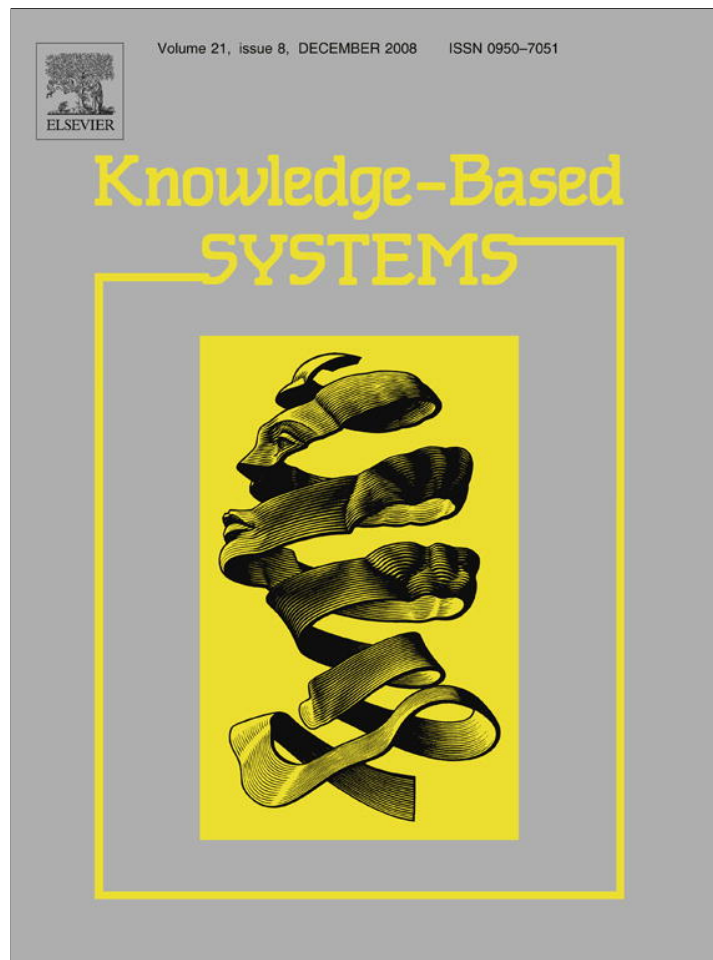


Provided for non-commercial research and education use.
Not for reproduction, distribution or commercial use.



This article appeared in a journal published by Elsevier. The attached copy is furnished to the author for internal non-commercial research and education use, including for instruction at the authors institution and sharing with colleagues.

Other uses, including reproduction and distribution, or selling or licensing copies, or posting to personal, institutional or third party websites are prohibited.

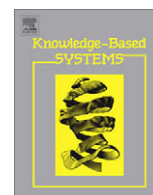
In most cases authors are permitted to post their version of the article (e.g. in Word or Tex form) to their personal website or institutional repository. Authors requiring further information regarding Elsevier's archiving and manuscript policies are encouraged to visit:

<http://www.elsevier.com/copyright>



Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Design issues for knowledge artifacts

G. Salazar-Torres^{a,*,1}, E. Colombo^{b,1}, F.S. Correa Da Silva^{a,1}, C.A. Noriega^{a,1}, S. Bandini^{b,1}^aDepartment of Computer Science – IME, University of São Paulo, Rua do Matão, 1010, Cidade Universitaria, 05508-090 São Paulo, Brazil^bDepartment of Computer Science, Systems and Communication, University of Milan-Bicocca, viale Sarca 336, 20126 Milan, Italy

ARTICLE INFO

Article history:

Received 19 December 2007

Accepted 30 March 2008

Available online 13 April 2008

Keywords:

Knowledge artifacts

Knowledge-based systems

Ontologies

Knowledge management

Agile software development

ABSTRACT

The notion of knowledge artifact has rapidly gained popularity in the fields of general knowledge management and more recently knowledge-based systems. The main goal on this paper is to propose and discuss a methodology for the design and implementation of knowledge-based systems founded on knowledge artifacts. We advocate that the systems built according to this methodology can be effective to convey the flow of knowledge between different communities of practice. Our methodology has been developed from the ground up, i.e. we have built some concrete systems based on the abstract notion of knowledge artifact and synthesized our methodology based on reflections upon our experiences building these systems. In this paper, we also describe the most relevant systems we have built and how they have guided us to the synthesis of our proposed methodology.

© 2008 Elsevier B.V. All rights reserved.

1. Introduction

Knowledge artifacts are artifacts made of knowledge. These artifacts can be very useful to ensure the effectiveness of the transfer and utilization of knowledge in organizations of all sorts.

The precise conceptualization of knowledge artifacts is still on the making. Indeed, this is a relatively recent concept found in the specialized literature devoted to knowledge management and artificial intelligence.

In the present article, we discuss the notion of knowledge artifacts, from a variety of perspectives. We focus more thoroughly on a perspective more akin to notions usually found in the field of Artificial Intelligence, so that we can identify the foundations for a methodology to build computational knowledge artifacts, i.e. knowledge artifacts embodied in the form of software systems.

Our proposed methodology has been built from the ground up. We have adopted the abstract notion of knowledge artifacts to build a variety of software systems with the specific purpose of providing support to the flow and application of knowledge in organizations and, based on our experience building these systems, we have abstracted a general model and a general methodology to build knowledge artifacts in general, in such way that they admit renditions in the form of software systems.

In Section 2, we discuss the notion of knowledge artifact, starting with the general notion of an *artifact* and specializing this notion to artifacts made of knowledge. In Sections 3 and 4, we present our two most representative experiments so far building knowledge artifacts based on their abstract conceptualizations. These experiments form the ground upon which we have built a general methodology to design and implement computational knowledge artifacts. In Section 5, we introduce our methodology. We emphasize that this methodology is work in progress, and in the present article we present it at its latest version. Finally, in Section 6 we present some further discussion and proposed future work.

2. Knowledge artifacts

In [15], we find that an artifact is an object that has been intentionally produced for a specific purpose. Moreover, according to [18], an artifact is the result of a disciplined human activity, following rules and based on training and experience. As a consequence, every artifact has an author and a purpose.

Artifacts can be evaluated in terms of how their actual features match the features intended by their authors and the purposes to which they are built. Given a purpose P , an author A devises an artifact and obtains an invention, an idea or a project that describes the artifact at an abstract level. We can refer to this object, resulting from the author intellectual work, using the symbol I . Finally, the actual artifact R is used for the purpose P . It should be noticed that the purpose that has produced both the design of I and the implementation of the artifact R could differ from the purpose of the user of an artifact.

* Corresponding author.

E-mail addresses: gsalazar@ime.usp.br (G. Salazar-Torres), ettore.colombo@disco.unimib.it (E. Colombo), fcs@ime.usp.br (F.S. Correa Da Silva), cnoriega@ime.usp.br (C.A. Noriega), bandini@disco.unimib.it (S. Bandini).¹ We thank Cassio Wallner at EMBRAER (Empresa Brasileira de Aeronautica) for his invaluable help in this work.

The artifact R described by I is the real object. The author would like R to have the intended features to fit the original purpose P . The description I of the artifact is the result of a design process. It is a formal and abstract representation of an actual object R . The devised artifact needs to be implemented, in order to produce the actual artifact R that is going to be used.

As presented in Fig. 1, P , I and R are related through *design*, *implementation* and *utilization*. A successful set of relationships is that in which a design leads to an I that perfectly matches a given purpose P , whose implementation leads to an R that perfectly matches I , and such that the utilization of R also perfectly matches P .

