Computing and Informatics, Vol. 37, 2018, 109–141, doi: 10.4149/cai_2018_1_109

TOWARDS A FORMALIZATION OF A FRAMEWORK TO EXPRESS AND REASON ABOUT SOFTWARE ENGINEERING METHODS

Miguel Morales-Trujillo*

Facultad de Ingeniería, Universidad Nacional Autónoma de México
Mexico City, Mexico
e-mail: migmor@ciencias.unam.mx

Hanna Oktaba, Francisco Hernández-Quiroz

Facultad de Ciencias, Universidad Nacional Autónoma de México
Mexico City, Mexico
e-mail: {hanna.oktaba, fhq}@ciencias.unam.mx

Boris Escalante-Ramírez

Facultad de Ingeniería, Universidad Nacional Autónoma de México
Mexico City, Mexico
e-mail: boris@servidor.unam.mx

Abstract. Software Engineering is considered a knowledge-intensive discipline, in which knowledge creation, collection and sharing is an uninterrupted process. However, a large part of this knowledge exists in a tacit form and depends on practitioners. Therefore defining a mechanism to transform tacit knowledge into explicit one is of upmost importance. This paper presents a formalization approach to represent Software Engineering practitioners' tacit knowledge, which is related to their ways of working, as a set of explicit statements. The formalization is based on KUALI-BEH, which is a normative kernel extension of ESSENCE formal specification, and consists of three parts: an ontology to share a common representation of knowledge

* corresponding author

as a set of concepts; a Situational Method Engineering based algebra that represents well-defined method properties and operations; and a knowledge representation of the ontology and algebra using Description Logics. The main objectives of this initial formalization are to improve communication among humans and machines, computational inference and reuse of knowledge.

Keywords: Software engineering, situational method engineering, ontology, description logics, ESSENCE, KUALI-BEH

Mathematics Subject Classification 2010: 68-N30

1 INTRODUCTION

Precisely specifying the process by which a Software Engineering activity takes place is a challenging task [1]. Every aspect of software development, particularly in large systems, demands a great deal of knowledge and understanding of the software practitioner [2].

Moreover, according to [3] and [4], creating software is one of the most knowledgeintensive professions. Software Engineering community has been motivated to collect all the knowledge that practitioners possess; the activity of knowledge gathering has become a relevant research line for the discipline.

Since knowledge creation is an uninterrupted process, it is true about its collection as well, thus we need a process to manage it. In [5] it is established that knowledge management process should address all of the following tasks:

1. to acquire new knowledge;
2. to transform the knowledge from tacit or implicit into explicit knowledge;
3. to systematically store, disseminate, and evaluate knowledge; and
4. to apply knowledge in new situations.

Providing Software Engineering with a knowledge-based approach allows us to create models and reason about them. At this point the wide scope of the discipline becomes an obstacle. Presentation and integration of the knowledge-based approach into the everyday working world of software engineers is a critical challenge for the Knowledge-Based Software Engineering (KBSE) community [4].

In 1992 [4] identified three crucial questions to give Software Engineering a knowledge-based focus:

1. What part of the software process is targeted?
2. What knowledge is applicable and how can it be represented, acquired, and maintained?

3. How can we present the knowledge to developers, teams, and managers to improve the quality, cost, and timeliness of software development?

By finding answers to these questions, that are still valid, researchers will be able to create a model to represent the knowledge of a targeted software process. Once the model of a process is precisely defined in a formal manner, process analysis techniques can be applied to such a model to identify problematic and erroneous steps, or to leverage efficiency improvements [1].

The objective of this paper is to present a formalization, which was created as a way of improvement of the communication among humans and machines, and of reasoning about the tacit knowledge possessed by Software Engineering practitioners. In particular, the paper is focused on the practitioners' ways of working during software projects. The proposed formalization is built on KUALI-BEH [6], a Normative Annex of ESSENCE – Kernel and Language for Software Engineering Methods [7], which is an Object Management Group (OMG) formal specification. The proposal uses three types of formalization: an ontology to share a common representation of knowledge as a set of concepts; an algebra based on Situational Method Engineering (SME) to represent well-defined method properties and operations; and a knowledge representation of the ontology and algebra using Description Logics (DL).

The motivation behind creating a formalization based on an ESSENCE Kernel extension is the lack of reasoning mechanisms in metamodels, like SPEM [8], or standards like ESSENCE itself. Moreover, the main reason that motivated the formalization was to provide Software Engineering practitioners with a mechanism to reason about the knowledge they possess. As [9] stated, there is a need to provide *"a simple specification language to describe any type of activity in a company, in a concurrent and modular fashion"*, which is also useful for software engineers.

This paper is organized as follows: the background is presented in Section 2, Section 3 demonstrates the proposed formalization, Section 4 describes its validation, Section 5 mentions the intended usage of the formalization, and conclusions and future work are discussed in the final section.

2 BACKGROUND

This section presents KUALI-BEH as the object to be formalized, and the following formalization approaches: Ontologies, SME and DL.

2.1 ESSENCE – Kernel and Language for Software Engineering Methods

In 2011 OMG initiated a standard project, the outcome of which was ESSENCE: a four layer approach that defines a kernel, a language and permits the construction of software engineering methods, see Figure 1 (left side).

The ESSENCE kernel is a set of universal components involved in software engineering efforts, which are expressed in a language by the syntax and semantics

rules. The path that guided the definition of ESSENCE was the separation of concerns, and it therefore defines three areas of concern: Customer, Solution and Endeavor. Another relevant concept is activity space, which according to [7], is the representation of the essential things to do; it describes the challenges a team faces when developing, maintaining, and supporting software systems. Within each area of concern an activity space is provided, in which practitioners can model the state of a particular software project with the help of a previously defined set of Abstract-Level Project Health Attributes (ALPHAs), see Figure 1 (right side).

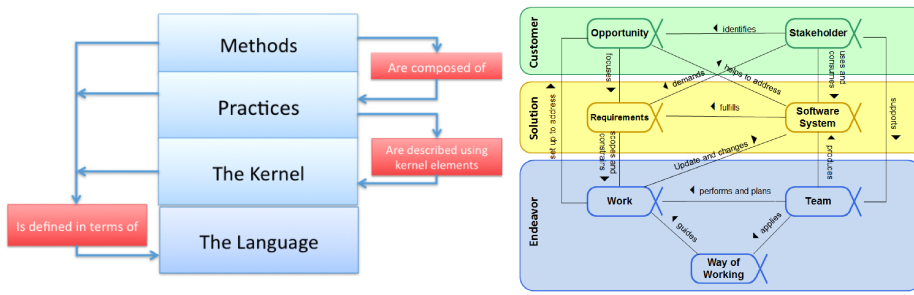


Figure 1. ESSENCE architecture and its areas of concern [7]

The Endeavor area of concern contains everything to do with team members and the way in which they approach their work [7]. Two ALPHAs are associated with the Endeavor: Work and Way-of-Working. In the context of software engineering, work is everything that the team does to meet the goals of producing a software system. The work is guided by the practices that make up the team's way-of-working. The Way-of-Working is the tailored set of practices and tools used by a team to guide and support their work, which evolves according to specific working contexts [7].

In particular, although the Endeavor area of concern addresses the practitioners' practical knowledge, it does not indicate how to express it nor how to reason about it. This lack of expressive mechanisms stimulated the extension of ESSENCE in the form of KUALI-BEH as an alternative to allow practitioners to transform their tacit knowledge into explicit knowledge. KUALI-BEH covers the top two layers of the ESSENCE architecture, which are Methods and Practices, and offers mechanisms to identify, express, agree, execute, optimize and consolidate ways of working. Consequently, KUALI-BEH became an extension of the ESSENCE Kernel, presented as a Normative Annex of the formal specification.

2.2 KUALI-BEH

KUALI-BEH is based on a set of common concepts involved in software projects and provides a framework for authoring Software Engineering Methods. KUALI-BEH is composed of two views: static and operational.

The static view defines the common concepts needed for the definition of the practitioners' diverse ways of working (see Figure 2), and arranges them into methods composed of practices. This knowledge creates an infrastructure of methods and practices that is built and used by practitioners.

