, we have considered several evolution changes to be applied to the KPI system presented in CS3 so that they provoked different incoherence situations in the system, some of them requiring consequent propagated changes to be performed to ensure the final system's consistency. Among such evolution changes, we have performed an update of a KPI's calculation rule so that it leads to circular dependencies among KPIs' formulas. More specifically, starting from the KPI system presented in CS3, we have assumed that the person in charge wants to perform an update of the calculation rule of the compound KPI `PersonnelTrainingCost` (defined as `HourlyCost*PersonnelTrainingTime`) by changing `HourlyCost` by another one that measures taxes in hourly cost (lets say `HourlyCostTaxes`). However, he/she makes a manifest error when selecting the `HourlyCostTaxes` KPI (choosing `InvestmentInEmplDevelopment` whose formula is `PersonnelTrainingCosts + TeachCost`). While the change does not contradict the *uniqueness* consistency condition, the new formula makes the *incoherence* query to be violated. Thus, according to the actions to be performed when inducing circular dependencies as established by the considered `UpdateCalculationRule` pattern, the evolution change must be aborted to ensure the system's consistency (see Table 1). Something similar occurs when consequent propagated changes, derived from the initial evolution change, must be performed to ensure consistency (such as the ones we have shown in CS2). Other evolution changes applied to the KPI system resulting in different incoherent situations in the system have been the deletion of an indicator that has other dependent indicators, the introduction of a new indicator with the same name as another one, or the introduction of a new indicator with the same formula than another one (see [35] for more details).

Findings. The evolution changes performed in CS3 show the importance of establishing well-defined consistency properties and required consequent propagation actions to guarantee the KPI system's consistency. The absence of mechanisms such as consistency conditions and propagation changes would lead to inconsistencies in the system when applied evolution patterns, which would be unacceptable.

Assumptions and limitations. Additionally to the limitations mentioned previously (related to the supported evolution changes, *inequivalence* condition check and automatic transformation trigger), we note that if the organization at hand had considered additional consistency constraints to those included in the transformation patterns implemented in the prototype, those transformations would have to be adequately extended.

Functional completeness. This subcharacteristic is related to the capability of a product to provide a set of functions that covers all the specified tasks and intended users' objectives. With the goal of providing complete support for KPI evolution, the research paper of the case study CS4 [6], considering that KPI modifications are produced as a consequence of changes

in the smart city model, classifies the KPI evolutions into three categories: *additive changes*, *subtractive changes*, and *structural changes*. For this reason, we have used CS4 for evaluating the functional completeness of our tool. More specifically, we have started from the 12 smart cities KPIs considered in [6], which are organized into 11 atomic and one compound ones, and we have performed the KPI evolutions proposed by the authors for each one of the three considered categories. An *additive change* can be motivated by the introduction of new services on a city, giving rise to new KPIs. For example, in CS4, the Bicycle network KPI (BN) has been added due to the development of a cycling infrastructure previously nonexistent. A *subtractive change* implies the deletion of a KPI that can be obsolete or redundant. For example, in CS4, the Particulate matter 2.5 emissions KPI ($E_{PM2.5}$) has been deleted considering that it is redundant with the Air quality index for particulate matter 2.5 KPI. Finally, *structural changes* provoke changes in the definition of KPIs (calculation rule, target,...). For example, in CS4, after the advent of 5G, the calculation of the Network coverage KPI (NC) has been modified by including the 5G network coverage. Finally, we have been able to perform the proposed evolution changes (the creation and the deletion of KPIs, and the modification of a KPI calculation logic) by applying the corresponding Formula transformations, demonstrating that our proposal provides functions for covering the three types of changes considered in [6]. More specifically, we have applied the corresponding Formula transformations (CreateKPI, DeleteKPI and UpdateCalculationRule, respectively) which, particularly, have shown that none of the evolutionary changes contradict any consistency condition from those implemented by the corresponding transformation. Additionally, in all cases, the transformations' executions have informed about the propagation actions that can be performed.