One interesting point to be noticed is that the arrows in Fig. 1 are one-to-many relations. In fact, a purpose P can lead to a variety of designed artifacts I_1, I_2 , etc. For example, the purpose of writing on paper can lead to the design of a fountain pen, a pencil, some sort of brush, etc. or to a more detail design, such as “XYZ model of fountain pen”. Analogously, a design I can lead to a variety of implementations R_1, R_2 , etc. For example, the implementation of a fountain pen (or of a more specific “XYZ model of fountain pen”) can lead to actual objects, each one with its specific nib, shape and ink bottle. Finally, an actual artifact R can serve a variety of purposes P_1, P_2 , etc., including purposes not initially taken into account during design and implementation, a noteworthy example being the music turntables, initially used for the purpose of reproducing previously recorded music and now use to create original pieces of music e.g. in hip hop.

Summarizing, an artifact is hence characterized by a particular triple (I, R, P) , in which a design I is implemented as an actual object R that is used for a purpose P (which, as mentioned above, is not necessarily the purpose P_0 that led to the design of I). The separate elements I, R and P can be put together in a triple (I, R, P) that is the artifact in the context view of the artifact R .

Our focus in this article, however, is not on artifacts in general but on knowledge artifacts (KAs). We thus need a working definition of knowledge, so that it can be the stuff with which we design and build artifacts.

As pointed out elsewhere [8], defining knowledge is not an easy or already sorted out task. Some characterizations of knowledge that are useful to our discussion are:

- The data-information-knowledge hierarchy [9,10,24]: data are the codification of observable facts. Information is data endowed with meaning. Knowledge is actionable information, i.e. information that provides agents with the necessary input to perform actions.
- The philosophical stance [11]: knowledge is justified true belief. This characterization, which amounts to Plato, assumes as knowledge any belief that is true and justifiably believable. It is a precise characterization, although it relies on three other complex concepts, namely belief, truth and justifiability.

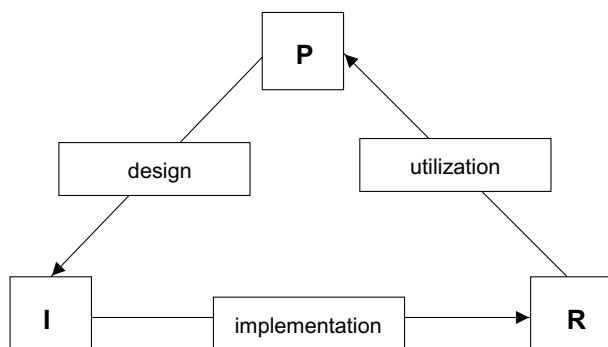


Fig. 1. Design, implementation and utilization of an artifact.

- The agent-based view [8]: knowledge is the capability of an agent to change the world. The world changes whenever the truth value of a sentence that describes one of its features needs updating. When one such change occurs as the result of the purposeful action of an agent, the agent necessarily has knowledge.

These characterizations are complementary to each other. We employ the three of them along this article.

Given a purpose P , an agent can design an artifact I “made of” knowledge, i.e. codified and interpretable data such that, when presented to any other agent, endows this agent with the capability to perform some action that can change the world. The designed artifact I is such that, when embodied in an implementation R , can be effectively used to change the world.

As an illustrative example, we can consider the purpose P of having a chocolate cake to eat. An agent (i.e. the baker in this case) can devise an artifact I , namely a recipe providing the steps to bake a cake. The designed artifact I is generic: it does not include that it will be a chocolate cake. We are considering a generic class of cakes which can be provisionally named “tea-time cakes” which share the same general steps and ingredients to be baked. The recipe R is the concrete recipe obtained from this process and it is derived from the generic devised artifact I . The obtained recipe can be used for some different purposes, e.g. so that an individual can bake a chocolate cake (i.e. the purpose P), or so that a student in a cookery school can complete his homework (i.e. a purpose P').

We find several definitions of a knowledge artifact (KA) in the literature. These definitions can be roughly classified in four groups:

- (i) a KA is an artifact which represents knowledge;
- (ii) a KA is an artifact which represents an encoding of knowledge;
- (iii) a KA is an artifact whose creation requires specialized knowledge;
- (iv) a KA is an artifact “made up” of knowledge.

One of the most frequently used definitions of a KA is due to Holsapple and Joshi. They describe KAs as “objects that convey and hold usable representation of knowledge” [16]. If we take definition 2 above together with Holsapple and Joshi’s definition, we have usable- i.e. executable-encodings of knowledge, which can suitably be embodied as computer programs, written in programming languages such as C, Java, or declarative modeling languages such as XML, OWL or SQL.

The utility of a KA results from the perspective from which it is approached. There are at least four perspectives that can be considered:

- The *knowledge management perspective* that aims to develop systems to create, store and manage knowledge artifacts. According to this perspective, a KA is an object that holds some representation of knowledge (documents, files, etc.), and their management can add value to the organization through improved accessibility to its knowledge resources. Based to this perspective, knowledge artifacts are vehicles for knowledge sharing.
- The *community support perspective* that aims to create sites and objects in which community processes (e.g. negotiation) are supported. These processes generate shared objects (the knowledge artifacts), such as email letters or the contents of a blog. According to this perspective, KAs are still carriers of knowledge that supports its sharing, but they also contribute to the evolution of the community by supporting its natural learning processes.

- The *computational perspective* that considers KAs as recorded representations of knowledge to be used in computations within a particular problem solving method in a computational system.
- The *artificial intelligence perspective* that aims to separate knowledge from its utilization. A KA persists even without a system for its management. Moreover, its identification in an organization could be a useful guide for the development of knowledge-based systems by following a KA-based methodology.

In this paper, we are going to follow the Artificial Intelligence perspective and consider a knowledge artifact as an artifact whose primary constituent is knowledge.

A more realistic example of a KA is the decision process for the compounder of a truck tyre rubber blend. For instance, given a specific purpose *P* coming from a market requirement on tyre performances, such as “obtain a higher tear resistance”, the agent (the compounder in this case) has to design a new rubber blend recipe. In order to do this the compounder obtains, through the adaptation of an already used recipe, a new one. In this case, the knowledge model involved in the definition of the new recipe is the knowledge artifact (KA) of the compounders’ community. Moreover, we can define and call KA also the abstract and domain-independent model of the “design by adaptation” process. We shall return to this example in the following sections.

In Fig. 2, we depict the KAs related to baking cake and to devising a new rubber blend for truck tyres.

The concept of KAs is interesting because it unveils and characterizes the opportunities to reuse the components of the sorts *P*, *I* and *R* to assemble different artifacts. Typically, when a knowledge artifact is reused, it characterizes the flow of high-level, abstract practices between different groups whose internal cohesion is given by some sort of commonality in their activities. In the general literature, such groups have been named Community of Practice (CoP) [25]. An object that is used across communities of practice, possibly by ascribing different semantics to its components, is called a boundary object, for the obvious reason that it becomes the point of contact between two (or more) communities of practice, thus being located in the boundary that separates them as a bridge that crosses the border between two nations. This point has been further explored in [2].

Since a KA is primarily an artifact, it is possible to describe it through a (P, I, R) triple including it in the *design-implementation-utilization* cycle. Considering a specific community of practice, the purpose *P* is the goal of the community (e.g. the design of a product or of a manufacturing process). According to the proposed cycle, *I* is the result of the design activity: in this case, *I* is not a single object but the join of the organizational elements of the community (e.g. formal as well as tacit roles, strategies and habits of the participants of the community) that directly influence the decision making process to complete the community specific work. To

complete the triple, *R* is the knowledge model involved in the this decision making process and shared among the community.

In the following sections we give two examples of KAs. The first example is named *design by adaptation*, and relates to the design of novel rubber blends for truck tyres. The second example is named *Maintenance and Repair*, and relates to the design of strategies to ensure the appropriate functioning of complex electromechanical devices. These two examples are presented in order to illustrate the concrete steps involved in the design and implementation of a KA. They have provided the empirical basis for the formulation of a methodology to design and implement KAs in general, which we then describe in Section 5.

3. Concrete KA 1: design by adaptation

In this section, we present an example of a KA built in the PTruck project [2]. The goal of this project was the development of a KBS based on the KA belonging to a community of rubber compounders. In particular, the application domain was the Business Unit Truck of Pirelli Tyres. Moreover, the KBS involved also other communities devoted to other design and manufacturing processes within the firm: curing and mixing.