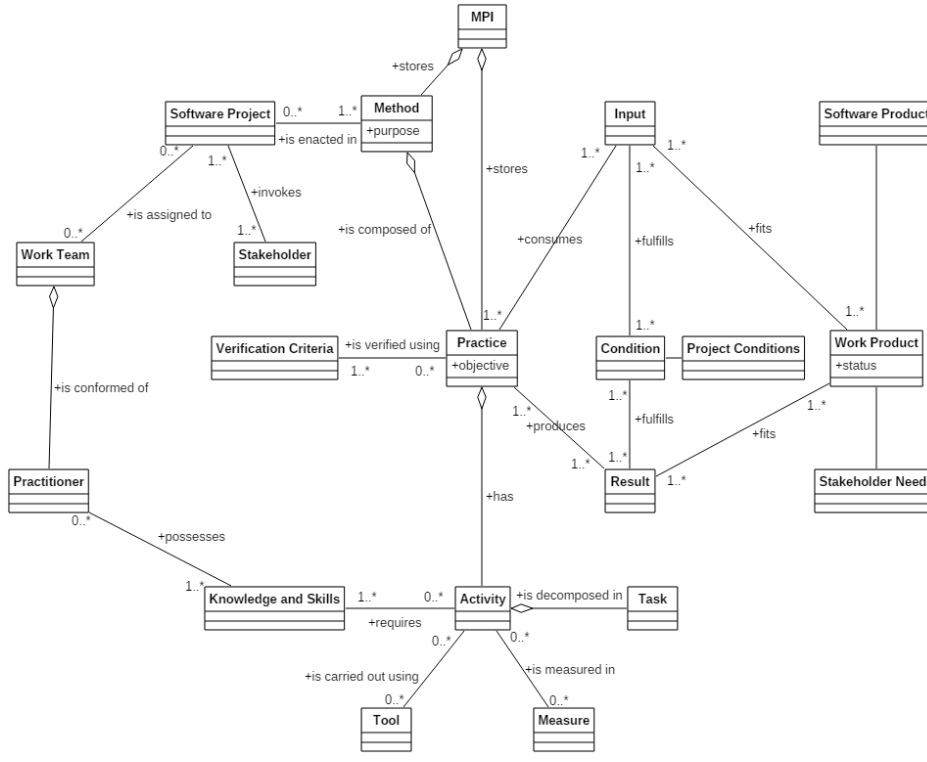


Figure 2. KUALI-BEH concepts and their relationships and attributes

The operational view is related to the software project execution. It provides work teams with mechanisms how to enact a method and adapt its practices to a specific context and stakeholder needs.

The KUALI-BEH static view target audience represents Software Engineering practitioners, who will be able to express their actual ways of working using KUALI-BEH to author methods and practices. The knowledge produced by practitioners should be validated and approved before being accumulated and shared, both inside and outside the organization.

During the process of authoring of methods and practices KUALI-BEH advises practitioners with regard to certain attributes. The set of practices that comprise a method should preserve the properties of coherency, consistency and sufficiency [6], which are formally defined in Section 3.2.3.

Having expressed methods and practices, practitioners can identify the method to be followed during software project execution. Due to the fact that every project is different, the work team will need to adapt the selected method, for which KUALI-BEH defines the following adaptations operations [6]:

1. substitution,
2. concatenation,
3. combination and
4. splitting.

These operations are explained in Section 3.2.4.

To carry out a formalization process of the software project common concepts, method properties and adaptation operations presented in KUALI-BEH, we used ontologies for the definition of common concepts, SME and Set Theory to state method properties and adaptation operations and DL for knowledge representation. They are described in more detail in the next subsections.

2.3 Ontologies

According to [10] an important challenge faced by current communities of researchers and practitioners in the field of Software Engineering and Technology is the lack of explicit knowledge shared among members of a group/project, with other groups and with other stakeholders.

The ambiguity of natural language implies potential mistakes and nonproductive efforts. Ontologies can mitigate these problems and, furthermore, some authors have intended to use ontologies as the back-bone of software tools and environments [10].

An ontology, defined by [5], is a data model that represents a set of concepts within a domain and relationships between those concepts, and it is used to reason about objects within that domain.

In [11] the main usages of ontologies in Software Engineering are

1. to clarify the knowledge structure,
2. to reduce conceptual and terminological ambiguity, and
3. to allow the sharing of knowledge.

Ontologies are meant to conceptualize; in the words of [11], conceptualization is understood to be "an abstract and simplified version of the world to be represented: a representation of knowledge based on objects, concepts and entities existing within the studied area, as well as the relationships existing among them". In [12] the authors have also considered it important to enrich this definition with the requirements of being formalized so that a machine can process it, and being shared, where the acquired knowledge is the consensus of a community of experts.

Using ontologies brings the following benefits:

1. they can be checked for inconsistencies;
2. reasoning can help to detect derived relationships or implicit class memberships;
3. errors can be removed, thus improving the quality of a knowledge base;
4. ontologies can be imported and shared [5].

Implementation of ontologies in Software Engineering in order to understand a specific field of knowledge is quite broad. For example, engineering of the ontology for the Software Engineering Body of Knowledge [13], software development methodologies and endeavours [14], software maintenance ontology [15], software measurement [16], an ontological approach to the SQL:2003 [17].

As we mentioned before, ontologies represent knowledge items in the form of concepts, relationships and attributes, which must be expressed as statements. There are different formats and languages to represent statements, e.g. Resource Description Framework (RDF) [18] or Web Ontology Language (OWL) [19].

RDF is a general-purpose language for representing and referencing information on the Web, and is intended for situations in which this information needs to be processed by applications rather than be presented to people directly [5]. RDF represents simple statements as a graph of resources, their properties and values. Based on [5] an RDF statement is composed of three elements:

1. Subject that is someone or something considered as a resource, it may be any person or item represented by a Uniform Resource Identifier (URI);
2. Predicate that indicates the subject's relation to another concept or the subject's activity; and
3. Object that defines what the subject is related to or what the subject is doing.

As for OWL, it is a markup language built on RDF and is used to publish and share ontologies on the web [5].

The usage of standardized languages like RDF and OWL helps to define and share ontologies. However, an important part of ontologies is the possibility of generating new knowledge, which is called reasoning. A tool that supports applying logic, querying and reasoning with ontologies is called a reasoner. Examples of reasoners are RACER[1], HermiT[2], FaCT++[3].

It is important to mention that most of the tools that support management of ontologies comprise more than one module, one of which is a reasoner. For the purposes of this formalization HermiT was chosen as a reasoner.

[1] `http://www.ifis.uni-luebeck.de/~moeller/racer/`

[2] `http://hermit-reasoner.com/`

[3] `http://owl.man.ac.uk/factplusplus/`

2.4 Situational Method Engineering

SME focuses on configuration of system development methods tuned to the situation of a project at hand [20]. SME aims to support software engineering by providing means for appropriate method engineering. That includes all aspects of creating, using and adapting a software development method based on local conditions, it is focused on formalising the use of methods for systems development [21]. Any coherent product, activity, or tool being part of an existing generic or situational method is a method fragment [20]. In other words, method fragments are building blocks of a situational method. It is important to mention that, in SME, a formalized method is usually an ideal type created as an abstraction of existing 'good practices' [22].

According to [23], SME contributes to reach the requirements of flexibility, experience accumulation, integration and communication, and quality. By flexibility it is assumed that the method to be used in a certain development project is situational, that is, completely tuned to the project situation at hand. The controlled adaptability of the method allows for the addition and accumulation of the project experience. All methods are based on one common repository, in which the building blocks of methods are also stored; this contributes to integration and communication. The fact that flexibility should be controlled guarantees that the constructed situational method meets the same quality requirements as standard methods.

The relevance of these requirements is present in the current research; in a recent study [21] discusses issues like tailoring a constructed method in order to apply it to a particular context (see flexibility), comparing chunks, fragments and components, and creating a methodology from them. Besides, the author addresses such questions as how to consider methods as action knowledge (see experience accumulation), and how to assess the quality of the method parts, the constructed method and its effectiveness in practice (see quality).

An initial formalization of method fragments is developed by [23] and organized in four groups: Sets, Predicates, Functions and Rules. This particular approach was used to formalize KUALI-BEH.

2.5 Description Logics

DL [24] is a family of knowledge representation (KR) formalisms that represents the knowledge of an application domain [25]. It is a popular formalism for ontologies and is regarded as the foundation of some ontology languages due to the fact that ontologies contain knowledge about a domain in a precise and unambiguous manner [1].

DL has been shown as common language for ontologies appearing in a Software Engineering process that needs support from Knowledge Engineering and is the natural successor in terms of evolution of UML [26]. The basic DL family is the $\mathcal{AL}$-languages [27], and it follows the syntax rule of:

$$C, D \rightarrow A \mid \top \mid \bot \mid \neg A \mid C \sqcap D \mid \forall R.C \mid \exists R.\top.$$

It is possible to create statements and build a knowledge base using this language. A knowledge base in DL comprises two areas: the Terminological Knowledge (TBox) and the Assertional Knowledge (ABox).

The TBox is a collection of concepts and roles. Concepts represent the entities of a *"universe"*, while Roles denote the relations (properties or associations) between these concepts. In other words, the TBox introduces vocabulary of a specific domain. The basic form of a declaration in a TBox is a concept definition, that is, the definition of a new concept in terms of other previously defined concepts. For example, in the context of KUALI-BEH we can define a Practitioner as a Person who is also a Software Engineer, so in DL this concept acquires the following representation:

$$\text{Practitioner} \equiv \text{Person} \sqcap \text{SoftwareEngineer}.$$

On the other hand, the ABox contains assertions about named individuals in terms of this vocabulary, that is, it contains extensional knowledge about the domain of interest. For example:

$$\text{Person}(\text{“Miguel”}) \sqcap \text{SoftwareEngineer}.$$

It states that the individual “Miguel” is a Person and also a Software Engineer. Given the above example of a Tbox, we obtain the assertion that *Miguel is a Practitioner*, which now belongs to the ABox.