Findings. AFFECTK provides suitable functions to cover the three categories of changes proposed in [6], so we can conclude that our proposal is complete in that sense.

Assumptions and limitations. Since our intention is to demonstrate the support our proposal gives to the *functional completeness* sub-characteristic, the findings cited above rely upon the KPI evolution classification the experiment is based on. While we have assumed that the category of changes proposed in [6] is complete, if any other classification with a category not included in [6] were found, we would have to evaluate the capability of our proposal to cover the evolution changes considered in such a category.

The evaluation we have performed using the ISO/IEC 25010 SQuARE standard [34] focuses on *quality* criteria, allowing us to analyze and evaluate four different aspects of our proposal and, in particular, of our reasoning-based prototype. However, other features and properties based on other criteria, for example, on *success* measures, could have been evaluated [33]. Specifically, it remains future work to prove the practical relevance of our framework, related to the success of using the tool in real environments. This evaluation should be performed with users and systems in a real setting, for example, by means of field experiments or expert interviews [31], embracing the complexity of human practice in real organizations [33].

7 | Discussion

To discuss our proposal, considering its threats to validity, assumptions, and limitations, we divide the discussion into four dimensions. First, we review the implications of the three cornerstone aspects of the proposal (*traceability*, *consistency*, and *propagation*). Then, we detail the *efforts required to adopt* AFFECTK and describe *when to adopt it*. We close the section with a *critical review* of the proposal as a whole.

Traceability, consistency conditions, and propagation. AFFECTK allows performing the evolution of indicators systematically and coherently, characteristics provided by the three cornerstone elements it includes.

More specifically, as advanced previously, traceability information could provide support for analyzing implications and integration changes taken place in KPI systems to allow users to better understand the system, or to support KPI system inspection. For these benefits to be realized, a traceability strategy must be established, aligned with the desired goals [50]. Our proposal envisages the recording of four varied traceability aspects that address, from different perspectives, the recording of evolution when applying patterns, as described in Section 4.2 (application of each pattern itself [T1], change that provokes such an application [T2], historical trace between KPI versions [T3], and historical trace between the different KPI values [T4]).

When performing an evolution change, consistency must be preserved among the involved KPIs, to ensure that the KPI system remains reliable [24]. Our framework considers the possibility of not only establishing the necessary consistency conditions when applying an evolution pattern but also implementing concrete policies when any of the consistency conditions is not satisfied (such as performing concrete propagation mechanisms to other indicators).

The logical relationships existing among KPIs and their components motivate the third key aspect of our proposal. The propagation element refers to derived actions to be performed as a consequence of a KPI evolution change, aspect widely supported in the literature within the software systems context [21]. In particular, the performed evaluation has pointed out the crucial importance for organizations to establish both well-defined consistency properties and required consequent propagation actions to guarantee the KPI system's consistency when performing an evolution change. Additionally, since such propagation changes can be managed, at the same time, by other evolution patterns, our proposal also considers the possibility of registering the propagation relationships between the applied patterns. This aspect allows the organization to keep track of the change propagations by analyzing and estimating the influences among patterns.

Following on from storage issues and related to traceability and propagation aspects, AFFECTK allows the organization to decide the level of detail it wants to obtain when reporting on the effects of an evolution process. Storage could be implemented at different levels of granularity. A coarse-grained recording is easier (and cheaper) than a fine granularity level, but it has a negative effect on benefits. For its part, a fine-grained recording

will increase the complexity of the system and tends to increase the costs because there will be more information to record, increased numbers of transactions, new systems, and procedures [51, 52]. We note that the registered information could constitute a useful source of data on how evolution changes take place, which could be used as a key factor in decision-making. As drawn from the evaluation, the more aspects registered, the more detailed the reported data. The challenge is to strike a balance between the desired tracked data and derived costs. For this reason, each pattern leaves the choice of deciding the desired granularity level.