In the following paragraphs, the BNF notation is exploited to describe the grammar related to the knowledge artifact design by adaptation.

3.1. Domain description

The compounding problem is about looking for the most suitable way of combining ingredients to design a product, i.e. a recipe, that indicates which ingredients and corresponding proportions are involved in the construction of a product by the indication of their amounts. The problem has been tackled exploiting the KBS approach in several domains (e.g. tablet [7], color [5] and rubber compound [4] design).

It concerns the definition of the raw materials (i.e. the ingredients) to be used and the related amounts (measured in Parts per Hundred Rubber – PHR). Domain experts are chemical technologists, skilled in managing raw materials (e.g. carbon black, sulfur). Since compounding is performed through the adaptation of a prior recipe, knowledge acquisition in the PTruck project was focused on the adaptation knowledge modelled in Design Relations (between tyre performances and blend features) and Compounding Relations (between blend features and recipe modifications).

In this case, purpose *P* is thus the design of a rubber compound through the definition of its recipe; *R* is the knowledge artifact called “design by adaptation” defined and specialized in the application domain of tyre rubber compounding; *I* is the set of elements in the Business Unit Truck organization that influence the compounding process, such as the role of compounders, their relations with market division people and with lab people who test rubber properties. Moreover, the strategy is relevant for the creation of

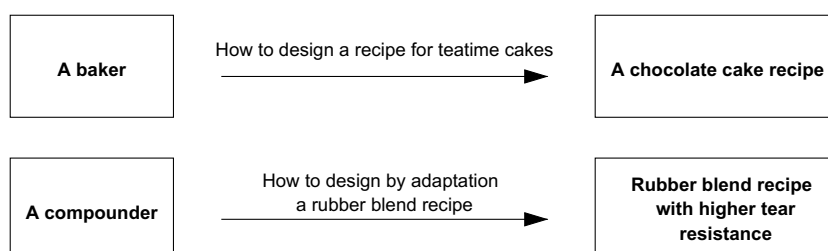


Fig. 2. KAs for baking a cake and for preparing a recipe for a rubber blend.

R: designing recipes starting from an existing recipe and exploiting the knowledge experienced in previously products directly guide the community members to the definition of the “design by adaptation” KA.

3.2. Ontological knowledge

Basic domain elements and Relations among them were identified during the ontological analysis. A particular kind of relation is represented by Functions: specific grammar elements that describe possible recipe transformations.

Basic elements are represented by symbols describing recipe names, low and high level properties (blend features and tyre performances).

```
<low level property>::="llp1" | "llp2" | ...
<high level property>::="hlp1" | "hlp2" | ...
<recipe>::="RECIPE_1" | "RECIPE_2" | ...
```

Recipes are made up of ingredients, each one belonging to a family (e.g. Natural Rubber-NR or Carbon Black-CB).

```
<ingredient>::="I1" | "I2" | "I3" ...
<family>::="F1" | "F2" | "F3" | ...
```

Within a recipe it is possible to identify blends of ingredients made up of elements of the same family (e.g. blends of CB) and systems (e.g. Polymeric Matrix, Fillers). Each system is a functional group of ingredients, inserted into a recipe to provide the rubber blend with a particular feature. An example of system is the Curing one, made up of sulfur and other elements playing a fundamental role during tyre curing process which gives the tyre required thermal–mechanical properties.

```
<system>::="S1" | "S2" | "S3" | ...
```

A family is a set of ingredients with common chemical and physical properties. Two elements of a family are distinguished by means of attributes (e.g. for CB, the structure and the surface area).

```
<attribute>::= <discrete attribute> | <continuous attribute> | <type>
<discrete attribute>::="DA1" | ...
<discrete attribute value>::="DAV1" | "DAV2" | ...
<continuous attribute>::="CA1" | "CA2" | "CA3" | ...
<type>::="T1" | "T2" | ...
<type value>::="TV1" | "TV2" | "TV3" | ...
```

Considering the truck tyre domain, the product is made up of several components (e.g. tread, liner). The component and the tyre utilization (i.e. axis, market segment and area) define the blend usage context (e.g. the tread for the directional truck axis used in Turkish quarries). This is important to define constraints on recipe formulation depending on its usage.

```
<usages>::="usage1" | "usage2" | ...
```

Recipe formulations are described by sentences like “(RECIPE_1, NR01, 50)” and “(RECIPE_1, CB01, 35)”. These indicate that ingredients NR01 and CB01 are in the recipe “RECIPE_1” with amounts of 50 and 35 PHR, respectively.

```
<recipe element>::= ‘(’ <recipe> ‘,’ <ingredient> ‘,’
<amount> ‘)’
```

```
<amount>::= <numeral-list> | <numeral-list> ‘,’ <numeral-list>
<numeral-list>::= <numeral> | <numeral><numeral-list> | empty
<numeral>::= ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’
| ‘8’ | ‘9’
```

Families are sets of ingredients. Their elements are described by attributes. For instance, “CB01 belongs to CB” and “NR01 belongs to NR” express that CB01 and NR01, respectively, belong to Carbon Blacks and Natural Rubbers. Additionally, “CB is described by structure” and “NR is described by NR Type” associate the structure attribute to CB and the NR Type attribute to NR.

```
<family ingredient>::= <ingredient> ‘belongs to’
<family>
<family attribute>::= <family> ‘is described by’
<attribute>
```

Each family is related to a functional system indicating what role specified ingredients play in the compound. Thus, sentences like “CB is involved in the Fillers system” and “NR is involved in the Polymeric Matrix” state that Carbon Blacks must be included in the Fillers system and that Natural Rubbers are in the Polymeric Matrix.

```
<family system>::= <family> ‘is involved in’ <system>
```

Attribute value descriptions are given as follows.

```
<attribute value>::= <ingredient> ‘has value’
<amount> ‘for’ <continuous attribute>
| <ingredient> ‘has value’ <type values> ‘for’
<type>
| <ingredient> ‘has value’ <discrete attribute values> ‘for’
<discrete attribute>
```

In these examples, “CB1 has value 30 for structure”, “NR01 has value RSS for NR Type” and “BR01 has value high for cis” determine that “structure”, “NR Type” and “cis” are names of a continuous attribute (a type and a discrete attribute, respectively). Some constraints defining recipe construction boundaries in the considered domain can be described referring to the blend usage. These are constraints on combination cardinality, combination amount, ingredient amounts and continuous attribute value limits.

```
<combination>::= <family> ‘has maximum blend cardinality’
<amount> ‘for’ <usage>
<ingredient constraint>::=
<amount> ‘is the upper bound for’ <ingredient>
‘for’ <usage>
| <amount> ‘is the lower bound for’ <ingredient>
‘for’ <usage>
<family constraint>::= <amount> ‘is the upper bound for’
<family> ‘for’ <usage>
| <amount> ‘is the lower bound for’ <family> ‘for’
‘usage’
<continuous attribute constraint>::=
<amount> ‘is the upper bound for’ <continuous attribute>
‘for’ <usage>
| <amount> ‘is the lower bound for’ <continuous attribute>
‘for’ <usage>
```

Hence, it is possible to indicate that A is the maximum number of elements for Carbon Black combinations for a specific usage

(i.e. “CB has maximum blend cardinality A for usage1”). Furthermore, given L and U, the values of the lower and upper bound of a generic interval, respectively, the amount of a particular ingredient (e.g. Sulfur) could be limited for usage1 (i.e. “L is the lower bound for Sulfur for usage1” and “U is the upper bound for Sulfur for usage1”). Similarly, also intervals on families (e.g. CB) could be specified for a usage (e.g. usage1) through the sentences “L is the lower bound for CB for usage1” and “U is the upper bound for CB for usage1”. Sentences “L is the lower bound for structure for usage1” and “U is the upper bound for structure for usage1” indicates the interval of admissible values for the continuous attribute structure. Also system constraints not depending on the usage exist: for instance “Polymeric Matrix has constant quantity” means that the amount augmentation of a NR ingredient is forbidden as the associated system (i.e. the Polymeric Matrix) must have a constant quantity. A modification that provides substitutions not modifying the total ingredient amounts is allowed.

```
<system constraint> ::= <system> ‘has constant quantity’
```