There are some restrictions when working with DL and according to [25] the most important are that:

1. only one definition for a concept name is allowed, and
2. definitions are acyclic in the sense that concepts are neither defined in terms of themselves nor in terms of other concepts that indirectly refer to them.

At this point we can observe that Person and Software Engineer are atomic concepts, but DL offers the possibility of building complex descriptions inductively using concept constructors [25]. By adding more constructors to $\mathcal{AL}$ we obtain more expressive languages. Table 1 shows the spectrum of $\mathcal{AL}$ families.

Family	Added Constructor	Expression
$\mathcal{U}$	Union	$C \sqcup D$
$\mathcal{E}$	Full existential quantification	$\exists R.C$
$\mathcal{N}$	At most and at least restrictions	$\leq n$ R, $\geq n$ R
$\mathcal{C}$	Negation	$\neg C$
$\mathcal{O}$	One of	$a_1, \ldots, a_n$
$\mathcal{I}$	Inverse relation	$P, Q, R \rightarrow \text{R}^-$
$\mathcal{Q}$	Qualified number restriction	$\leq nR.C$, $\geq nR.C$
$\mathcal{R}$	Complex role inclusion	$P \circ Q \subseteq \text{R}$

Table 1. DL family of languages, adapted from [26]

Due to the fact that DL belongs to a KR formalism and to the assumption that a KR system always answers user's queries in a reasonable time, the reasoning and decision procedures of DL are its strength [25]. Besides, a knowledge representation system based on DL and derived from a KR formalism assures four main elements: the TBox and the ABox, together with a reasoner (Inference System) and a user interface (Interface), see Figure 3.

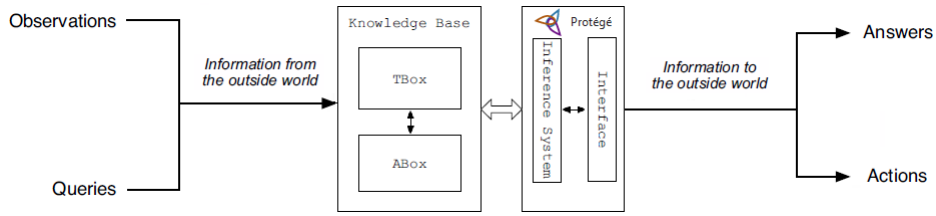


Figure 3. Architecture of a system based on DL, adapted from [1] and [2]

3 KUALI-BEH FORMALIZATION

This section presents the KUALI-BEH formalization, which is divided into three parts: the ontology of common concepts; the representation of method properties and adaptation operations using SME; and the representation of the ontology through DL.

3.1 KUALI-BEH Language

KUALI-BEH language is an initial approach to share a common representation of knowledge as a set of concepts, attributes and relationships of a domain in the form of ontology, abbreviated as KB-O. This ontology offers method engineers the means how to describe, analyze and reason about software projects and information related to them.

Due to its simplicity and the fact that it was created specifically for Software Engineering, Representation Formalism for Software Engineering Ontologies (REFSENO) [28] was chosen to define the KB-O ontology. REFSENO provides constructs to define concepts with their attributes and relationships between them. The construction of REFSENO ontologies is based on three tables that use text and, optionally, diagrams and contain a glossary of concepts, attributes and relationships, respectively. REFSENO allows definition of cardinalities for the relationships and value ranges for the attributes.

The specification of an ontology should contain the modeled domain, the purpose of the ontology, the scope, and administrative information like the authors and knowledge sources [28]. Table 2 displays the KB-O requirements specification. Below

is the KB-O definition based on the general background and the KB-O requirements specification mentioned above.

Domain	Software Projects
Last modified date	January 14, 2016 (updated)
Conceptualized by	KUALI-KAANS Research Group
Purpose	To describe the common concepts involved in software projects and their relationships
Level of formality	Semi-formal (UML Diagrams, text and REFSENO tables)

Table 2. KB-O requirements specification

3.1.1 Definition of KB-O

After establishing the KB-O requirements specification, we carried on with the development of the ontology itself, using the suggested by REFSENO process model and a Unified Modeling Language (UML) [29] Class diagram. Note that for the purpose of this paper, a reduced version of REFSENO is presented in order to maintain it readable and easy to assimilate.

The resulting ontology consists of a graphical representation, a UML class diagram, and a textual semi-formal representation of knowledge using REFSENO. Figure 2 shows the corresponding UML class diagram.

3.1.2 Concepts Glossary

The concepts glossary lists alphabetically all the concepts of the ontology. One row of the concepts glossary corresponds to one concept. The columns are labeled Name, Definition and Example, denoting the respective components of the concept definition. REFSENO requires an extra column named References; however, in this case it was omitted and, instead, a reference list with all the sources considered to create the respective concept definition is reported in [6].

Table 3 displays a fragment of the glossary from the KUALI-BEH ontology[4], showing the specific concepts used in this paper in order to illustrate the proposed formalization. The full version can be consulted in [6].

3.1.3 Relationships

Relationships model the way in which a particular software engineering entity is related to other software engineering entities and are labeled as follows: Name, Con-

[4] This definitions were created in the context of ESSENCE standardization process. ISO/IEC 24744:2007 (now 2014) was not considered because of important differences between both efforts, for example in the clabject and powertype concepts. As future work we consider to make a comparison between the concepts of both standards.

Name	Definition	Example
Method	A method is a composition of a coherent, consistent and sufficient set of practices, with a specific purpose that fulfills the stakeholder needs under specific conditions.	Software Implementation
MPI	The methods and practices infrastructure (MPI) is a set of methods and practices learned by the organization members by experience, abstraction or apprehension. This base of knowledge is continuously expanded and modified by the practitioners.	Organizational Base of Knowledge
Practice	A practice is work guidance, with a specific objective, that advises how to produce a result originated from an input. This guide provides a systematic and repeatable set of activities focused on the achievement of the practice objective and result.	Software Requirements Analysis
Practitioner	A practitioner is a professional in Software Engineering that is actively engaged in the discipline. The practitioner should have the ability to make a judgment based on his or her experience and knowledge.	Miguel

Table 3. KB-O concepts glossary

cepts (cardinality) and Description. The relationships of this ontology are equivalent to the non-terminal concept attributes defined in REFSENO. Table 4 shows a subset of the 29 relationships of KB-O.

Name	Concepts (Cardinality)	Description
Consumes	Practice (*) – Input (*)	A practice consumes an input.
Is assigned to	Work Team (*) – Software Project (*)	A work team is assigned to a software project.
Is composed of	Method (*) – Practice (*)	A method is composed of practices.
Is formed of	Work Team (*) – Practitioners (*)	A work team is formed of practitioners.
Produces	Practice (*) – Result (*)	A practice produces a result.

Table 4. KB-O relationships

3.1.4 Attributes

An attribute is represented using the concept attribute table, which is concept-specific and contains one row for every attribute. The columns are labeled as follows: Name, Description, Mandatory and Type. The attributes of this ontology are equivalent to the terminal concept attributes defined in REFSENO. Table 5 presents the attributes of KB-O.

Attribute (of Concept)	**Description**	**Mandatory**	**Type**
Objective (Practice)	Description of the goal that a practice pursues.	Yes	Text
Purpose (Method)	Description of the goal that a method pursues.	Yes	Text
Status (Work Product)	Description of the actual state or situation of a work product.	No	Text

Table 5. KB-O attributes

3.2 KUALI-BEH Algebra

Based on the ideas outlined in [20] and [23], we defined KUALI-BEH algebra (KB-A), which is a set of axioms, predicates, functions and operations to represent the KUALI-BEH method properties and operations. KB-A is defined in the next subsections.

3.2.1 KB-A Axioms and Definitions

The methods and practices infrastructure (MPI) contains all the elements that are built using the common concepts. Therefore, it is the first axiom of KB-A.

Axiom 1. Let $\mathcal{MPI}$ be all the *things* that can be created using KUALI-BEH.

$$\mathcal{MPI} = \{\mathcal{M} \oplus \mathcal{P} \oplus \mathcal{J} \oplus \mathcal{W} \oplus \mathcal{C}\}$$

where

$$\begin{aligned}\mathcal{M} &= \{m \mid m \text{ is a method}\},\\ \mathcal{P} &= \{p \mid p \text{ is a practice}\},\\ \mathcal{J} &= \{j \mid j \text{ is a software project}\},\\ \mathcal{W} &= \{w \mid w \text{ is a work product}\},\\ \mathcal{C} &= \{c \mid c \text{ is a condition}\}.\end{aligned}$$

Axiom 2. $\mathcal{W}$ and $\mathcal{C}$ are disjoint sets.

$$\mathcal{W} \cap \mathcal{C} = \emptyset.$$

Definition 1. Let $p_1, \ldots, p_n$ be practices, a method m composed of this set of practices can be expressed as $m = \{p_1, \ldots, p_n\}$.