7.1 | Effort needed to adopt AFFECTK

The cost of implementing the proposal corresponds to two main tasks. First, it is assumed that, as stated in [14], the organization has a dedicated quality specialist (or team) who is an expert in the organization's business goals. This specialist should have the necessary skills to manage the organization's KPIs over time, including the ability to refine measures to reflect the organization's new circumstances and the ability to identify when indicators are no longer appropriate to measure the organization's performance [14], being able to decide what kind of evolution change should be made to them to adjust to the new needs. We highlight this last aspect since, as stated in the findings obtained from the evaluation, organizations need to have a well-thought-out evolution policy, so that they can manage evolution changes as required by emerging circumstances. The specialist should act according to an effective process for reviewing, modifying, and deploying measures as the organization's circumstances change [14]. In our framework, he/she would be in charge of making the specific decisions that constitute the *KPI evolution strategy* for the organization, that is, (1) which KPIs' elements are susceptible to change and evolve, establishing the corresponding instances of the bottom part of the *execution view* in the *KPI evolution* metamodel, and (2) the potential evolution changes allowed for its KPIs that establish the concrete evolution patterns that can be applied in the organization (*what*), including the aspects that are left open in the patterns (*how*), thus configuring the corresponding instances of the *structural view* metaclasses. In this latter case, for each evolution pattern, the specialist will have to establish the organization's policy concerning each of the adaptable options of the pattern regarding traceability, consistency, and propagation, ensuring a proper alignment of evolution changes with consistency and propagation actions to guarantee the system coherence after the change. The multiple degrees of freedom in configuring the policy for the evolution of the organization's KPIs make, precisely, our proposal very versatile and flexible. The above-mentioned specialist will be the prospective user of the tool.

In this regard, one possible threat to the validity of our approach is that it works on the assumption not only that such a specialist or team is available in the organization and also that the organization considers a measurement's review process. These aspects are also highlighted in [14] among the complex range of inter-related factors that affect the evolution of performance measurement systems, indicating that any organization must have them in place for measurement systems to evolve effectively. Nevertheless, we believe that the need for such a role, together

with a formal review process, constitutes an intrinsic aspect of an organization with a measurement culture [14], regardless of whether or not it adopts the AFFECTK framework.

Second, an effort needed to adopt our proposal refers to the implementation of the *KPI evolution strategy* itself either as a software module integrated into the existing KPI management system used by the organization or as a stand-alone tool. In fact, as stated in [6, 14], the lack of flexibility of information systems, especially accounting systems, constitutes a barrier to the evolution of measures by several companies. In this respect, some companies tend to design new systems from scratch to be flexible enough to enable measures to be changed easily [14]. Even with the difficulties acknowledged in the literature for extending existing performance measurement systems, one of the advantages of AFFECTK is that it can be used with whatever particular KPIs management approach. More specifically, our framework mainly relies on both, a set of *KPI evolution patterns*, completely customized by the organization at hand, and the *KPI evolution metamodel*. As for the *KPI evolution patterns*, if the organization's specialist decides, among the potential evolution changes allowed for its KPIs, the possibility of performing a change not considered by the evolution patterns included in the supplementary material [35] (i.e., Create KPI, Delete KPI, Update Calculation Rule, Update Target, and Update Status Options), then such a specialist would have to define the corresponding evolution pattern as well as implement it. Additionally, as mentioned previously, specially in case a fine-grained recording strategy is chosen, suitable software artifacts would have to be implemented not only to manage the reported data but also to provide a clear, effective, and well-designed data reporting that allows users to see and understand the performed evolution actions for decision-making. Regarding the *KPI evolution metamodel*, it considers elements general enough to model KPIs coming from "off the shelf systems" (which refer to the bottom part of the *execution view* metaclasses of the metamodel). For example, in the case of the UCR pattern of Table 1, the `KPIInvolvedElement` could be referred, for example, as "measure definition", or as "formula", depending on the KPI terminology considered by the concrete system. The adaptation of these general elements to the specific terminology of an organization has to be taken into account before the application of our proposal. The remainder elements defined by the metamodel, which are included in both the *structural* and the *execution view*, complement with evolution capabilities the existing structural elements in those systems, according to the evolution patterns with their three cornerstone elements. The inclusion (or implementation, in case of the development of a stand-alone tool) of these elements in the system would need to consider the natural relationship among the elements in both views (structure versus application of evolution patterns). Additionally, whereas the implementation of the structures related to the three cornerstone elements themselves will be up to the developer, certain considerations can be taken into account. For example, while traceability aspects and the storage of propagation relationships could be considered independent from each other (the organization can decide either one, the other, or both independently), specific attention should be paid to the implementation of propagated actions when they are used for maintaining/re-establishing the consistency of the KPI system and, at the same time, are related to actions to be performed if consistency conditions are not