Possible recipe modifications have been identified as functions. These stand for the possible Recipe Interventions (RIs) to adapt a recipe.

```
<modification action> ::= <shift>
| <augmentation>
| <substitution for family type>
| <substitution for discrete attribute>
| <substitution for continuous attribute>
<shift> ::= ‘+’ <family> ‘-’ <family>
<augmentation> ::= ‘+’ <family>
<substitution for discrete attribute> ::=
<family> ‘for’ <discrete attribute> ‘
from’ <discrete attribute value> ‘to’ <discrete attribute value>
<substitution for continuous attribute> ::= <family>
‘for’ <continuous attribute>
<substitution for family type> ::=
<family> ‘for’ <type> ‘from’ <type value> ‘to’
<type value>
```

For example the augmentation of Carbon Black blend amount is “+CB”, while a Natural Rubber substitution taking care of their type is specified by “NR for NR Type from RSS to SMR”, where “RSS” and “SMR” are two possible NR types.

3.3. Expert knowledge

Knowledge elicitation also involves the epistemological analysis of the compounding decision making process. An important result is the T-Matrix (see Fig. 3) that shows the relationships between

TP 1	⊙↑	⊙↑	⊙↓	⊗
TP 2	⊗	⊙↑	▲↓	⊙↑
TP 3	⊙↑	⊗	▲↓	⊙↓
TP 4	⊙↓	▲↑	⊗	▲↑
TP 5	⊗	⊙↓	⊙↑	⊗
	BF 1	BF 2	BF 3	BF 4
RI 1	⊙↓	▲↓	⊗	⊗
RI 2	⊙↑	⊙↓	▲↑	⊙↑
RI 3	⊙↓	⊗	⊗	▲↑
RI 4	▲↓	⊙↑	▲↑	⊗

Correlation	⊙	Strong
	⊙	Good
	▲	Weak
	⊗	No corr.
Proportionality	↑	Direct
	↓	Inverse

Fig. 3. An example of T-Matrix binding compounding relations and design relations.

Tyre Performances (TPs) and Blend Features (BFs) and between BFs and Recipe Interventions (RIs).

Each T-Matrix relates to a particular usage and shows the recipe modification effects in terms of blend feature variations (i.e. as Compounding Relations) and the blend feature change effects on tyre performance (i.e. as Design Relations). For example, the strong correlation with a direct proportionality between RI2 and BF4 stands for the sentence: RI2 variation produces a strong variation on BF4. In this grammar, “is related in compounding to” and “is related in design to” and some additional terms to describe the grade of correlation and proportionality have been identified.

```
<correlation> ::= ‘strong’ | ‘good’ | ‘weak’
<proportionality> ::= ‘direct’ | ‘inverse’
<correlation description> ::= ‘no’
| (‘<correlation> ‘correlation and’ <proportionality> ‘proportionality’)
```

For instance, calling C and P the value of correlation and proportionality, “+CB is related in compounding to tensile strength with (C correlation and P proportionality)” describes how the CB amount increase and the tensile strength feature (i.e. a low level property) are correlated and with which proportionality.

```
<design relation> ::=
<low level property> ‘is related in design to’ <high level property> ‘
with’ <correlation description> ‘correlation’
<compounding relation> ::=
<modification action> ‘is related in compounding to’ <low level property> ‘
with’ <correlation description> ‘correlation’
```

Moreover, given a usage, it is possible to weight the relevance of a blend feature or of a tyre performance. For each blend feature (or tyre performance) it is also possible to indicate the preferred modification direction.

```
<low level property weight> ::=
<amount> ‘is the weight of’ <low level property>
‘for’ <usage>
| <direction> ‘is the preferred direction of’ <low level property>
‘for’ <usage>
<high level property weight> ::=
<amount> ‘is the weight of’ <high level property>
‘for’ <usage>
| <direction> ‘is the preferred direction of’ <high level property>
‘for’ <usage>
<direction> ::= ‘augmentation’ | ‘reduction’
```

Calling W the tensile strength weight, “W is the weight of tensile strength for usage2” and “augmentation is the preferred direction of wet handling for usage1” can be written.

3.4. Knowledge artifact based support

Adaptation knowledge can be used to support the research of interventions that could generate a new recipe starting from an existing one and incrementally adjusting to given requests on property variations. Considering these instances of Compounding and Design Relations, “RI2 is related in compounding to BF4 with (strong correlation and direct proportionality)” and “BF4 is related in design to TP4 with (weak correlation and direct proportionality)”, the developed KBS must be able to treat them: if RI2 is an augmentation function and a recipe intervention able to weakly

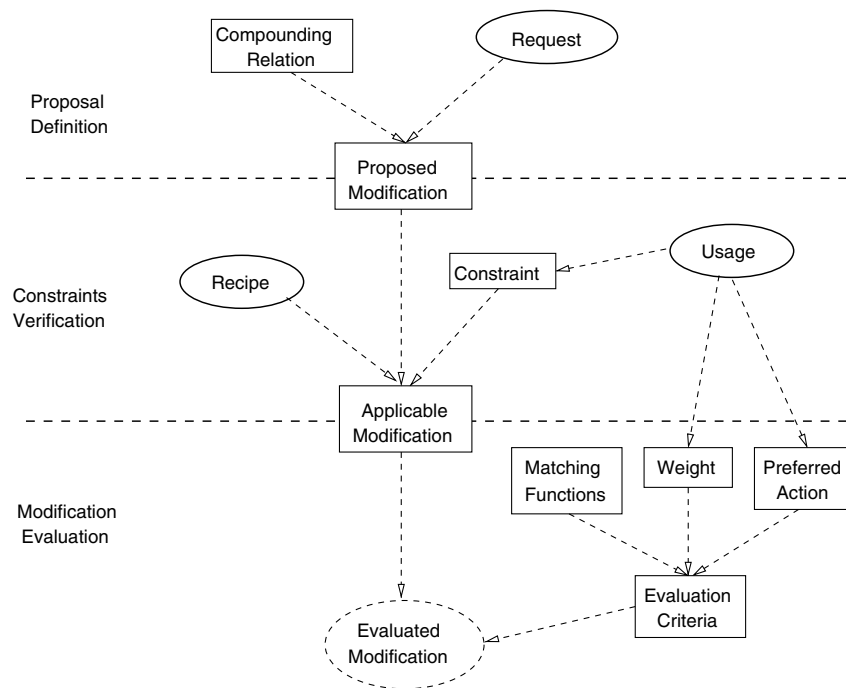


Fig. 4. The problem solving method capturing the semantics of the design by adaptation KA. Bold ellipses stand for PSM input.

decrease TP4 is needed, an interpretation of the introduced sentences can suggest an inverse application of RI2 (i.e. the reduction of the ingredient involved in RI2). The problem solving method (PSM) defined in the PTruck project (see Fig. 4) gives an evaluation of the applicable recipe interventions to meet requests of blend feature variations.

The PSM requires specific inputs (i.e. a recipe and a usage) and a list of variations described by a blend feature specification, a direction and a variation extent. The direction can be to increase, to decrease or to maintain a blend feature, while managed variation extents are strong, good and weak. The PSM is divided into three rule sets: Proposal Definition, Constraints Verification and Modifications Evaluation. Proposed modifications are defined to change blend features in accordance to the input requests. The extent of each one of them is used to qualify the related proposed modification. The recipe and the usage allow to set which constraints the recipes must satisfy. Each proposed modification, if applied, can bring to a recipe that violates or complies with these constraints. In the latter case, the modification is an applicable modification because it stands for an admissible solution. Since the PSM goal is to rank applicable modifications, the third rule set involves a evaluation criteria to calculate the modification votes. In doing so, the blend feature importance expressed by the weight related to the input usage is considered. This is combined with the matching function result adopting a weighted average function. The matching function gives an estimation of how much a sub-effect on a specific blend feature fits the related request.

