

# Kuaba Ontology: Design Rationale Representation and Reuse in Model-Based Designs

Adriana Pereira de Medeiros, Daniel Schwabe and Bruno Feijó

Dept. of Informatics, PUC-Rio, Rua Marquês de São Vicente 225,  
22453-900, Rio de Janeiro - RJ, Brasil  
{adri, dschwabe, bruno}@inf.puc-rio.br

**Abstract.** This paper presents the Kuaba Ontology, a knowledge representation model for Design Rationale described in an ontology definition language. The representation of this model in a specific ontologies specification language, such as OWL or F-Logic, allows attributing semantics to recorded Design Rationale content, and defining rules that enable performing computable operations to support the use of Design Rationale in the design process of new artifacts. In addition, we propose to support the software design process through the use of the semantic descriptions defined by formal models of the artifacts. Representing Design Rationale using an ontology definition language and the artifacts formal model, enables a type of software reuse at the highest abstraction level, where rationales are re-employed in designing a new artifact. This kind of reuse is possible in knowledge domains where there are formal models describing the artifacts, in particular, in the Software Design domain.

## 1 Introduction

Designing a software artifact typically involves understanding the problem being addressed, identifying possible solution alternatives, analyzing them, and deciding which solutions will be used to construct the final artifact. The final products of this process, the artifact and its specifications, represent but a fraction of the knowledge employed by designers during the design process. They represent the final solution chosen for the particular design problem, but do not represent, for instance, the reasons that led the designers to choose that one among the other available alternatives, and why the others were discarded. In other words, they do not capture the Design Rationale (DR) that led to the artifact in question.

DR is the reasons behind design decisions. However a more complete definition is proposed by J. Lee [1]: design rationales include not only the reasons behind a design decision but also the justification for it, the other alternatives considered, the tradeoffs evaluated, and the argumentation that led to the decision. In most cases the DR is not adequately documented, which leads to requiring a high degree of verbal communication among persons that must work with an artifact, in order to understand the reasoning followed by the designer. For instance, this is fundamental when maintaining a software artifact designed by another person, or when trying to reuse it in the context of a new design. This is true even in the case of a single person, since in

many cases, over a longer period of time, the designer himself may not recall all the rationale he himself used in the design of a particular artifact. Therefore, recording the DR during the design process is critical to allow its reuse.

There are several proposals in the literature for representing DR, such as IBIS [2], PHI [3], QOC [4] and DRL [5]. Most of them are incomplete or informal, not enabling machine-processable computations over the represented DR. Consequently, it is not possible to guarantee that the representation is consistent and even that it does actually provide some sort of explanation about the captured design. Furthermore, when applying them to formally defined artifacts (such as software), their informality prevents automatically taking into consideration alternatives prescribed by the design methods, as well as incorporating their restrictions. In other words, it is not possible to leverage the semantics of the artifact provided by the formal model that describes it.

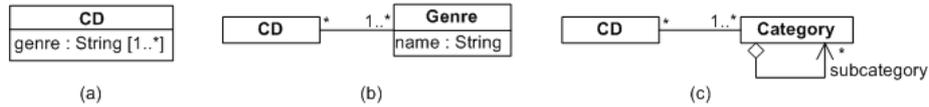
For many knowledge domains, particularly in software design, there are formal models that describe the artifacts and present semantic descriptions, which allow reasoning over the artifacts being produced. In this paper, this special type of design domain is called "model-based design". An example of such a formal model is the UML specification language [6] used to describe a class diagram. A formally defined DR representation may allow integrating the formal semantics of the artifacts being designed, and allows automated computations over such representations. When such representations are available in a distributed environment, it is possible to envisage the collaboration between designers with semi-automated support, where DR representations can be searched for, recovered and integrated during the process of designing a new artifact. Such availability can therefore be the basis for collaborative (and even participatory) design, among designers working with a given artifact.

The integration of different DRs is possible only if the following conditions are satisfied: the artifacts are built from the same type of formal model (e.g. two UML class diagrams); the DRs are used to represent the same domain of application (e.g. a CD catalogue) and the DR of the artifacts is represented using the same (or compatible) representation scheme(s) - e.g. an ontology vocabulary.

A semi-automated computational environment is being built to support designers through processing of formal DR representations. This environment uses the formal model for the artifact being designed to suggest design options at each step in the design, and records the corresponding choices made by the designer, using a special purpose description language that will be described later. Depending on the richness of the formal model of the artifact being designed, the system may suggest new alternatives, and also check the consistency of decisions made by the designer. Theoretically, when formal semantics for the artifacts are available, fully automated systems could be constructed to automatically synthesize artifacts, but this is neither the approach nor the focus taken in this paper. We explicitly require human intervention in defining design steps or operations in producing the final design.