Definition 2. The definition of any element of the KUALI-BEH static view is a conceptual definition and is denoted by the letter "c" as super-index. For example, a conceptual definition of a work product w is denoted as w^c, which is related to its characteristics.

Definition 3. The definition of any element of the KUALI-BEH operational view is a technical definition and is denoted by the letter "t" as super-index. For example, a technical definition of a work product w is denoted as w^t, which is its instance.

Definition 4. The practitioners are the only individuals who can determine whether:

1. similarity between inputs and results is held;
2. the method purpose is fully achieved; and
3. the objective of a practice supports a method purpose.

This ability corresponds to the practitioner's judgment.

3.2.2 KB-A Functions and Predicates

In this section we discuss the functions defined in the KB-A.

The objective of a practice p is obtained through the function *objective* applied to the conceptual definition of the practice.

$$\text{objective} : \mathcal{P} \longrightarrow \text{Text}.$$

In a similar way, the purpose of a method m is obtained through the function *purpose* applied to the conceptual definition of the method.

$$\text{purpose} : \mathcal{M} \longrightarrow \text{Text}.$$

In the same way, the status of a work product w is obtained through the function *status* applied to the technical definition of the work product.

$$\text{status} : \mathcal{W} \longrightarrow \text{Text}.$$

In order to decide whether the purpose of a method is fully achieved, the function *fully_achieved* is defined.

$$\text{fully\_achieved} : \mathcal{M} \longrightarrow \mathbb{B}.$$

The function *similarity* is defined in order to decide whether a work product or condition fits the characteristics required by an input or result. Comparing the

technical definition against the conceptual definition of a work product or condition, practitioners can decide on their similarity.

$$\text{similarity} : (\mathcal{W}^c \cup \mathcal{C}^c) \times (\mathcal{W}^t \cup \mathcal{C}^t) \longrightarrow \mathbb{B}.$$

Likewise, the functions *input* and *result* are defined for methods and practices. These functions receive a practice or a method and return a set of work products and/or conditions.

$$\text{input} : \mathcal{M} \cup \mathcal{P} \longrightarrow \mathcal{W} \cup \mathcal{C},$$

$$\text{result} : \mathcal{M} \cup \mathcal{P} \longrightarrow \mathcal{W} \cup \mathcal{C}.$$

The KUALI-BEH predicates are expressed in the following way:

- produces(p_i, r) denotes that a practice p_i produces a result r:

$$\text{produces} : \mathcal{P} \times (\mathcal{W} \cup \mathcal{C}),$$

- consumes(p_j, i) denotes that a practice p_j consumes an input i:

$$\text{consumes} : \mathcal{P} \times (\mathcal{W} \cup \mathcal{C}),$$

- precedes(p_i, p_j) denotes that a practice p_i precedes a practice p_j:

$$\text{precedes} : \mathcal{P} \times \mathcal{P},$$

- follows(p_j, p_k) denotes that a practice p_j follows a practice p_k:

$$\text{follows} : \mathcal{P} \times \mathcal{P},$$

- supports(p, m) denotes that the objective of a practice p supports the purpose of a method m:

$$\text{supports} : \mathcal{P}^c \times \mathcal{M}^c.$$

3.2.3 KB-A Method Properties

In order to represent method properties, as stated in Definition 1, let us define a method $m \in \mathcal{M}$ as a set:

$$m = \{p \mid p \ \in \mathcal{P}\}.$$

The coherency, consistency and sufficiency properties of a method are defined below.

Coherency. Let us define a function named *coherency*, which receives a method and returns *true* if it is coherent or *false* if it is not.

$$\text{coherency} : \mathcal{M} \longrightarrow \mathbb{B}.$$

This function **coherency(m)** is evaluated as follows:

$$\text{coherency}(m) = \begin{cases} \textit{true} & \text{if for all practices } p_i \text{ of a method } m, \text{ the objective} \\ & \text{of } p_i \text{ supports the purpose of } m, \\ \textit{false} & \text{otherwise.} \end{cases} \tag{1}$$

In other words we have:

$$\text{if } \forall p \in m, \text{supports}(\text{objective}(p^c), \text{purpose}(m^c)) = \textit{true}.$$

Consistency. Let us define the function *consistency*, which receives a method and returns *true* if it is consistent or *false* if it is not.

$$\text{consistency} : \mathcal{M} \longrightarrow \mathbb{B}.$$

This function **consistency(m)** is evaluated as follows:

$$\text{consistency}(m) = \begin{cases} \textit{true} & \text{if all the practice inputs are produced and all} \\ & \text{the practice results are consumed, except for} \\ & \text{the method input and result;} \\ \textit{false} & \text{otherwise.} \end{cases} \tag{2}$$

In other words we have:

$$\begin{aligned} \text{if } \forall p_1 \in m\ \exists p_2, p_3 \in m(&\text{produces}(p_1, r) \rightarrow \text{consumes}(p_2, r) \vee \text{result}(m^c) = r) \\ &\wedge (\text{consumes}(p_1, i) \rightarrow \text{produces}(p_3, i) \vee \text{input}(m^c) = i). \end{aligned}$$

Sufficiency. Let us define the function *sufficiency*, which receives a method and returns *true* if it is sufficient or *false* if it is not.

$$\text{sufficiency} : \mathcal{M} \longrightarrow \mathbb{B}.$$

This function **sufficiency(m)** is evaluated as follows:

$$\text{sufficiency}(m) = \begin{cases} \textit{true} & \text{if the method } m \text{ is coherent, consistent and its} \\ & \text{purpose is fully achieved,} \\ \textit{false} & \text{otherwise.} \end{cases} \tag{3}$$

In other words we have:

$$\text{if coherency}(m) \wedge \text{consistency}(m) \wedge \text{fully\_achieved}(m) = \textit{true}.$$

3.2.4 KB-A Adaptation Operations

In order to express the operations of adaptation, let us define a practice P as a triple formed by an Input (I), an Objective (O) and a Result (R)

$$P = (I, O, R).$$

The operations of substitution, concatenation, combination and splitting are defined below.

Substitution. The *substitution* of practices consists in replacing a practice by another equivalent practice.

$$\begin{gathered}\text{Let } P_1 = (I_1, O_1, R_1) \text{ and } P_2 = (I_2, O_2, R_2) \text{ be practices.}\\ P_1 \text{ can be } \textit{substituted} \text{ by } P_2 \text{ if and only if}\\ P_1 \equiv P_2.\end{gathered}$$

Notice that similarity is recognized and dictated by the practitioner's judgment. After applying the adaptation operation the original properties of a method are preserved, since the new practice holds an objective, input and result similar to the substituted practice.

Concatenation. If one practice has a result similar to the input of another practice, both can be integrated into one practice, applying the *concatenation* operation, which is defined as follows:

$$\begin{gathered}\text{Let } P_1 = (I_1, O_1, R_1) \text{ and } P_2 = (I_2, O_2, R_2) \text{ be practices and } R_1 \text{ is similar to } I_2.\\ \text{A practice } P_3 \text{ is a correct } \textit{concatenation} \text{ of the practices } P_1 \text{ and } P_2 \text{ if}\\ P_3 = (I_1, O_1 \wedge O_2, R_2).\end{gathered}$$

Combination. A *combination* of practices consists of bringing two different practices into one and is defined as follows:

$$\begin{gathered}\text{Let } P_1 = (I_1, O_1, R_1) \text{ and } P_2 = (I_2, O_2, R_2) \text{ be practices.}\\ P = (I, O, R) \text{ is a correct } \textit{combination} \text{ of } P_1 \text{ and } P_2 \text{ if}\\ I \text{ is similar to } I_1 \cup I_2 \text{ and}\\ R \text{ is similar to } R_1 \cup R_2 \text{ and}\\ O \equiv O_1 \wedge O_2.\end{gathered}$$

Splitting. A *splitting* of practices consists in the partition of the original practice into two different practices preserving the original objective and similar inputs and results. Formally, the splitting operation is defined as follows:

$$\text{Let } P_1 = (I_1, O_1, R_1) \text{ and } P_2 = (I_2, O_2, R_2) \text{ be practices.}$$
$$P_1 \text{ and } P_2 \text{ are a correct } \textit{split} \text{ of } P = (I, O, R) \text{ if}$$
$$I_1 \cup I_2 \text{ is similar to } I \text{ and}$$
$$R_1 \cup R_2 \text{ is similar to } R \text{ and}$$
$$O_1 \wedge O_2 \equiv O.$$

Strictly following these rules while applying the adaptation operations to practices assures the preservation of the original properties of coherency, consistency and sufficiency of a method.