4. Concrete KA 2: Maintenance and Repair

The *Maintenance and Repair* KA was implemented in a large manufacturing company in Brazil, in order to help in the diagnostics and repair of complex electromechanical devices. In the following paragraphs, after a description of the application domain, the BNF rules to describe the knowledge artifact Maintenance and Repair are given.

4.1. Domain description

Maintenance and Repair stands for the task of diagnosis and repair of some kind of complex device in which damages are identified in some of its components. The purpose is to reestablish its original properties in such way as to reduce maintenance costs. The KA is mainly used in maintenance environments where an abstract (e.g. mathematical) model is used to describe the device (e.g. Finite Element Analysis, Static and Fatigue Stress calculations), because it would be too costly to develop real devices and test several damages scenarios.

The process of diagnosis and repair starts with applying damages over the structure of the device. The description of the damages is deep since the structure can accept damages in its finest components.

Damages are classified according to their attributes (e.g. depth, location) across the structure. According to this classification, the damaged parts are tested using the mathematical models to define their status. Finally, repair procedures are applied to restore the original properties of the structure.

To show the structure of the *Maintenance and Repair* KA, we shall use the domain of a car engine repair.

4.2. Ontological knowledge

The grammar describing the ontological structure includes, in the case of a car, the engine and its parts:

```
<device>::="Nissan350ZEngine" | "FerrariEngine" | ...
<device description>::= <device> "has" <part>
<part>::=
  <cylinder> | <piston> | <spark plug> |
  <valve> | <cylinder bank>
```

For example we could have the FerrariEngine described by the sentence “FerrariEngine has bank1” to indicate that the “FerrariEngine” engine has a cylinder bank “bank1”.

Since the purpose of the Maintenance and Repair KA is to repair fine parts, it is necessary to go deeper in the description of the device (e.g. cylinders):

```
<cylinder arrange>::="v"|"flat"|"inline"
<cylinder bank>::="bank1"|"bank2"
<cylinder bank description>::=
<cylinder bank>" has" <cylinder arrange>" arrange" |
  <cylinder bank>" has" <number>"cylinders" |
  <cylinder bank>" is composed by" <cylinder> |
<cylinder material>::="aluminum" |...
<cylinder>::="cylinder1"|"cylinder2" |...
<cylinder description>::=
  <cylinder>" contains" <piston> |
  <cylinder>"has depth" <real number>
  |<cylinder>" has" <cylinder material>
```

Following the previous example, we could describe the cylinder bank “bank1” with the sentences “bank1 has v arrange”, “bank1 has 10 cylinders”, “bank1 is composed by cylinder1” and “cylinder1 contains piston1”. Further description could include the piston, its material (e.g. aluminum) or thickness and finer elements like the piston head and skirt:

```
<piston>::="piston1"|"piston2" |...
  <piston description>::=
    <piston>" has" <piston material>" material" |
    "(" <piston>"," <piston component>")"
<system>::="mm"|"cm"
<piston component>::="piston1head"
|"piston1skirt" |...
<piston material>::="aluminum"|"ceramic-aluminum"
<piston component type>::="piston head"|"piston
skirt"
<piston component description>::=
  "(" <piston component>"," <piston component type>,"
  <piston component property>"," <amount>
<system>)"
<piston component property>::="thickness"
<damageable component>::= <piston component>
```

```
<damageable component type>::= <piston component
type>
<damageable component property>::= <piston compo-
nent property>
```

We could describe “piston1” head by “(piston1, piston1head)” to state that “piston” is composed of “piston1head”. In the sentence “(piston1head, piston head, thickness, 5 mm)” we describe deeply “piston1head” saying that it is piston head part type and has a thickness of 5 mm.

4.3. Epistemological knowledge

Now we have to describe what kinds of damage can be applied to this device and how they are related to the elements of the car engine. It is worth remarking that that not all of the parts can have a related damage. The basic idea is to have damages categorized by status and class. Status tells us how to apply a repair (if it can be repaired) and the class tells us what kind of damage it is. This last property can also indicate if a particular damaged part can have a provisioned or permanent repair. The purpose of this KA is to reset the initial properties of the damaged parts, so that the entire device can continue working:

```
<repair>::="repair1"|"repair2" |...
<repair type>::="temporal"|"permanent" |...
<repair description>::=
  "(" <repair>"," <repair type>"," <damage>)"
<damage>::="damage1"|"damage2" |...
<damage type>::= <mechanical> | <termo-mechanical>
<mechanical>::="crack"|"hole" |...
<damage status>::="admissible"|"permanent" |...
  <damage description>::=
    <damage>" is a" <damage type> |
    "(" <damage>"," <damageable component>","
    <damageable component property>"," <amount>
<system>)"
<damage classified>::= "(" <damage>"," <damage
status>)"
<damage limit>::=
```

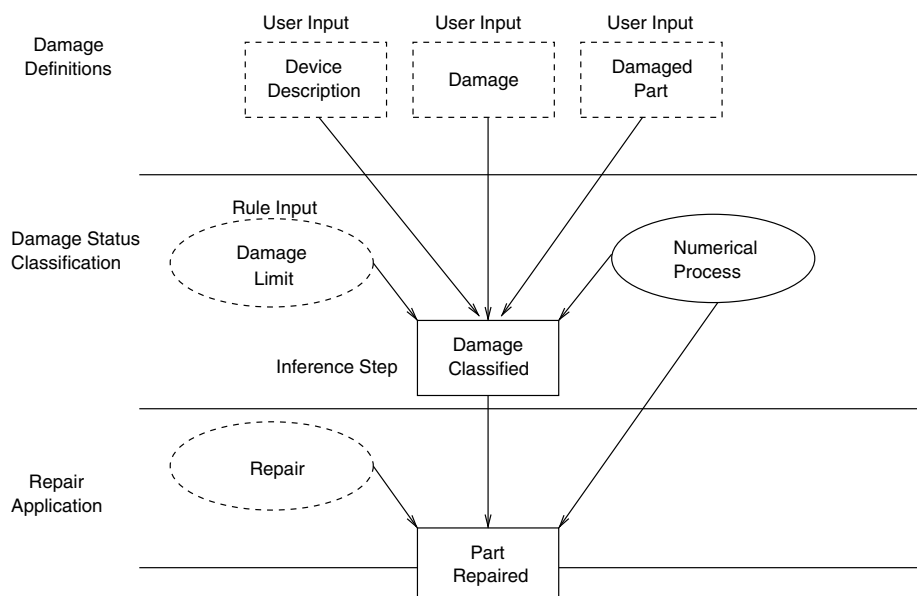


Fig. 5. Maintenance and Repair problem solving method.

```
"(" <damage status>"," <damageable component
type>","
<damageable component property>"," <amount>"
```

Let's assume that "piston1" has a damage in its head "(damage1, piston1head, thickness, 1.5 mm)". Every damage reduces some property of the part in this case, its thickness (e.g. in a spark plug it could reduce its heat resistance). We could refine our description with the sentence "damage1 is a crack", meaning that thickness reduction on the piston has been caused by a crack.

4.4. Knowledge artifact based support

The PSM showed in Fig. 5 determines the status of the damage (e.g. admissible status means that a damage is allowed to stay in the part) and apply a repair procedure to restore initial properties of that part.

Following our example, we have to determine the status of "damage1". For this purpose we use the "(damage limit)" grammar rule to establish the maximum amount of reduction on a part property. If the damage amount is greater than that maximum, then it is classified as a repair, otherwise it is an admissible or another kind of damage status.