Consider the following motivating example shown in Fig.1. This example shows three design options defined by different designers to model the "Genre" information item in an UML class diagram modeling a CD catalogue. In Fig.1-a, the designer decided to model "Genre" as an *attribute* with multiplicity one or more. In Fig.1-b, the designer decided to model "Genre" as a *class* that has an association with a *CD* class, and in Fig.1-c another designer decided to model a "Category" information item instead of "Genre" to represent the same kind of information. This designer

decided to model *Category* as a *class* with a self-relation of type aggregation to represent the subcategory concept.



**Fig. 1.** Design options to model Genre information item

Since these artifacts are described in the same formal model (UML), and refer to the same domain (CD catalogues), a fourth designer could retrieve the DR representations of these artifacts (for instance, in a distributed environment), and integrate both rationales to design a new class diagram for this domain, reusing existing (partial) solutions.

In this particular case, the fourth designer could consider the “*Genre*” and “*Category*” information items to be really the same, and thus integrate the DR representations for each. For instance, she may consider modeling “*Category*” as an aggregation, but also taking the idea of allowing multiplicity one or greater, taken from the other modeling alternative. Thus, this designer could incorporate into her design the reasoning (arguments for and against each alternative considered) used by the other designers, and add her own reasons as well, finally making her own decisions.

This DR enables a type of reuse at the highest abstraction level, where rationales are re-employed in designing a new artifact. Starting with an existing DR, the designer can review and extend it, adding new alternatives or making different choices with respect to already defined alternatives, generating a new DR. From this point of view, both software maintenance and evolution can be considered as simply a continuation of a previous design process, captured in a given DR.

This paper is a significant extension and expansion of [7], in which we approach and exemplify the different uses of the Kuaba<sup>1</sup> ontology to representing and reusing DR. In the remainder of this paper, we first present a DR representation model for software designs, using the Kuaba ontology. Next, we address the issue of how the formal semantics of the artifacts being described can be integrated with the design process, which seen as an instantiation process of the formal model for the artifact. From this point of view, the richer the semantics of this formal model of the artifact, the greater the degree of support automation that could be achieved. Next, we present some scenarios of use for these DR representations, and the operations needed to support their reuse when designing new artifacts. We conclude by discussing related work and pointing out further work, and drawing some conclusions.

<sup>1</sup> “Kuaba” means “knowledge” in Tupy-guarany, the language of one of the native peoples in Brazil.

## 2 Kuaba: The Design Rationale Ontology

As previously mentioned, it is desirable to represent DR in a formally precise and computable way. Ontologies are good candidates for this, since, as defined in [8], they represent “an explicit specification of a conceptualization”. In other words, they are knowledge representations, where a set of objects and their relationships are described through a defined vocabulary.

The Kuaba ontology describes a set of elements (classes, properties, relations and constraints) that express the DR domain. Our objective in proposing this ontology is to provide a vocabulary for DR described in an ontology definition language that allows attributing semantics to recorded DR content, and defining rules that enable performing computable operations and inferences on this content. The first version of the ontology was created using the Web Ontology Language (OWL) [9]. The current version is described in F-Logic [10] due to the availability of free inference engines.

The vocabulary described by Kuaba extends the argumentation structure of the Issue Based Information System (IBIS), whose approach for DR is to register the *issues* raised during design, the *positions* that address these issues and the *arguments* against or in favor of these positions. The extension enriches this argumentation structure by explicating the representation of the decisions made during design and their justifications, and the relations between the argumentation elements and generated artifacts. It consists also of integrating this argumentation structure with descriptions of the produced artifacts, and with information about the design history (when decisions were made, who made them, what design method was used, etc). Fig. 2 shows the elements of the vocabulary defined by the Kuaba ontology, using the UML notation to help visualization. Notice that such object oriented model is used only as a suggestion of illustration of ontology vocabulary; some relations and constraints were hidden to simplify the presentation.