satisfied. Thus, in this latter case, concrete programming structures will have to be established to maintain the coherence between both types of actions.

7.2 | When to adopt AFFECTK

First of all, we note that even before considering the adoption of any KPIs evolution management proposal, the existing KPI management system must be used actively in the organization before it can evolve [14]; thus, the organization has to make a clear commitment in this regard so that such an adoption makes sense. With this in mind, any organization considering the adoption of our framework should analyze and evaluate the changing nature of its indicators in both a backward- and forward-looking manner. In this regard, the organization could consider aspects as diverse as changes in the internal and external environment, revisions and prioritizations of objectives as the environment changes, the deployment of changes in objectives and priorities, and the attempt to ensure that the gains achieved through improvement programmes are sustained [53]. The result of this analysis could be varied, including situations such as a large set of indicators that change little over time or a small number of indicators that undergo many changes over time. After performing this task, the organization should assess whether it would be worth adopting the AFFECTK framework by evaluating the benefits the framework would provide to the organization, against the effort needed to integrate it into its management system. To make this decision, it will also be necessary to evaluate other aspects, such as the impact that the use of the framework will have on the environment and users [31].

7.3 | Review and critical assessment of the AFFECTK proposal

The main contribution of this paper is a comprehensive approach to the management of KPI evolution, which considers traceability, consistency, and propagation aspects and also contemplates customization to be established by the organization at hand. Instead of providing a solution either too general (which would not be optimal for adapting to the specific needs and variability of all organizations) or too specific to an organization (so that it could be difficult to be applied beyond such an organization), we bet for a solution capable of adapting to the evolution policies and strategies required by each organization. While this decision could add a certain complexity to the proposal, we aim to reduce this complexity by establishing a three-level conceptual architecture, namely, *method*, *organizational*, and *application*. More specifically, to achieve the aforementioned adaptability, concrete structures at the *method* level are required to support the particularities of each organization. These structures correspond to a set of *KPI evolution patterns* and a *KPI evolution metamodel*. While patterns and metamodels are well-known software engineering artifacts, their use requires being familiar with their ins and outs. Therefore, the group of people that applies our solution to an interested organization (that is, at the *organizational* level) must be multidisciplinary, including not only a quality specialist (or team) but also software engineers that transfer the organization's needs to the *application* level. Although this could be seen as a limitation of the proposal, we

argue that the complexity of the problem justifies that the organizations allocate their resources to effectively manage KPIs and their evolution. One of the difficulties that could arise when applying our solution is related to the use of the evolution patterns. More specifically, although we provide as a contribution several patterns related to the most frequent KPI evolution changes (and which include different alternatives of traceability, consistency, and propagation conditions), a concrete organization could have specific evolution needs that require either the definition of additional patterns or the modification of the existing ones (e.g., to consider different consistency conditions to be applied when applying a pattern). Our proposal expressly includes traceability, consistency, and propagation aspects both in the patterns and the metamodel, guiding the quality specialist (or team) in their decisions concerning KPIs management. On the other hand, this fact requires them to be familiar with these concepts. Another of the complexities previously mentioned has to do with the decisions that the organization has to make regarding the consistency and level of detail of the registered evolution information, given that a balance must be struck between the desired tracked data and the (computational) derived costs involved in such recording.