In our example we could have a limit rule of 20% of the original piston head's thickness for admissible damages "(admissible, piston head, thickness, 20%)", thus "damage1" is not an admissible repair and need a permanent repair indicated by the rule "(damage1, permanent)". Before applying the repair, the damaged part is submitted to the mathematical procedures to establish if the manufacturer has indeed a repair procedure for this kind of repair. After this process we could have a rule describing the repair "(repair1,permanent,damage1)". Generally, in the case of damaged piston heads, the entire piston is replaced.

5. Development of KA-based systems

In this section we propose an Agile knowledge engineering method based on XP.K to develop KA-based systems (KABS). For an author *A* to build an artifact *R* made of knowledge (Fig. 1), there must be a "method" to elicit the corresponding relevant knowledge.

The intent of the method here discussed is to get computational the knowledge artifact detected. The KA-based System is thus a knowledge artifact embodied in the form of software systems.

Many methods (and methodologies) in the field of Knowledge Engineering have been proposed to elicitate knowledge, among which CommonKADS and MIKE are the most used in the industry. Despite their success, these methods do not effectively handle the characteristics of knowledge modeling (an iterative, collaborative and evolutionary task [17,20,21]) because they are based on a waterfall model.

In [17] and more recently in [1], two new knowledge engineering methods based on the Agile Manifesto were proposed: XP.K and RapidOWL. These methods claim to be more suitable for the knowledge modeling task since the Agile Manifesto principles match the characteristics of knowledge modeling.

The values, principles and practices of the proposed method are presented in the next sections. This proposal is based on our previous experience implementing the PTruck project and more recently at the project in Brazil.

5.1. Values

5.1.1. Community

Knowledge Engineers and KA-based Systems (KABS) developers need to see themselves like members of the community of practice that develops a KABS, and furthermore to get aware of the existence of the user's CoP.

In the first case, as suggested by the *Communication* XP value, maintaining a fluent communication is necessary because it assures CoP's practice. It's worth to notice the KA's role as a relevant mean of communication among team members.

In the second case, the CoP users see the organization as a set of Communities of Practice (CoPs) to have a better notion of who are the people involved in the KABS development.

The principle of *peripheral participation* is about the kind of interaction that exist between the CoP of Knowledge Engineers and Developers and the CoP of users. The practice of *Joint KA Design* details the activities in this interaction.

It's important to consider the XP values of *Humility* and *Respect* because it brings to the Knowledge Engineers and Developers a behavioural code with CoP members, which frequently do not have training in knowledge modeling or programming.

5.1.2. Simplicity

Simplicity is a consequence of using a KA in the development of a KABS. In fact, since the KA is a cognitive pattern owned by the CoP itself, the use of the grammar directly derived from the KA makes the integration among CoP members simpler.

An important recommendation coming from XP.K suggests to design knowledge models as simple as possible. We are considering this hint in our method because though the interaction is got simple by the use of the KA, the design phase (as defined by Winograd in [26]) is difficult and complex. The design is even more complex because the development of the KABS requires all the three dimensions of the knowledge model (intensional, extensional and reasoning, see [3]).

5.1.3. Feedback

A tool that gives support to the KA design is the main source to obtain feedback by the people involved in the development of a KABS. This tool should allow to configure tests on all the dimensions of the KA-based model, along with a simple and intuitive GUI that will provide information about the KA status.

Besides that, this tool must generate source code in a suitable programming language (e.g. Java, C#) reflecting the architecture of a KABS ready to be integrated with the GUI.

5.1.4. Courage

Courage is needed to abandon a KA design when this does not fulfill the requirements of the CoP where the KA is being developed.

It is worth to notice that although much of the source code has to be abandoned, the results of the ontological and epistemological analysis remain intact, since they can be stored independently of the KA instance and capable to be reused in another KA.

5.2. Principles

Many principles of XP.K are reused here, though, some of them affect more the development of a KA and are further analysed. Fig. 6 shows the values and principles of our method.

5.2.1. Work with people's instincts

We are basing our method on CoP theory, so the tacit knowledge of the CoP members is important to design a KA.

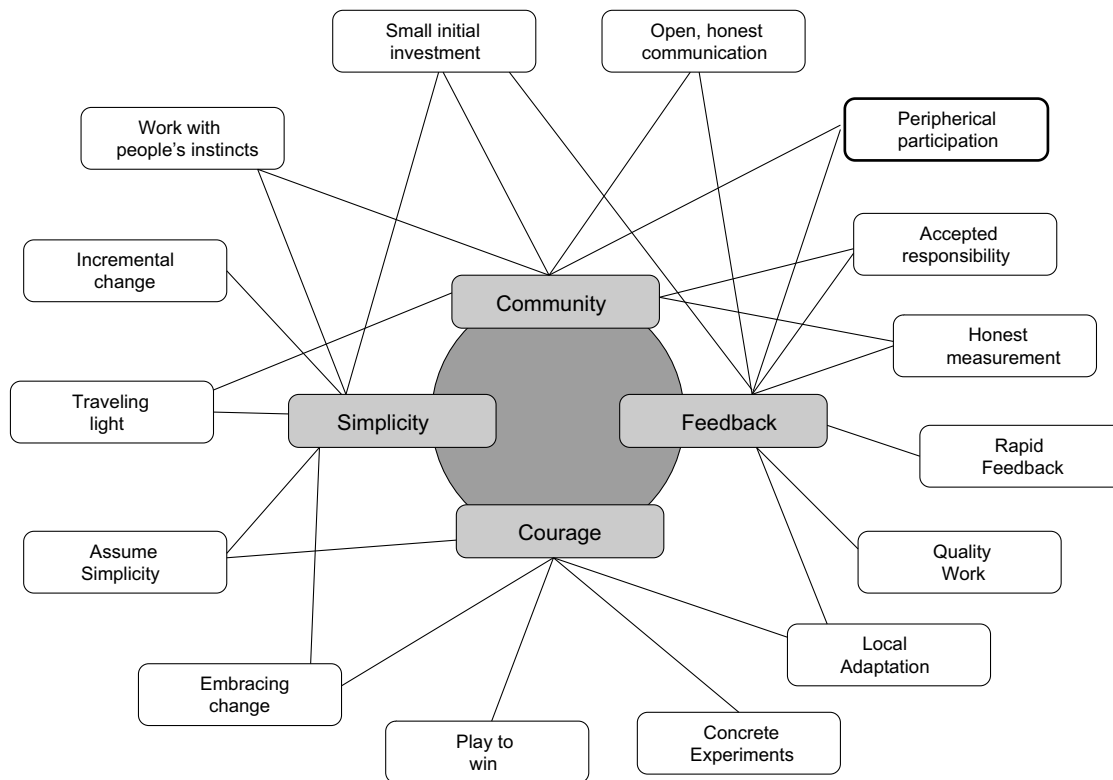


Fig. 6. The proposed values and the supporting principles of our Agile method. Some of these values and principles were redefined from the CoP and KA point of view (adapted from [17]).

5.2.2. Rapid feedback and concrete experiments

As a result of using a tool like the one suggested in the description of the *Feedback* value, the experiments with source code will be faster since the tool can generate it based on the KA architectural framework [3] and the KA-based model.

This tool should also perform automated tests and logical consistency verification on the KA-based model to prevent late costly modifications.

5.2.3. Embracing change

It is possible that much of the generated source code be abandoned to start the design of a new KA. This principle gives support to the value of *Courage*.

5.2.4. Peripheral participation

The method we are proposing considers two main communities of practice. People belonging to the first one are Users and Experts, while people belonging to the second one are Knowledge Engineers and Developers.

The interaction among members of both communities begins at the peripheral. In the first scenario, the CoP of Users and Experts accept a Knowledge Engineer or Developer as a new user of the community. He can get to the core of the CoP where knowledge is held by the Experts. This activity helps the engineers and developers to have a better notion of the development scope of the KABS.

The practices of *on-site domain expert* and *Joint KA Design* show the second scenario, when the Expert participates in the CoP of Knowledge Engineers and Developers. In this case, the interaction is also peripheral since the Experts gradually learn about programming or knowledge modeling techniques at the same time the KA-based model is created.