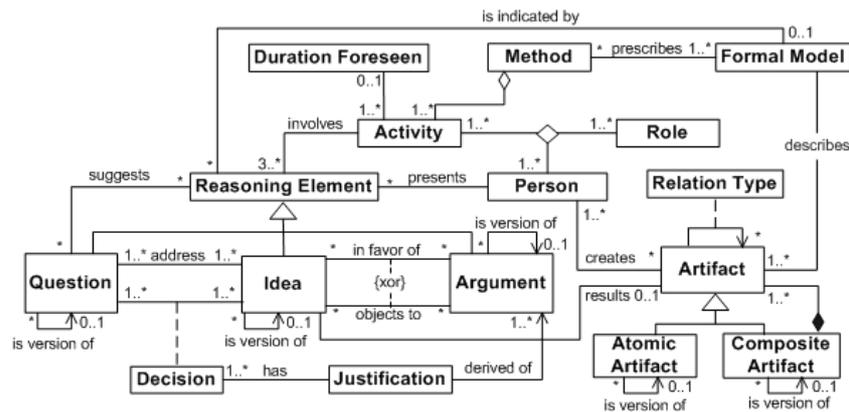


Fig. 2. The elements of the Kuaba Ontology Vocabulary

Briefly described, the Kuaba ontology vocabulary represents the people involved in a design activity and their respective roles. After defining the design method and

the activities that will be undertaken, the people involved in the design activity use reasoning elements for organizing and recording their solution ideas about the artifact that is being constructed. Similarly to IBIS, these reasoning elements represent the design problems (questions) that the designer should deal with, the possible solution ideas for these problems and the arguments against or in favor of the presented ideas. In Kuaba some of these elements are described according to the formal model of the artifact prescribed by the design method used. People involved in the artifact design make decisions about the acceptance or rejection of the solution ideas presented. Each decision must have a justification that explains the “*why*” it was made. Justification is always derived from one or more arguments presented during the design. The ideas accepted during the design process originate artifacts that can be either atomic artifacts or composite artifacts. All reasoning elements (*Question, Idea* and *Argument*) and artifacts have a “*is-version-of*” relation, representing the fact that any one of them may be based on an existing element. This element may be either part of a previous version of this same artifact, and therefore the design is actually evolving it, or part of a different design that is being reused in a new context.

Below we show a portion of the Kuaba ontology vocabulary shown in Fig. 2 expressed using F-Logic.

```
// CONCEPTS -----
question::reasoning_element.
idea::reasoning_element.

reasoning_element[hasText->STRING; hasCreationDate->STRING;
                  isInvolved->activity; suggests->>question;
                  isPresentedBy->person;isIndicatedBy->formal_model].

question[hasType->STRING; isAddressedBy->>idea; hasDecision->>decision;
         isSuggestedBy->>reasoning_element; isVersionOf->question].

idea[address->>question; results->artifact;
     isConcludedBy->decision; isVersionOf->idea].

decision[isAccepted->BOOLEAN; hasDate->STRING; isMadeBy->>person
         concludes->idea; hasJustification->justification].

// ALGEBRAIC PROPERTIES OF RELATIONS (INVERSE) -----
FORALL X,Y X[address->>Y] <-> Y[isAddressedBy->>X].
FORALL X,Y X[concludes->>Y] <-> Y[isConcludedBy->>X].
```

## 2.1 Representing Reasoning Elements and Decisions

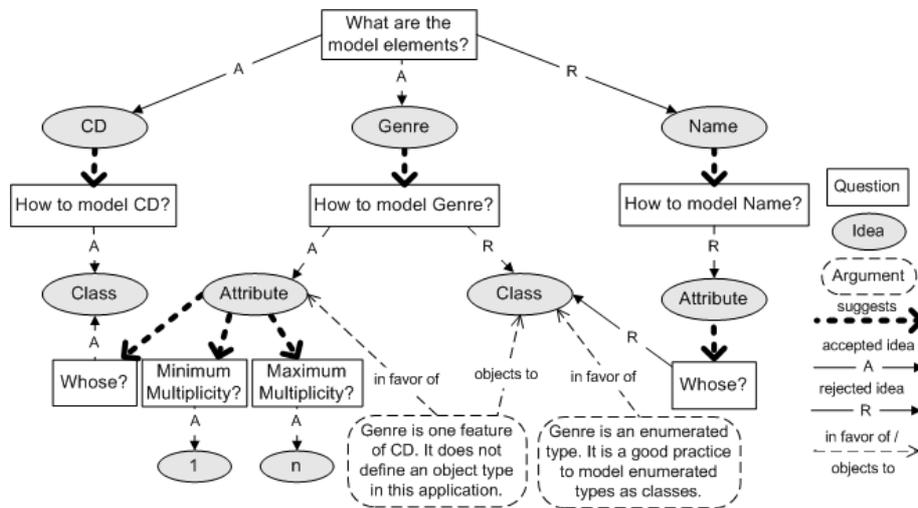
Normally, the first activity done by the designer in designing a software artifact is the choice of design method or process that will be used to achieve the design. When the designer chooses a design method or process, she indirectly determines the formal model(s) (specification language) that will be used to describe the artifact. For example, when a designer chooses the Unified Process [11] to achieve the artifact design she indirectly determines the formal model defined by the UML to describe this artifact.