Regarding the *application* level, in this paper, we have presented a prototype that has been useful in identifying certain limitations that may appear during the practical application of the proposal. For example, we have chosen *Formula* to develop the prototype, because of its reasoning-based capabilities that allow us to easily implement evolution transformations and consistency conditions. However, our proof-of-concept has shown certain limitations in capturing concrete aspects of our proposal, such as the impossibility of imposing certain consistency conditions and simulating evolution propagations. This aspect serves as a lesson learned for each organization to remember that, depending on the platforms or tools it uses, some elements of the proposal may be harder to implement. We have also used this prototype to demonstrate that the framework achieves its purpose. While we have performed a more formal evaluation based on the ISO/IEC 25010 SQuaRE standard where we have focused on *quality* criteria, other evaluation alternatives are possible [31, 33], such as field experiment or expert interview.

Taking into account all the aspects outlined above, the limitations of our approach do not stray much from those barriers affecting the evolution of performance measurement systems identified by Kennerley and Neely [14], which can be broadly categorized under the headings: *culture*, *process*, *people*, and *infrastructure*. More specifically, it is crucial to establish an organizational *culture* that embraces the use of performance measurement to manage the business and that is open to change [14]. This culture is necessary for any evolution measure system and, thus, it is a prerequisite for our framework. From another point of view, the adoption of AFFECTK can act as a prescriber of this culture by highlighting the need for the evolution of indicators in changing environments. Similarly, a *process* is required for reviewing, modifying and deploying measures as the organization's circumstances change. Without considering the importance of the existence of this somewhat "basic" process, it is difficult to take into account the incorporation of more complex frameworks such as the one we are proposing. From a proactive point of view, AFFECTK provides certain tools (especially

conceptual ones, such as patterns) that could assist in the definition of this key process. Additionally, our proposal relies on a specialist or team (*people*) with skills not only to manage the organization's KPIs over time, acting according to the effective process mentioned previously, but also to make the specific decisions that constitute the *KPI evolution strategy* for the organization. Last but clearly not the least, an *infrastructure* that gives support to the framework according to the organization's *KPI evolution strategy* is needed, which consists of a software module integrated into the existing KPI management system or as a stand-alone tool. To sum up, in relation to these four aspects, we consider that our framework offers significant advantages to the evolution of performance measurement systems that compensate the possible effort required to implement it, as stated in this section.

8 | Conclusions and Future Work

This paper presents AFFECTK, a framework aimed at supporting the evolution management of KPIs. The framework is characterized by two main features. First, it provides conceptual structures (a set of evolution patterns, and an evolution metamodel) that establish a context in which each organization can precisely define its KPI systems and its concrete evolution strategies. And secondly, it determines, through the above structures, the set of characteristics that need to be taken into account for any KPI system to evolve in a coherent way: traceability, consistency conditions, and consequent propagation changes. The feasibility of the approach has been demonstrated by the implementation of a proof-of-concept reasoning-based prototype, together with its evaluation along four remarkable characteristics: functional suitability, appropriateness, correctness, and completeness.

Several lines of work could be opened to empower and extend the present research. Among the most promising directions, the framework could be extended so that the decisions each organization makes regarding its KPI evolution strategy could be parameterized, thus facilitating the semi-automatic generation of KPI management tools. The framework could also be extended by providing additional patterns for addressing more KPI changes, including Update Name, Update Responsible, and Create Relationship. Furthermore, we could enhance the evaluation proposal by focusing on the success of the framework in real environments, embracing the complexity of human users. Another interesting direction is to take the most of our reasoning-based implementation proposal and extend it to verify user specified properties on KPI systems, in the same way that is explored in [44, 45] regarding UML models.

Data Availability Statement

Data sharing is not applicable to this article as no new data were created or analyzed in this study.