3.3 KUALI-BEH Knowledge

In this section we discuss KB-K, short for the knowledge representation of KUALI-BEH using DL and based on KB-O. To make a clear picture of the process of creating KB-K, we will provide some examples. Let us consider the concept practitioner defined in Table 3 of KB-O. This concept can be defined using sets as:

$$\{x \mid \text{Practitioner}(x)\}.$$

To express roles we proceed in a similar way. Let us consider the relationship *isFormedOf* presented in Table 4. This relationship expresses that a work team is formed of practitioners, and it can be denoted as follows:

$$\{(x, y) \mid \text{isFormedOf}(x, y)\}.$$

Now let us transform these sets into First Order Logic (FOL) predicates, which is a common transformation process of semantic networks and ontologies. Let us assume that every work team in our organization must be formed of practitioners. This assertion can be rewritten in FOL as:

$$\forall x.\text{WorkTeam}(x) \rightarrow \exists y.\text{isFormedOf}(x, y) \wedge \text{Practitioner}(y).$$

Then, this predicate can be rewritten again, but now in DL:

$$\text{WorkTeam} \sqsubseteq \forall \text{isFormedOf.Practitioner}.$$

In a similar way, the inverse relationship of *isFormedOf* can be represented as the relationship *belongsTo*. This expresses that each practitioner in the organization belongs to a work team, and can be represented as follows:

$$\text{Practitioner} \sqsubseteq \exists \text{belongsTo.WorkTeam}.$$

The creation process of KB-K used two DL association types:

has-part: This type can be used to denote aggregation and composition in UML [30]. Although these associations are completely different in UML, since they denote strong and weak relationships, in DL has-part can represent both of them.

is-a: This type is equivalent to the generalization in Entity-Relationship model or to inheritance in Object Oriented paradigm. It is a general association that can be interpreted as is-a-kind-of and is-an-instance-of.

For example, we identified in KUALI-BEH that an Activity consists of four elements: Knowledge and Skills, Tasks, Tools and Measures (see Figure 2). So, expressing in DL that an Activity *has-parts* we have:

$$\begin{aligned}\text{Activity} \sqsubseteq (&\exists\, \text{requires.KnowledgeAndSkills}) \wedge \\ &((\exists\, \text{isDecomposedIn.Task}) \vee \\ &(\exists\, \text{isCarriedOutUsing.Tool}) \vee \\ &(\exists\, \text{isMeasuredIn.Measure}) \vee \mathit{true}).\end{aligned}$$

Notice that only Knowledge and Skills are mandatory, while Task, Tool and Measure are not.

In KUALI-BEH the Stakeholder Needs is a specialization of a WorkProduct. Then, we can say that *StakeholderNeeds is-a WorkProduct*, which is written in DL as:

$$\text{StakeholderNeeds} \sqsubseteq \text{WorkProduct}.$$

This association is used for the three instances defined in KB-O: Stakeholder Needs, Project Conditions and Software Product.

To represent an attribute, for example, we know that a *WorkProduct* has an attribute named *status* and its datatype is String, so we have:

$$\{x \mid \text{WorkProduct}(x) \wedge (\exists s.\text{Status}(x, s) \wedge \text{String}(s))\}.$$

In DL the datatype equivalent to String is Text, so we can represent a Work Product in DL as follows:

$$\text{WorkProduct} \sqsubseteq (= 1\ (\text{status.Text})).$$

Let us examine a more general example: if we have a work product named *Database Model* and its status is *Draft*, in DL we have:

$$\begin{aligned}&\text{WorkProduct}(\text{“DatabaseModel”}) \wedge \\ &\text{Status}(\text{“DatabaseModel”}, \text{“Draft”}) \wedge \text{Text}(\text{“Draft”}).\end{aligned}$$

3.3.1 Definition of KB-K

After having transformed KB-O into DL expressions, we generated KB-K, which is defined as follows:

Method:

$$\sqsubseteq (= 1\ (\text{mpi.MPI})) \wedge (\exists\, \text{isComposedOf.Practice}) \wedge (\exists\, \text{isEnactedIn.SoftwareProject}) \wedge (= 1\ (\text{purpose.Text})).$$

MPI:

$$\sqsubseteq (\exists\, \text{stores.SoftwareProject}) \vee (\exists\, \text{stores.Method}) \vee (\exists\, \text{stores.Practice}) \vee (\exists\, \text{stores.WorkProduct}) \vee (\exists\, \text{stores.Condition}).$$

Practice:

$$\sqsubseteq (\exists\, \text{method.Method}) \wedge (\exists\, \text{consumes.Input}) \wedge (\exists\, \text{produces.Result}) \wedge (\exists\, \text{isVerifiedUsing.VerificationCriteria}) \wedge (\exists\, \text{has.Activity}) \wedge (= 1\ (\text{objective.Text})).$$

Practitioner:

$$\sqsubseteq (\exists\, \text{workTeam.WorkTeam}) \wedge (\exists\, \text{possesses.KnowledgeAndSkills}).$$

WorkTeam:

$$\sqsubseteq (\exists\, \text{isFormedOf.Practitioner}) \wedge (\exists\, \text{isAssignedTo.SoftwareProject}).$$

Due to space restrictions, only 4 definitions are presented. The full KB-K is available online on WebProtege site[5].

[5] http://webprotege.stanford.edu/#Edit:projectId=d846a165-258c-4ce9-b703-44a29dd690c8

4 VALIDATION

This section presents a KB-K proof of concept and exemplifies the KB-K usage through a method, which was expressed in a real organization during the validation stage of KUALI-BEH. At the end, we present a comparison with related work.

4.1 Proof of Concept

As it has been mentioned before, DL systems not only store terminologies and assertions, but also offer services to reason about them. Typical reasoning tasks are to determine whether a description is satisfiable (i.e., non-contradictory) [25]. In order to show the usage of KB-K, we offer a proof of concept through an example in the next subsections.

4.1.1 Source of the Example

To prove the usefulness of the proposed formalization, we need an example of an expressed way of working that would contain a description of a software project activity. In this case, the example was taken from Annex C: Case Studies and Examples of [8] and is a fragment of a typical information system delivery process done in Fujitsu DMR Macroscope modeled in SPEM:

The *Information System Delivery Process* is developed during the *Preliminary Analysis* phase and consists of only one iteration that is the *First Joint Requirements Planning Workshop*. In this iteration the *Define Owner Requirements* task is developed by the *System Architect* role in three steps: *Define objectives based on stated needs*, *Define the key issues* and *Determine the relevant enterprise principles*. Besides, one work product is required to start the task and two new items are produced, the *Enterprise Architecture*, *Assessment of Current System* and *Owner Requirements*, respectively.

The specific SPEM elements appear in bold fonts. Acronyms of each element of the process are presented in the parenthesis, this with the purpose of facilitating process representations in KB-K.

Activity {kind: Phase}: Preliminary Analysis (PA)
  Process: Information System Delivery Process (ISDP)
    Activity {kind: Iteration}: First Joint Requirements Planning Workshop (FJRPW)
      TaskUse: Define Owner Requirements (DOR)
        RoleUse: System Architect (SA)
        WorkDefinitionParameter {kind: in}
          WorkProductUse: Enterprise Architecture (EA)
        WorkDefinitionParameter {kind: out}
          WorkProductUse: Assessment of Current System (ACS) {state: initial draft}

WorkProductUse: Owner Requirements (OR)

{state: initial draft}

Steps

Step: Define objectives based on stated needs (DOBOSN)

Step: Define the key issues (DTKI)

Step: Determine the relevant enterprise principles (DREP)

At this point we have an expressed way of working, but how well is it defined? It was implied in [1] that the process was not fully completed and five inconsistencies related to two major classes of problems were pointed out. Firstly, *Phase* and *Iteration* have no performer. Since both elements are a specialization of an Activity, they must have at least one performer each. Secondly, the *WorkProductUse*'s kinds are not clear, because they are defined through the *WorkDefinitionParameter*. And it is not possible to know precisely if a specific work product is input or output.

In order to prove that KUALI-BEH solves these inconsistencies, the Fujitsu way of working was modeled using the formalization presented in this paper.

4.1.2 Example of KB-K Usage

To provide a comparison between SPEM and KUALI-BEH, a partial mapping between the KUALI-BEH concepts and the SPEM elements was done. Table 6 shows a subset of the mapping and contains only the concepts required for the purpose of the example.

SPEM	KUALI-BEH
Process	Method
Activity	Practice
TaskUse	Activity
Step	Task
RoleUse	KnowledgeAndSkills
WorkProductUse	WorkProduct

Table 6. Mapping between KUALI-BEH and SPEM concepts

This research followed the idea of [31] who propose the use of ontologies for the evaluation of metamodels. In their study, they create a reference ontology, i.e., an ontology of method in general against which they can then compare any given 'branded' method or SME approach [21].