The existence of a formal model for the artifact determines, to a great extent, the questions and ideas that the designer can propose, since they are pre-defined by this

model. In this sense, designing an artifact according to the method amounts to a stepwise instantiation of the formal model. Consider once again the motivating example shown in Fig. 1; if the designer has chosen the Unified Process, the DR will be expressed in terms of questions, ideas and arguments defined by the UML formal model for class diagrams.

The DR representation usually begins with a general question that establishes the problem to be solved. This general question can generate new questions that represent new design (sub) problems related to the main problem. For each question presented the designers can suggest ideas, formulating possible solutions to the problem expressed in the question.

According to the UML formal model, the first problem to be solved in designing a class diagram is the identification of its constituting elements. Applying the vocabulary described by Kuaba, this results in instantiating the “*Question*” class with the instance “*What are the model elements?*”. Fig. 3 shows a graphical representation we have created to help visualizing instances of the Kuaba ontology, showing the portion of the design regarding the alternatives to model “*Genre*” in the motivating example (Fig.1-a and Fig.1-b). In this representation, the root node is an initial question (represented as rectangles), “*What are the model elements?*”, which is addressed by the ideas “*CD*”, “*Genre*” and “*Name*”, represented as ellipses. Notice that these values are determined by the designer’s knowledge of the domain, or were extracted from the DR of a previous phase, requirements elicitation, which is not addressed in this paper.



**Fig. 3.** A portion of a DR representation regarding “Genre” in the motivating example (Fig. 1)

Once these first alternative ideas for the CD Catalogue model elements have been established, the designer must decide how each one of them will be modeled using the UML, to make up the final artifact, the class diagram. This next step is represented in

Fig. 3 by the “suggests” relation, which determines questions entailed by ideas – “*How to model CD?*”, “*How to model Genre?*” and “*How to model Name?*”.

The possible ideas that address these questions are determined by the UML formal model for class diagrams – elements can essentially be a class, an attribute, or an association. Accordingly, the “*Class*” and “*Attribute*” ideas linked to the “*How to model Genre?*” node are established as an instantiation of the UML formal model. Strictly speaking, the designer should consider all the other alternatives proposed by the UML, but for the sake of simplicity we have shown only these two. Since the “*Attribute*” idea, in turn, must be associated with a “*Class*” according to the UML model, the question “*Whose?*” is suggested, which in turn will be addressed by the idea corresponding to the class with that attribute it is. Similarly, the questions “*Minimum Multiplicity?*” and “*Maximum Multiplicity?*” are also defined by the UML model to be associated with the idea “*Attribute*”.

In this way, it is possible to envisage how the formal model “drives” the instantiation of the Kuaba ontology recording the DR. This formal model can be used by a support environment to suggest to the designer the possible Kuaba ontology instances that must be defined at each step of the process. The designer has to choose the desired alternative, and record the arguments for and against each option (represented as dashed rectangles in Fig. 3). It is through the “*Argument*” element that the designers can record the experiences and the knowledge that they are employing in the artifact design.

In Fig.3, we also show the final decision made, indicating each alternative answer to each question with an “A” (for accepted) or “R” (for rejected) label. Thus, the example represents the fact that the designer decided to accept the “*Attribute*” alternative to the question “*How to model Genre?*”, in detriment of the “*Class*” alternative.

The acceptance or the rejection of an idea as a solution to a question is recorded by the “*Decision*” element. Differently from IBIS, in our ontology the acceptance or rejection of an idea is represented as a property of the relation between the elements *Question* and *Idea*, as shown in Fig. 2. We consider that the acceptance or rejection of an idea is not an intrinsic property of the “*Idea*” element, but must be defined with respect to a certain “*Question*”, since the same idea can address more than one question, and be accepted for one and not for another.