References

1. B. Estrada-Torres, A. del-Río-Ortega, M. Resinas, and A. R. Cortés, "On the relationships between decision management and performance measurement," in *Proceedings of the 30th international conference of*

advanced information systems engineering (caise 2018), Lecture Notes in Computer Science, Vol. 10816 (Springer, 2018): 311–326.

2. R. Cognini, F. Corradini, S. Gnesi, A. Polini, and B. Re, "Business Process Flexibility-A Systematic Literature Review With a Software Systems Perspective," *Information Systems Frontiers* 20, no. 2 (2018): 343–371.

3. D. Parmenter, *Key Performance Indicators: Developing, Implementing, and Using Winning KPIs* (New York, NY, USA: John Wiley & Sons, Inc., 2007).

4. V. Popova and A. Sharpanskykh, "Modeling Organizational Performance Indicators," *Information Systems* 35, no. 4 (2010): 505–527.

5. E. Domínguez, B. Pérez, A. L. Rubio, and M. A. Zapata, "A Taxonomy for Key Performance Indicators Management," *Computer Standards & Interfaces* 64 (2019): 24–40.

6. M. De Sanctis, L. Iovino, M. T. Rossi, and M. Wimmer, "MIKADO: A Smart City KPIs Assessment Modeling Framework," *Software and Systems Modeling* 21 (2021): 281–309.

7. M. Reichert and B. Weber, *Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies*, (Springer, 2012).

8. E. Domínguez, B. Pérez, A. L. Rubio, and M. A. Zapata, "Towards a Framework for KPI Evolution," in *Proceedings of the 15th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2020)* (2020): 463–469.

9. K. Peffers, T. Tuunanen, M. Rothenberger, and S. Chatterjee, "A Design Science Research Methodology for Information Systems Research," *Journal of Management Information Systems* 24, no. 3 (2007): 45–77.

10. A. del-Río-Ortega, M. Resinas, C. Cabanillas, and A. R. Cortés, "On the Definition and Design-Time Analysis of Process Performance Indicators," *Information Systems* 38, no. 4 (2013): 470–490.

11. C. Momm, R. Malec, and S. Abeck, "Towards a Model-Driven Development of Monitored Processes," *Internationale Tagung Wirtschaftsinformatik 2007-Band 2* (2007), 319.

12. B. Wetzstein, Z. Ma, and F. Leymann, "Towards Measuring Key Performance Indicators of Semantic Business Processes," in *Business Information Systems: 11th International Conference, BIS 2008, Innsbruck, Austria, May 5-7, 2008. Proceedings 11*, Lecture Notes in Business Information Processing, Vol. 7 (Berlin Heidelberg: Springer, 2008): 227–238.

13. E. K. Jackson, T. Levendovszky, and D. Balasubramanian, "Automatically Reasoning About Metamodeling," *Software & Systems Modeling* (2013): 1–15.

14. M. Kennerley and A. Neely, "A Framework of the Factors Affecting the Evolution of Performance Measurement Systems," *International Journal of Operations & Production Management* 22, no. 11 (2002): 1222–1245.

15. P. Garengo, S. Biazzo, and U. S. Bititci, "Performance Measurement Systems in SMEs: A Review for a Research Agenda," *International Journal of Management Reviews* 7, no. 1 (2005): 25–47.

16. C. Searcy, S. Karapetrovic, and D. McCartney, "Integrating Sustainable Development Indicators With Existing Business Infrastructure," *International Journal of Innovation and Sustainable Development* 1, no. 4 (2006): 389–411.

17. C. Searcy, "Corporate Sustainability Performance Measurement Systems: A Review and Research Agenda," *Journal of Business Ethics* 107 (2012): 239–253.

18. C. Schütz and M. Schrefl, "Customization of Domain-Specific Reference Models for Data Warehouses," in *Proceedings of the IEEE 18th International Enterprise Distributed Object Computing Conference (EDOC)* (2014): 61–70.