Based on the mapping (Table 6), we obtain the following expressions, which are equivalent to the process fragment presented in the previous subsection:

$$\begin{gathered}\text{Method}(\text{“ISDP”}) \wedge \text{isComposedOf}(\text{“ISDP”}, \text{“FJRPW”}) \wedge \\ \text{Practice}(\text{“FJRPW”}) \wedge \text{has}(\text{“FJRPW”}, \text{“DOR”}) \wedge \\ \text{Activity}(\text{“DOR”}) \wedge \text{requires}(\text{“DOR”}, \text{“SA”}) \wedge \\ \text{KnowledgeAndSkills}(\text{“SA”}) \wedge \text{isDecomposedIn}(\text{“DOR”}, \text{“DOBOSN”}) \wedge \\ \text{isDecomposedIn}(\text{“DOR”}, \text{“DTKI”}) \wedge \text{isDecomposedIn}(\text{“DOR”}, \text{“DREP”}) \wedge \\ \text{Task}(\text{“DOBOSN”}) \wedge \text{Task}(\text{“DTKI”}) \wedge \text{Task}(\text{“DREP”}) \wedge \\ \text{input}(\text{“DOR”}, \text{“EA”}) \wedge \text{WorkProduct}(\text{“EA”}) \wedge \\ \text{Status}(\text{“EA”}, \text{“initial draft”}) \wedge \text{Text}(\text{“initial draft”}) \wedge \\ \text{result}(\text{“DOR”}, \text{“ACS”}) \wedge \text{WorkProduct}(\text{“ACS”}) \wedge \\ \text{Status}(\text{“ACS”}, \text{“initial draft”}) \wedge \text{Text}(\text{“initial draft”}) \wedge \\ \text{result}(\text{“DOR”}, \text{“OR”}) \wedge \text{WorkProduct}(\text{“OR”}).\end{gathered}$$

It is worth mentioning that using Protégé (version 5.0.0) and HermiT (version 1.3.8.413) we demonstrated the satisfiability of this example. Figure 4 shows the resulting KB-K graph generated using Protégé.

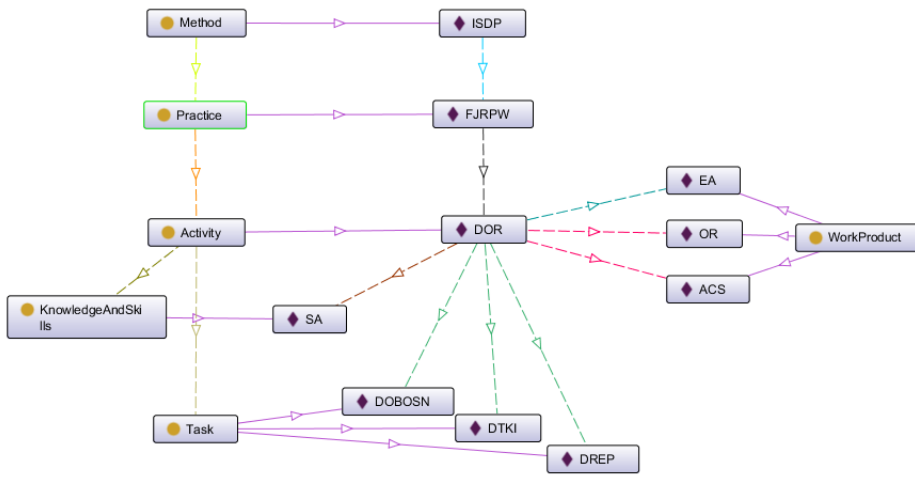


Figure 4. KB-K inferred model

It can be noticed that DOR has a performer (SA) and each of the work products associated with DOR can be differentiated as inputs (blue arrow) or results (pink arrows).

4.1.3 Results of the Example

It was pointed out by Wang [1] that direct reasoning about models built using metamodels like SPEM is difficult and it is hard to keep these models consistent, thus making SPEM to be not the best choice when analyzing models. Nevertheless, we found that KUALI-BEH is capable of representing ways of working through a clear model that preserves its structure [32].

Our example has demonstrated how the SPEM inconsistencies found by [1] can be corrected. These inconsistencies are solved by KUALI-BEH because

1. KUALI-BEH has a well-defined hierarchical approach, and
2. KUALI-BEH does not have reflexive associations.

Since KUALI-BEH's structure is a tree, the dependencies of its elements can be hierarchically stated. As a consequence, the practitioners' ways of working are modeled as a fixed structure avoiding ambiguities and allowing a uniform interpretation of data.

Importantly, KUALI-BEH has no cyclical dependencies and no element can be defined recursively, which is a major drawback in SPEM. For example, in the above described Fujitsu process, an *Activity* can represent either a process or an iteration using the stereotype *kind*; however, the elements that form an activity remain the same regardless of the fact that different kinds of objects are modeled. In fact, this constitutes the main cause of inconsistencies found in [1].

Last but not least, the coherency, consistency and sufficiency properties, defined in KUALI-BEH, can be evaluated and are meant to decide if the expressed way of working is well or ill-formed, thus giving practitioners an initial means of verifying and improving their own ways of working.

4.2 Examples Obtained from Case Studies

During the validation of KUALI-BEH, specifically during case studies, we identified situations where the formalization was used in order to improve practices and methods created by participants. It is important to mention that the validation focus was not the formalization per se, however, the case study participants naturally used properties and operations to improve their practices and methods.

The first appearance of a formalization element occurred when practitioners of case study 1, reported in [32], evaluated the consistency of its method, called IB. In particular, one practice produced two results that consequently became input for another practice. The problem was that the practice did not consume one of the results. This behavior occurred because the practices were executed by different practitioners. After having expressed explicitly the whole method, both practitioners realized that there exists a practice which result (a subset of it) is not similar to the input of the rest of the method's practices. In other words, the practice was

producing a result that nobody else consumed:

$$\exists p_1 \in IB(\text{produces}(p_1, r) \text{ but } \nexists p_i \in IB \text{ consumes}(p_i, r).$$

Therefore:

$$\text{consistency}(IB) = \textit{false}.$$

Another instance of the formalization occurred during the case study 3. At a particular moment of method authoring, the participants decided to adapt the defined method using adaptation operations. On the one hand, they needed to split a practice in order to make it easier to execute and distribute the work. On the other hand, they wanted to merge four practices having similar objectives and create a more generic practice that could be applied in different contexts. When this was done empirically, we evaluated the rules defined by the adaptation operations against the method properties confirming that the latter were totally preserved.

4.3 Comparison with Related Work

Method engineering approaches offer guidance to express methods and methods fragments with the purpose of formalizing knowledge and tailoring methods to particular situations. Aharoni [33] identifies four approaches: the OPEN Process Framework (OPF) [34], the assembly-based SME approach [35], the scenario-based approach [36] and the application-based approach [37].

These fragment representation approaches were evaluated by Aharoni in terms of expressiveness, consistency, formalism and comprehensibility. He reports the lack of comprehensibility of the obtained representations where only 1 of 4 approached supports assembly operations and formalisms. Besides, all the four approaches rely on visual semi-formal languages. There is a special need to provide a formal representation and to define adequate operations to manipulate methods.

On the other hand, these approaches require the means to represent methods or processes, i.e. a process modeling language (PML). For example, an extensively used PML in software engineering is UML. In [38] several other PMLs, such as Petri nets, Business Process Model and Notation (BPMN) [39] and SPEM are mentioned.

Analyzing the variety of alternatives to modelling methodologies and hence many aspects of SME, it is generally agreed that there are at least three core elements: producer, work unit and work product [21]. SPEM and ISO/IEC 24744 [40] follow this line and, together with BPMN and UML, are the most representative alternatives for modeling processes.

However, some drawbacks can be identified. Firstly, SPEM, which is also a OMG standard specially created for Software Engineering methods, is perceived as very complex and hard to tackle making it difficult to learn [41]. Second, despite formalism and structure advantages of the ISO/IEC 24744, the introduced concepts are not only unfamiliar to practitioners, but also are distanced from their context [42]. Finally, as reported by [43], the important drawbacks of UML, BPMN

and other PMLs remain deficiency of evolution, evaluation and human decision support.

KUALI-BEH formalization supports a comprehensive representation of knowledge and permits to reason about created methods. On the one hand, it is based on a simpler set of concepts (KB-O), allows a formal representation and manipulation of methods (KB-A) and provides a method rationale mechanism (KB-K). On the other hand, the KUALI-BEH concepts are compatible with alternatives used in the software engineering community, e.g. SPEM, BPMN or UML. Therefore, any process that is modelled with these alternatives can be formalized through KUALI-BEH taking advantage of its simplicity, common language and method reasoning.

5 INTENDED USAGE

According to [44], the main objectives of formalizations are to improve the communication, computational inference and reuse of knowledge. As a whole, the formalization of KUALI-BEH (KB-K, KB-O and KB-A) is a starting point to motivate and improve these aspects, giving practitioners and organizations a viable alternative to structure their wide tacit knowledge.

KB-O offers a unified vocabulary (terms and definitions) and improves communication among humans, consequently discussions and agreements over a specific domain are possible.