5.2.5. Traveling light

Our methodology promotes the manipulation of a KA through a design tool with an appropriate GUI. This GUI presents the KA-based model dimensions in an effective way. The KA is the main object to be manipulated during the KABS development process. Other artifacts can be used to support the development of the KABS like UML diagrams or electronic worksheets as long as the KA is kept as the main model.

5.3. Practices

Like in XP.K, our method also uses some XP practices for the parts of the KABS that do not involve knowledge. Other XP.K practices involving knowledge modeling are adapted to design KA-based models (Fig. 7).

5.3.1. Test in simulated or real world, constraint checking

We promote the use of Semantic Web languages to express the dimensions composing a KA-based model because they are being widely adopted in industrial applications and research activities. It also facilitates the diffusion of KA-based models over the Internet being able to be used by virtual CoPs. Inference engines like Pellet [23] or Racer [14] allow to perform consistency verification over an OWL-DL ontology.

Besides testing the OWL-DL ontology, the epistemological rules expressed in SWRL [19] can also be tested (the merging of OWL-DL and SWRL DL-Safe rules is implemented in Pellet). Finally, OWL-S [12] can be used to define the PSM. This language allows to describe semantic web services from the highest level (e.g. Inputs) to the lowest (execute web services). In this description, a set of expression constructs (e.g. If-Then-Else) are used to define the execution path for the semantic web service. This execution path

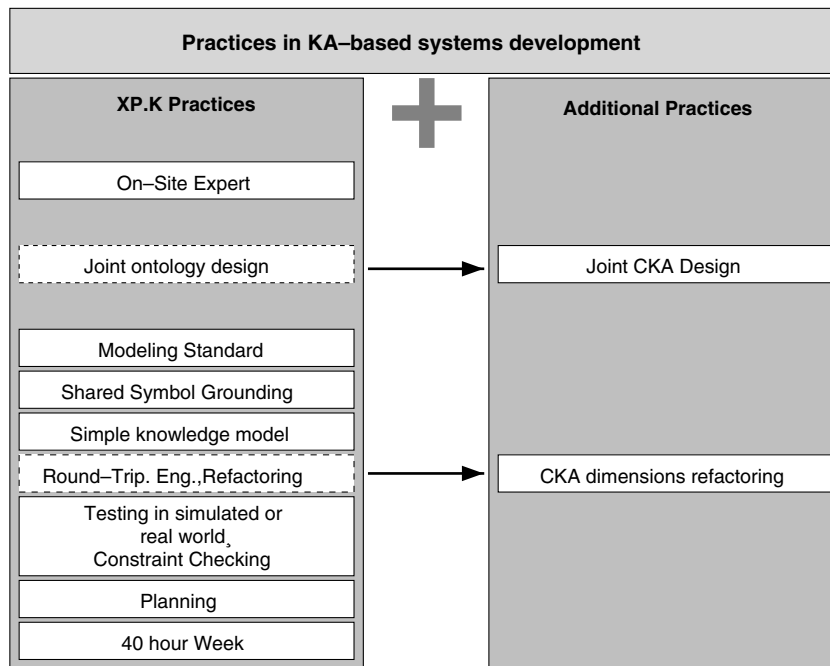


Fig. 7. The practices that compose our method are inspired by XP.K practices. Some of them were replaced like the case of *Joint KA Design* (adapted from [17]).

can be reused to define the sequence of steps in a PSM. Moreover, OWL-S engines ([22,13]) can execute these descriptions.

5.3.2. Shared symbol grounding

The KA serves as a metaphor for the development team. The metaphor is driven by the message the KA brings (e.g. in the case of *design by adaptation*, this message is partially given by the grammar elements “(low level property)”, “(high level property)”, “(recipe)”).

5.3.3. Simple knowledge model

A KA-based model includes different kind of knowledge models (e.g. ontological) which must be modeled keeping in mind the current needs. The practice of *Rapid Feedback* provides information about the consistency of these models to help guiding modeling.

5.3.4. Joint KA design

In XP.K, the practice of *Joint Ontology Design* was defined so the Knowledge Engineer and the Domain Expert were able to find a suitable representation for the domain ontology. In our case, this is translated to designing a KA.

The goal is to build a KA-based model composed by a grammar, its sentences and the PSM, formally expressing the ontological and epistemological CoP knowledge along with a method to process this elements.

The grammar is designed after and ontological and epistemological analysis of the application domain. It is derived from a domain-dependant non-formal vocabulary (e.g. the professional language) used by the Domain Experts. In other words, the KA-based model is a knowledge model specified by sentences using the designed language.

Entities, properties, relationships, processes are defined by the grammar of the language composing the intensional domain model.

In order to avoid the ambiguity of the model due to the use of the CoP's natural language, it is mandatory this language to be formal. The *Knowledge Engineer* promotes the participation of the *Experts* in both tasks: the activities defining the language (e.g.

grammar and semantics) and the specification of the initial KA based model to be implemented in the KABS and maintained by the *Expert Maintainer*.

5.3.5. Modeling standard

This is defined by both Knowledge Engineers and the Experts. Depends greatly on the professional language used by the COP's members and has an impact on the definition of the KA-based model.

5.3.6. Refactoring KA dimensions

In XP.K the practice of *Refactoring and Round-Trip Engineering* is defined. It involved techniques to generate source code from UML models and viceversa.

This cannot be done in our case since we are using OWL-DL as metamodel language. However, refactoring in the context of KABS can be performed in each dimension of the KA-based model and be of two types: (1) External: every dimension is an independent OWL-DL entity (e.g. ontology), it can be reused in the design of another KA, (2) Internal: Knowledge Patterns [6] can be useful to apply recurrent patterns in ontology modeling.

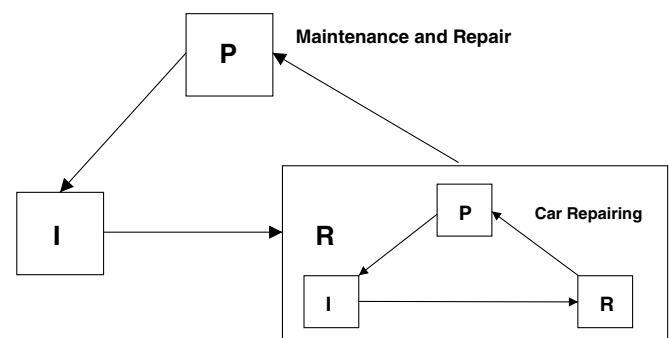


Fig. 8. The Maintenance and Repair KA reified in the car repairing domain.

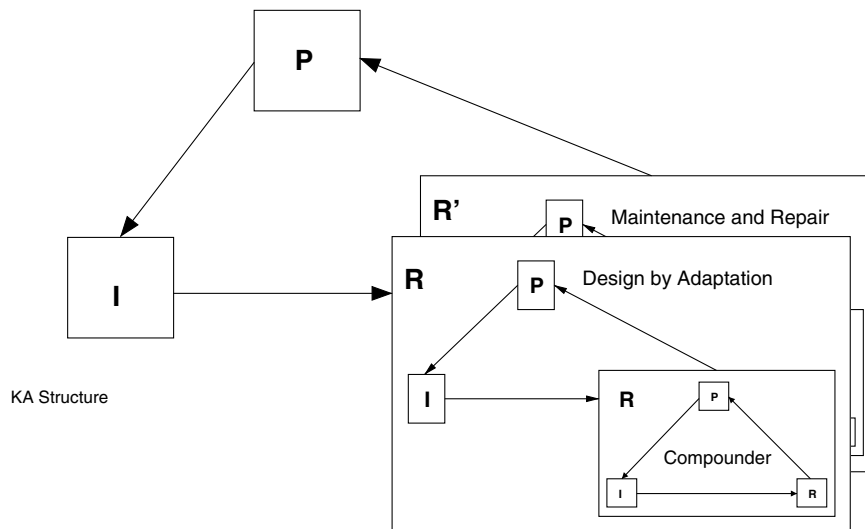


Fig. 9. The metaKA and the relationship with the set of KAs.