19. HIQA: Guidance on Developing Key Performance Indicators (KPIs) and Minimum Data Sets to Monitor Healthcare Quality Version 1.1. Health Information and Quality Authority, (2013): Online at <https://www.hiqa.ie/sites/default/files/2017-01/KPI-Guidance-Version1.1-2013.pdf>. Last visited on January, 2025.

20. I. B. M., Knowledge Center. Business Monitor 8.5.6, "KPI history," (2015), Online at https://www.ibm.com/support/knowledgecenter/it/SS7NQD_8.5.6/com.ibm.wbpm.mon.widget.ic.doc/topics/kpihist_ov.html. Last visited on January, 2025.
21. H. Schwarz, J. Ebert, V. Riediger, and A. Winter, "Towards Querying of Traceability Information in the Context of Software Evolution," in *Proceedings of the 10th Workshop Software Reengineering* (2008): 144–148.
22. G. Spanoudakis and A. Zisman, "Software Traceability: A Roadmap," *Handbook of Software Engineering and Knowledge Engineering* 3 (2005): 395–428.
23. J.-P. Friedenstab, C. Janiesch, M. Matzner, and O. Muller, "Extending BPMN for Business Activity Monitoring," in *Proceedings of the 45th Hawaii International Conference on System Sciences (HICSS)* (2012): 4158–4167.
24. C. Diamantini, D. Potena, and E. Storti, "SemPI: A Semantic Framework for the Collaborative Construction and Maintenance of a Shared Dictionary of Performance Indicators," *Future Generation Computer Systems* 54 (2016): 352–365.
25. D. Karagiannis and H. Kühn, "Metamodelling Platforms," in *Proceedings of the 3rd International Conference EC-Web 2002*, LNCS, Vol. 2455 (2002): 182–196.
26. M. Ruiz, S. España, O. Pastor, and A. González, "Supporting Organisational Evolution by Means of Model-Driven Reengineering Frameworks," in *Proceedings of the 7th International Conference on Research Challenges in Information Science (RCIS)* (2013): 1–10.
27. C. Pons, G. Baum, and R.-D. Kutsche, "Model Evolution and System Evolution," *Journal of Computer Science and Technology* 1, no. 3 (2000): 18.
28. E. Domínguez, J. Lloret, B. Pérez, A. Rodríguez, A. L. Rubio, and M. A. Zapata, "Evolution of XML Schemas and Documents From Stereotyped UML Class Models: A Traceable Approach," *Information and Software Technology* 53, no. 1 (2011): 34–50.
29. A. Cicchetti, F. Ciccozzi, and T. Leveque, "A Solution for Concurrent Versioning of Metamodels and Models," *Journal of Object Technology* 11 (2012): 1:1–32.
30. N. Russell, W. M. P. van der Aalst, and A. H. M. ter Hofstede, *Workflow Patterns: The Definitive Guide* (MIT Press, 2016).
31. C. Sonnenberg and J. Vom Brocke, "Evaluation Patterns for Design Science Research Artefacts," *Practical Aspects of Design Science*, M. Helfert and B. Donnellan, (eds.) (Berlin Heidelberg: Springer, 2012): 71–83.
32. S. T. March and G. F. Smith, "Design and Natural Science Research on Information Technology," *Decision Support Systems* 15, no. 4 (1995): 251–266.
33. J. Venable, J. Pries-Heje, and R. Baskerville, "FEDS: A Framework for Evaluation in Design Science Research," *European Journal of Information Systems* 25, no. 1 (2016): 77–89.
34. ISO/IEC 25010:2023: Systems and Software Quality Requirements and Evaluation (SQuaRE)-Product Quality Model. International Organization for Standardization (2023).
35. The AFFECTK framework, "Supplementary Material," Online at <https://zenodo.org/record/3627954>.
36. C. G. Schütz, B. Neumayr, M. Schrefl, and T. Neuböck, "Reference Modeling for Data Analysis: The BIRD Approach," *International Journal of Cooperative Information Systems* 25, no. 02 (2016): 1650006:1–1650006:46.