Applying KB-A, its rules and properties, we can manipulate knowledge in a uniform and standardized manner and achieve communication between humans and computers.

With KB-K, comprising a knowledge base and inference rules, the communication between software tools will foster the exchange and analysis of data. Besides, providing a structure and a management mechanism to the knowledge involved in software projects, we will be able to make computational inferences. Moreover, practitioners will have means to evaluate and enrich their knowledge.

It is important to keep in mind that the target audience of KUALI-BEH formalization is process engineers; this formalization aims at providing them with the means of description, analysis and reasoning about software projects.

According to [45], an advantage for knowledge representation is the coupling between theory and practice. In fact, KUALI-BEH was born as a proposal to bridge the gap between theory and practice by structuring and reasoning about practitioners' knowledge and enriching the Software Engineering body of knowledge. It has also been validated by practitioners directly involved in software developer organizations with solid results [32], which has encouraged its formalization.

Finally, the main intended usage of the formalization is to establish the basis for creating a Computer-Aided Method Engineering (CAME) tool. There is a necessity for creating a tool that would fulfill the needs defined by [23] or [46], and which were

never completely achieved, based on the words of [47]: "Unfortunately none of these tools can express the process part of a method and support the enactment of the method process mode". We hope that KB-K, -A, and -O will become the basis of a tool that really helps practitioners to carry out the processes of method authoring and enactment.

The tool, named KB-Tool, is currently being developed. We already released two modules: the first module expresses and shares ways of working using KB-O; the second module uses KB-A to determine the accomplishment of method properties. The module that integrates KB-K with the reasoning process is still a prototype and is under development. The idea is to go from manual to systematic (semi-automatic) and then to automatic approaches [9]. As it was mentioned before, the proof of concept used the functionalities provided by Protégé.

6 CONCLUSIONS AND FUTURE WORK

KUALI-BEH formalization preserves the foundations of SME. On the one hand it defines the common concepts required to express the practitioners' ways of working by taking advantage of KUALI-BEH ontology. On the other hand, it permits the adaptation of its elements, practices and methods, in a controlled manner using the KUALI-BEH adaptation operations and properties defined as a set of axioms, definitions, predicates and functions. Following these rules it is possible to customize, modify and assemble complex elements from other elements.

Moreover, the hierarchical organization of KUALI-BEH common concepts allows for the consistency on the granularity level of its elements, no matter if it is analyzed separately or as a part of a whole.

Finally, through DL, a knowledge representation formalism, which is able to capture virtually almost all class-based representation formalisms used in Artificial Intelligence, Software Engineering, and Databases [45], we can improve communication among humans and machines, allow for computational inference and promote the reuse of knowledge.

We can conclude that with KUALI-BEH formalization it is possible to build a knowledge base with the following characteristics:

1. it has the necessary knowledge (completeness);
2. the knowledge is reliable to the real world (correctness);
3. the knowledge is not self-contradictory (consistency); and
4. the system has efficient algorithms to perform inferences (competence), which, according to [2], are the stringent requirements for a knowledge base.

Besides, as stated in [48], the practical usefulness of a formal semantics for a language is that it provides a rigorous standard that can be used to judge the correctness of an implementation, in our case the correct forming of ways of working.

This work is at the initial stage, and the DL representation of KUALI-BEH is the first step to provide method engineers with an alternative to reason directly about the actual knowledge, which is expressed by practitioners themselves, so several lines arise as future work. In the first place our research group will work on:

1. enhancing the robustness and completeness of KB-A;
2. proving the usefulness and applicability of KB-K applying it to more real life cases and ways of working;
3. making the formalization available to method engineers for feedback and improvements; and
4. use existing theory expression methods, like Petri nets or fuzzy logic, to capture and check inconsistencies of expressed ways of working.

Second, but not less important is to develop and release a fully functional technological environment, which will motivate more practitioners to use KUALI-BEH and will result in the spread of the proposal and its formalization.

Acknowledgment

The authors thank Pascual Julian Iranzo, Ph.D. for his expert advice in Description Logics.

This work has been funded by the Postdoctoral Fellowships Program of the General Directorate of the Academic Staff (DGAPA-UNAM) and the PAPIIT project IN113013 (DGAPA-UNAM); the Graduate Science and Engineering Computing (PCIC-UNAM) and CONACYT (Mexico).

REFERENCES

[1] Wang, S.—Jin, L.—Jin, C.: Represent Software Process Engineering Metamodel in Description Logic. Proceedings of World Academy of Science, Engineering and Technology, Vol. 11, 2006, pp. 109–113.

[2] Devanbu, P. T.—Jones, M. A.: The Use of Description Logics in KBSE Systems: Experience Report. Proceedings of the 16th International Conference on Software Engineering (ICSE '94), Los Alamitos, CA, USA. IEEE Computer Society Press, 1992, pp. 23–35.

[3] Edwards, J.: Managing Software Engineers and Their Knowledge. In: Aurum, A., Jeffery, R., Wohlin, C., Handzic, M. (Eds.): Managing Software Engineering Knowledge. Springer, Berlin, Heidelberg, 2003, pp. 5–27, doi: 10.1007/978-3-662-05129-0_1.

[4] Selfridge, P. G.—Hoebel, L. J.—White, D. A.: The Sixth Annual Knowledge-Based Software Engineering Conference (KBSE-91). SIGART Bulletin, Vol. 3, 1992, No. 1, pp. 33–35, doi: 10.1145/130836.130839.

[5] Schneider, K.: Experience and Knowledge Management in Software Engineering. Springer-Verlag, Berlin, Heidelberg, 2009, doi: 10.1007/978-3-540-95880-2.

[6] Morales-Trujillo, M.—Oktaba, H.: KUALI-BEH Software Project Common Concepts. Technical Report, Object Management Group, Needham, MA, USA, 2012.

[7] OMG. ESSENCE – Kernel and Language for Software Engineering Methods. Version 1.0, Formal/2014-11-02, Object Management Group, Needham, MA, USA, 2014.

[8] OMG. Software and Systems Process Engineering Metamodel (SPEM). Version 2.0, Formal/2008-04-01, Object Management Group, Needham, MA, USA, 2008.

[9] Masalagiu, C.—Chin, W.-N.—Andrei, Ş.—Alaiba, V.: A Rigorous Methodology for Specification and Verification of Business Processes. Formal Aspects of Computing, Vol. 21, 2009, No. 5, pp. 495–510, doi: 10.1007/s00165-009-0106-y.

[10] Calero, C.—Ruiz, F.—Piattini, M. (Eds.): Ontologies for Software Engineering and Software Technology. Springer-Verlag, Berlin, Heidelberg, 2006, doi: 10.1007/3-540-34518-3.

[11] Ruiz, F.—Hilera, J.: Using Ontologies in Software Engineering and Technology. In: Calero, C., Ruiz, F., Piattini, M. (Eds.): Ontologies for Software Engineering and Software Technology. Springer, Berlin, Heidelberg, 2006, pp. 62–119.

[12] Gómez-Pérez, A.—Fernández-López, M.—Corcho, O.: Ontological Engineering. Springer-Verlag, London, 2004.

[13] Abran, A.—Cuadrado, J.—García-Barriocanal, E.—Mendes, O.—Sánchez-Alonso, S.—Sicilia, M.: Engineering the Ontology for the SWEBOK: Issues and Techniques. In: Calero, C., Ruiz, F., Piattini, M. (Eds.): Ontologies for Software Engineering and Software Technology. Springer, Berlin, Heidelberg, 2006, pp. 120–138.

[14] González-Pérez, C.—Henderson-Sellers, B.: An Ontology for Software Development Methodologies and Endeavours. In: Calero, C., Ruiz, F., Piattini, M. (Eds.): Ontologies for Software Engineering and Software Technology. Springer, Berlin, Heidelberg, 2006, pp. 139–168.

[15] Dias, M.—Anquetil, N.—De Oliveira, K.: Organizing the Knowledge Used in Software Maintenance. Journal of Universal Computer Science, Vol. 9, 2003, No. 7, pp. 641–658.

[16] García, F.—Bertoa, M. F.—Calero, C.—Vallecillo, A.—Ruíz, F.—Piattini, M.—Genero, M.: Towards a Consistent Terminology for Software Measurement. Information and Software Technology, Vol. 48, 2006, No. 8, pp. 631–644.

[17] Calero, C.—Ruiz, F.—Baroni, A.—Brito, F.—Piattini, M.: An Ontological Approach to Describe the SQL:2003 Object-Relational Features. Computer Standards and Interfaces, Vol. 28, 2006, No. 6, pp. 695–713, doi: 10.1016/j.csi.2005.09.002.

[18] W3C. Web Ontology Language. Standard, World Wide Web Consortium, Cambridge, MA, 2012.

[19] W3C. Resource Description Framework. Standard, World Wide Web Consortium, Cambridge, MA, 2014.