6. Discussion and future work

In this article, we have described two concrete KAs related to two different projects. Both its Design by Adaptation and the Maintenance and Repair KAs have guided us in the development of specific software systems, and these experiences have also formed the basis of our proposal of a general method to implement KA-based systems.

The theory of knowledge artifacts given in the first sections of this article can be further enriched. For instance, the relationship between concrete KAs and abstract KAs (such as the “Maintenance and Repair” KA) must still be further analyzed and explained: The abstract KA “Maintenance and Repair” has as its purpose P to diagnose and repair some kind of complex device. Its implementation R can be represented by the ontological elements present in the grammar and by the problem solving method describing how the sentences written in this grammar have to be considered. In the domain of car repair, this KA is reified with specific knowledge (e.g. car repair) to fulfill the purpose of fixing a car model. This results in a concrete KA with a different purpose P (fix damages in a car engine) and an implementation I (the software system or KA-based System). In Fig. 8 this relationship is shown.

There is also a less trivial relationship between all abstract KAs and the KA of KAs that we name “metaKA”. Fig. 9 shows this “metaKA”. Its purpose P is to create a KA and its implementation R is a “more concrete” KA (e.g. “Design by Adaptation”, “Maintenance and Repair”). We could continue to climb up on abstraction levels, until we reach a general domain like Design or Diagnosis.

At the first level of analysis shown in Fig. 8, we are implementing a tool to aid developers to build KA-based Systems. We believe that the utilization of a KA together with an Agile method to develop KA-based systems allows developers to build such systems more effectively and rapidly. One of the main problems in developing knowledge-based systems is the lack of tools to test the knowledge being elicited. In the case of a KA-based model this turns out to be a challenging problem because different models corresponding to different logical systems (e.g. SWRL and OWL) have to be tested. Some existing tools can perform partial tests (e.g. Protégé and SWOOP).

Moreover, we have explored the relationship between Design and AI [27] in order to produce a simple GUI that drives the developers in the elicitation of special-purpose knowledge more specifically, in the retrieval of the grammar and PSM descriptions from

the encoded KA and in the translation of them into the user-interface tool.

Due to the source code generation requirements defined by the method, we have decided to use the Eclipse Plugin architecture to implement a plugin to integrate source code generation and knowledge elicitation using Semantic Web languages. An initial description of this project can be found in <http://www.ime.usp.br/~gsalazar/kadesigner>.

A detailed rendition of this tool shall be presented in future publications.

References

- [1] S. Auer, The RapidOWL methodology – towards Agile knowledge engineering, in: WETICE, IEEE Computer Society, 2006.
- [2] S. Bandini, E. Colombo, G. Colombo, F. Sartori, C. Simone, The role of knowledge artifacts in innovation management, The Case of a Chemical Compound Designer CoP, Kluwer, B.V., Deventer, The Netherlands, 2003 (Chapter 1).
- [3] S. Bandini, E. Colombo, G. Vizzari, The role of knowledge artifacts in knowledge maintenance, in: B. Prasad (Ed.), Proceedings of the Second Indian International Conference on Artificial Intelligence (IICAL), Pune, India, December 20–22, 2005.
- [4] S. Bandini, S. Manzoni, Modeling core knowledge and practices in a computational approach to innovation process, in: L. Magnani, N. Nersessian (Eds.), Model-Based Reasoning: Scientific Discovery, Technologic, Values, Kluwer Academic/Plenum Publishers, 2002.
- [5] W. Cheetham, J. Graf, Case-based reasoning in color matching, Case-based reasoning research and development, in: Proceedings of the Second International Conference on Case-Based Reasoning, Springer-Verlag, Providence, RI, 1997.
- [6] P. Clark, J. Thompson, B. Porter, Knowledge patterns, in: A.G. Cohn, F. Giunchiglia, B. Selman (Eds.), KR2000: Principles of Knowledge Representation and Reasoning, Morgan Kaufman, San Francisco, 2000.
- [7] S. Craw, N. Wiratunga, R. Rowe, Case based design for tablet formulation, in: Proceedings of the Fourth European Workshop on Case Based Reasoning, Springer-Verlag, Berlin, 1998.
- [8] F.S.C. da Silva, J.A. Cullell, Knowledge Coordination, John Wiley & Sons, 2003.
- [9] R.L. Daft, Organization Theory and Design, ninth ed., South-Western College Publication, 2006.
- [10] T.H. Davenport, L. Prusak, Working Knowledge, second ed., Harvard Business School Press, 1998.
- [11] J.P. Delgrande, J. Mylopoulos, Knowledge representation: features of knowledge, Fundamentals of Artificial Intelligence: An Advanced Course (1986) 3–38.
- [12] Z.-J. Ding, J. Wang, C.-J. Jiang, Semantic Web Service Composition Based on OWL-S, in: SKG, IEEE Computer Society, 2005.
- [13] J. Giampapa, M. Paolucci, N. Srinivasan, R. Vaculin, The OWL-SVM project, 2007. Available from: <http://projects.semwebcentral.org/projects/owl-s-vm/>.
- [14] V. Haarslev, R. Möller, Racer: an OWL reasoning agent for the semantic web, in: Proceedings of the International Workshop on Applications, Products and Services of Web-based Support Systems, in conjunction with the 2003 IEEE/WIC International Conference on Web Intelligence, É Halifax, Canada, October 13, 2003. Available from: <http://www.sts.tu-harburg.de/r.f.moeller/racer/>.
- [15] R. Hilpinen, Artifact, the Stanford Encyclopedia of Philosophy (fall 2004 Edition), in: Edward N. Zalta (Ed.), 2007.

- [16] C.W. Holsapple, K.D. Joshi, Organizational knowledge resources, *Decision Support Systems* 31 (1) (2001) 39–54.
- [17] H. Knublauch, *An Agile Development Methodology for Knowledge-Based Systems Including a Java Framework for Knowledge Modeling and Appropriate Tool Support*, Ph.D. thesis, Fakultät für Informatik, Universität Ulm, 2002.
- [18] G.D. Michelis, *Aperto, moteplce, continuo*, Dunod, Milano, 1998.
- [19] M.J. O'Connor, H. Knublauch, S.W. Tu, B.N. Grosz, M. Dean, W.E. Grosso, M.A. Musen, Supporting rule system interoperability on the semantic web with SWRL, in: Y. Gil, E. Motta, V.R. Benjamins, M.A. Musen (Eds.), *Proceedings of the Fourth International Semantic Web Conference, ISWC 2005*, in: *Lecture Notes in Computer Science*, vol. 3729, Springer, Galway, Ireland, November 6–10, 2005.
- [20] W. Rammert, M. Schlese, G. Wagner, J. Wehner, R. Weingarten, *Wissensmaschinen: Soziale Konstruktion eines technischen Mediums Das Beispiel Expertensysteme*, Campus Verlag, Frankfurt, Germany, 1998.
- [21] G. Salomon, *Distributed cognitions: Psychological and Educational Considerations*, Cambridge University Press, Cambridge, UK, 1993.
- [22] E. Sirin, J.A. Hendler, B. Parsia, Interactive composition of semantic web services, in: *WWW (Posters)*, 2003.
- [23] E. Sirin, B. Parsia, B.C. Grau, A. Kalyanpur, Y. Katz, Pellet: a practical OWL-DL reasoner, *Journal of Web Semantics* 5–2 (2007).
- [24] A. Tiwana, *The Knowledge Management Toolkit: Practical Techniques for Building a Knowledge Management System*, Prentice Hall, 1999.
- [25] E. Wenger, *Communities of Practice: Learning, Meaning and Identity*, Cambridge University Press, Cambridge, 1998.
- [26] T. Winograd, *Bringing Design to Software*, Addison-Wesley, 1996.
- [27] T. Winograd, Shifting viewpoints: artificial intelligence and human-computer interaction, *Artificial Intelligence* 170 (2006) 1256–1258.