37. B. Estrada-Torres, A. del Río-Ortega, M. Resinas, and A. Ruiz-Cortés, "Identifying Variability in Process Performance Indicators," in *Proceedings of the Business Process Management (BPM) Forum*, (2016): 91–107.
38. A. Calabrò, F. Lonetti, and E. Marchetti, "KPI Evaluation of the Business Process Execution Through Event Monitoring Activity," in *Proceedings of the International Conference on Enterprise Systems, (ES)*, (2015): 169–176.
39. Global Reporting Initiative, "G4 Sustainability Reporting Guidelines," (2013), Online at <https://www.commdev.org/wp-content/uploads/pdf/publications/Global-Reporting-Initiative-G4-Sustainability-Reporting-Guidelines.pdf>. Last visited on January, 2025.
40. Global Reporting Initiative, "GRI Sustainability Reporting Standards," (2016), Online at <https://www.globalreporting.org>. Last visited on January, 2025.
41. X. Zhang, M. Zolghadri, and P. Girard, "Changes Handling Architecture for Co-Evolution of Products and Networks of Partners," *Projetcs / Projéctica / Projectique* 5 (2010): 41–62.
42. FORMULA-Modeling Foundations, (2020), Website: <http://research.microsoft.com/en-us/projects/formula/>. Last visited on January, 2025.
43. J. Joxan and M. J. Maher, "Constraint Logic Programming: A Survey," *Journal of logic programming* 19/20 (1994): 503–581.
44. B. Pérez and I. Porres, "Reasoning About UML/OCL Class Diagrams Using Constraint Logic Programming and Formula," *Information Systems* 81 (2019): 152–177.
45. B. Pérez, "Reasoning About Clinical Guidelines Based on Algebraic Data Types and Constraint Logic Programming," *Journal of Biomedical Informatics* 92 (2019), <https://doi.org/10.1016/j.jbi.2019.103134>.
46. J. Bastos, R. Garcia, and F. Freire, "Indicators for Waste Prevention and Management - Measuring Circularity," (2019), UrbanWINS project- Innovative Strategic Plans for Urban Waste Reduction and Management. Online at https://edisciplinas.usp.br/pluginfile.php/7005693/mod_resource/content/1/d2.3-Indicators-for-waste-prevention-and-management.pdf. Last visited on January, 2025.
47. IAPT, "Improving Access to Psychological Therapies, Key Performance Indicators (IAPT KPIs)," Online at <https://digital.nhs.uk/data-and-information/publications/statistical/improving-access-to-psychological-therapies-key-performance-indicators-iapt-kpis>. Last visited on January, 2025.
48. E. Roubtsova and V. Michell, "Modelling and Validation of KPIs," in *Third International Symposium on Business Modeling and Software Design (BMSD 2013)*, (2013): 96–105.
49. S. Griffiths and S. Steen, "Improving Access to Psychological Therapies (IAPT) Programme: Setting Key Performance Indicators in a More Robust Context: A New Perspective," *Journal of Psychological Therapies in Primary Care* 2 (2013): 133–141.
50. S. Maro. (2017)., "Addressing Traceability Challenges in the Development of Embedded Systems," Ph.D. Thesis.
51. E. Golan, B. Krissoff, F. Kuchler, L. Calvin, K. Nelson, and G. Price, "Traceability in the US Food Supply: Economic Theory and Industries Studies," *Agricultural Economic Report Number 830*. (Washington, DC, 2004).
52. J. Cleland-Huang, O. Gotel, and A. Zisman Eds., *Software and Systems Traceability Edited by J. Cleland-Huang, O. Gotel, and A. Zisman* (Springer, 2012).
53. U. S. Bititci, T. Turner, and C. Begemann, "Dynamics of Performance Measurement Systems," *International Journal of Operations & Production Management* 20, no. 6 (2000): 692–704.

Supporting Information

Additional supporting information can be found online in the Supporting Information section.