[20] Harmsen, F.—Brinkkemper, S.: Design and Implementation of a Method Base Management System for a Situational CASE Environment. Proceedings of the Asia Pacific Software Engineering Conference, 1995, doi: 10.1109/APSEC.1995.496992.

[21] Henderson-Sellers, B.—Ralyté, J.—Agerfalk, P.–Rossi, M.: Situational Method Engineering. Springer-Verlag, 2014. ISBN 978-3-642-41466-4, doi: 10.1007/978-3-642-41467-1.

[22] Agerfalk, P.—Ahlgren, K.: Modelling the Rationale of Methods. In: Khosrowpour, M. (Ed.): Managing Information Technology Resources in Organizations in the Next Millennium. Proceedings of the 10th Information Resources Management Association International Conference. IDEA Group, Hershey, PA, 1999, pp. 184–190.

[23] Harmsen, F.—Brinkkemper, S.—Oei, H.: Situational Method Engineering for Information System Project Approaches. In: Verrijn Stuart, A. A., Olle, T. W. (Eds.): Methods and Associated Tools for the Information Systems Life Cycle. Proceedings of the IFIP WG 8.1 Working Conference, Maastricht, Netherlands, September 1994. IFIP Transactions A-55, North-Holland, 1994, pp. 169–194. ISBN 0-444-82074-4.

[24] Baader, F.—Calvanese, D.—McGuinness, D. L.—Nardi, D.—Patel-Schneider, P. F. (Eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, New York, NY, USA, 2003.

[25] Baader, F.—Nutt, W.: Basic Description Logics. In: Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., Patel-Schneider, P. F. (Eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, 2003, pp. 47–100.

[26] Kaplanski, P.: Description Logic as a Common Software Engineering Artifacts Language. 1st International Conference on Information Technology (IT 2008), IEEE, 2008, pp. 1–4.

[27] Schmidt-Schauss, M.—Smolka, G.: Attributive Concept Descriptions with Complements. Artificial Intelligence, Vol. 48, 1991, No. 1, pp. 1–26.

[28] Tautz, C.—von Wangenheim, C.: REFSENO: A Representation Formalism for Software Engineering Ontologies. IESE-Report No. 015.98/E, Fraunhofer Institute IESE, 1998.

[29] OMG. Unified Modeling Language (UML) Infrastructure. Version 2.5, Formal/15-03-01, Object Management Group, Needham, MA, USA, 2015.

[30] Sattler, U.: Description Logics for the Representation of Aggregated Objects. In: Horn, W. (Ed.): Proceedings of the 14th European Conference on Artificial Intelligence, Berlin, Germany, 2000, pp. 239–243.

[31] Iacovelli, A.—Souveyet, C.: Towards Common Ground in SME: An Ontology of Method Descriptors. In: Ralyté, J., Mirbel, I., Deneckère, R. (Eds.): Engineering Methods in the Serviceoriented Context. Proceedings of 4th IFIP WG8.1 Working Conference on Method Engineering (ME 2011), Paris, France. Springer, Heidelberg, IFIP Advances in Information and Communication Technology, Vol. 351, 2011, pp. 77–90.

[32] Morales-Trujillo, M.—Oktaba, H.—Piattini, M.: Using Technical-Action-Research to Validate a Framework for Authoring Software Engineering Methods. 17th International Conference on Enterprise Information Systems (ICEIS '15), INSTICC, 2015, pp. 15–27, doi: 10.5220/0005338800150027.

[33] Aharoni, A.—Reinhartz-Berger, I.: Representation of Method Fragments: A Comparative Study. In: Ralyté, J., Brinkkemper, S., Henderson-Sellers, B. (Eds.):

Situational Method Engineering: Fundamentals and Experiences. Springer, Boston, MA, IFIP Advances in Information and Communication Technology, Vol. 244, 2007, pp. 130–145.

[34] Firesmith, D.—Henderson-Sellers, B.—Zowghi, D.: Using the OPEN Process Framework to Produce a Situation-Specific Requirements Engineering Method. Software Engineering Institute, 2005.

[35] van de Weerd, I.—Brinkkemper, S.—Souer, J.—Versendaal, J.: A Situational Implementation Method for Web-Based Content Management System-Applications: Method Engineering and Validation in Practice. Software Process: Improvement and Practice, Vol. 11, 2006, No. 5, pp. 521–538.

[36] Rolland, C.—Plihon, V.—Ralyté, J.: Specifying the Reuse Context of Scenario Method Chunks. Proceedings of the 10th International Conference on Advanced Information Systems Engineering (CAiSE '98). Springer, Lecture Notes in Computer Science, Vol. 1413, 1998, pp. 191–218, doi: 10.1007/BFb0054226.

[37] Sturm, A.—Reinhartz-Berger, I.: Applying the Application-Based Domain Modeling Approach to UML Structural Views. International Conference on Conceptual Modeling (ER 2004). Lecture Notes in Computer Science, Vol. 3288, 2004, pp. 766–779, doi: 10.1007/978-3-540-30464-7_57.

[38] Kelemen, Z. D.—Kusters, R. J.—Trienekens, J.—Balla, K.: Selecting a Process Modeling Language for Process Based Unification of Multiple Standards and Models. Technical Report TR201304, Budapest, Hungary, 2013.

[39] OMG. Business Process Model and Notation (BPMN). Version 2.0, Formal/2011-01-03, Object Management Group, Needham, MA, USA, 2011.

[40] ISO/IEC, 24744 Software Engineering – Metamodel for Development Methodologies. International Organization for Standardization, 2007.

[41] Niknafs, A.—Asadi, M.: Towards a Process Modeling Language for Method Engineering Support. 2009 WRI World Congress on Computer Science and Information Engineering, 2009, pp. 674–681, doi: 10.1109/CSIE.2009.956, doi: 10.1109/CSIE.2009.956.

[42] Morales-Trujillo, M.—Oktaba, H.—Piattini, M.: Bottom-Up Authoring of Software Engineering Methods and Practices. Journal of Applied Research and Technology, Elsevier, Submitted 2016.

[43] Zamli, K. Z.—Mat Isa, N. A.: A Survey and Analysis of Process Modeling Languages. Malaysian Journal of Computer Science, Vol. 17, 2004, No. 2, pp. 68–89.

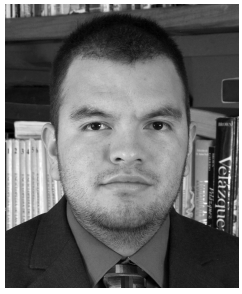
[44] Gruninger, M.—Lee, J.: Ontology: Applications and Design. Communications of the ACM, Vol. 45, No. 2, 2002, pp. 39–41.

[45] Zhang, Y.—Zhang, W.: Description Logic Representation for Requirement Specification. In: Shi, Y., van Albada, G. D., Dongarra, J., Sloot, P. M. A. (Eds.): Computational Science (ICCS 2007). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 4488, 2007, pp. 1147–1154.

[46] Abad, Z. S. H.—Sadi, M. H.—Ramsin, R.: Towards Tool Support for Situational Engineering of Agile Methodologies. 2017 17th Asia Pacific Software Engineering Conference (APSEC 2010), IEEE, 2010, pp. 326–335.

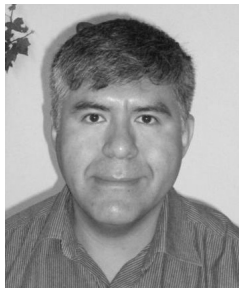
[47] Arni-Bloch, N.: Towards a CAME Tools for Situational Method Engineering. Interoperability of Enterprise Software and Applications, IFIP-ACM, 2005.

[48] Boronat, A.—Meseguer, J.: An Algebraic Semantics for MOF. Formal Aspects of Computing, Vol. 22, 2010, No. 3-4, pp. 269–296, doi: 10.1007/s00165-009-0140-9.



Miguel Morales-Trujillo received his Ph.D. in computer science from the National Autonomous University of Mexico (UNAM). He is the UNAM Representative at the Object Management Group. He has been Assistant Professor at the Science Faculty of the UNAM since 2010. His research interests are software engineering and process engineering.

Hanna Oktaba received her Ph.D. in computer science from the University of Warsaw, Poland. She has been Full Professor at the UNAM since 1983. She was in charge of the MoProSoft project for the Mexican government's PROSOFT program. She is Technical Leader of the Mexican delegation in WG24 of ISO/IEC JCT1 SC7. Nowadays she leads the KUALI-KAANS research group. Her research interests are software engineering and software quality.



Francisco Hernández-Quiroz received his Ph.D. in computer science from the Imperial College of Science, Technology and Medicine in London. He has been Full Professor at the UNAM since 2002. His research interests are computability theory and its practical and philosophical implications, as well as modal logic in computer science and philosophy.

Boris Escalante-Ramírez received his Ph.D. in computer science from the Technical University of Eindhoven. His research interests are computational models of human vision and their applications to digital image processing. He is a member of the National Research System of Mexico.