The sub-graph of the DR made up of “*Question*” and “*Ideas*” is actually an AND/OR graph [12] that can be seen as a goal decomposition of the root node, which is always a “*Question*”. The “*Question*” class in the Kuaba ontology has a “*type*” attribute, with possible values “AND”, “OR” and “XOR”. The value “XOR” indicates that all ideas that address this question are mutually exclusive, meaning that only one idea can be accepted as a solution to the question. The value “AND” indicates that the designer should accept all ideas that address the question or reject all of them. Finally, the value “OR” indicates that various ideas can be accepted as a solution to the question. This kind of information allows us to define rules that can suggest decisions about the acceptance or not of the proposed solution ideas. For example, the software could suggest rejecting certain ideas based in the following rule: if an idea associated with a question of type “XOR” is accepted by the designer, then all other ideas associated to this question will be rejected. This rule is formulated in F-Logic as follows:

```

FORALL Q, I1, I2, D1, D2 D2[isAccepted->>"false"]
  <- Q:question[hasType->>"XOR"; isAddressedBy->>{I1,I2};
      hasDecision->>{D1,D2}]
  AND D1:decision[isAccepted->>"true"; concludes->>I1:idea]
  AND D2:decision[concludes->>I2:idea]
  AND NOT (equal (I1,I2)).

```

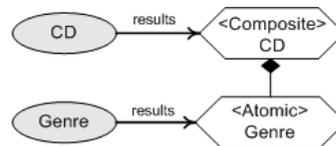
Notice that, if the decision to accept the idea of modeling “Genre” as an attribute was the first one made by the designer, a support system could apply the rule above, and automatically propose that the idea of modeling “Genre” as a class be rejected, given that there are only two ideas associated with this question. At this point, the designer also has the option of not accepting this suggestion, and revising the possible answers to the original question “What are the model elements?”, rejecting “Genre” altogether. In any case, the support environment can apply consistency rules defined by the Kuaba ontology, as well as those expressed for the particular formal artifact representation being used. Therefore, the order of accepting or rejecting an idea does not affect the represented rationale.

## 2.2 Representing Artifacts

A useful part of a DR representation must associate the designed artifact itself, or its components, to the corresponding design decisions that led to them being the way they are. This integrates the artifact description with the DR description. In Fig. 2, this is represented by the “*Results*” relation between “*Idea*” and “*Artifact*”.

An artifact corresponds to a final design solution, made up of a set of accepted ideas in the DR representation. Therefore, in a DR representation every artifact must be associated with at least one idea, and at least one of them must have been accepted. Clearly, an artifact cannot be associated with an idea that was rejected.

In the Kuaba ontology vocabulary, artifacts are represented by two classes, *Atomic Artifact* and *Composite Artifact*. For example, in the UML formal model a class can be seen as an aggregation of attributes, and therefore an element modeled as a class can be modeled as a composite element. This is the case, for example, of class “*CD*” in Fig. 3. In Fig. 4 we show an example of an artifact representation after the decisions shown in Fig. 3 have been made. In particular, notice that the “*Genre*” element is represented as an *Atomic Artifact*, since the designer decided to model it as an attribute of the CD class.



**Fig. 4.** Examples of Artifacts

### 3 Using the Representations of Design Rationale

As exemplified previously, representing DR in a more formal language and using the artifact formal model permits us to assign semantics to recorded content, enabling processing this content by computable operations. These operations, in turn, give support to new scenarios of use of DR in model based designs of software artifacts.

Consider the scenario where a designer wishes to construct a class diagram to represent information that will be used in a CD Store application. Since the online stores domain is a common domain in software design, the designer decides to perform a search for existing designs in a distributed environment, trying to find similar artifacts, before she begins a new design. As a result, she finds two different class diagrams for the CD domain with their respective DR representations. After receiving the search result and analyzing the artifacts found, the designer decides to reuse these artifacts taking advantage of the knowledge already used by other designers that is recorded in the available DR representations. In this scenario the designer can reuse a found artifact in two ways. She can import its DR representation and begin her design based on it; or she can integrate the DR representations of both found artifacts to compose a more complete solution of design.

An example of the first type of reuse is the artifact evolution process, in which a copy of a DR representation of the artifact is used as a starting point for the design, and subsequently modified, generating a new artifact (the new version of the original one). Incorporating a DR representation of an artifact means to import a set of already defined reasoning elements into the design that is being performed. Imagine that the designer had selected the first class diagram found and that in this diagram the author had considered the “*Genre*” and “*Record Label*” information items, as Fig. 5 shows.

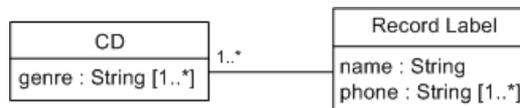
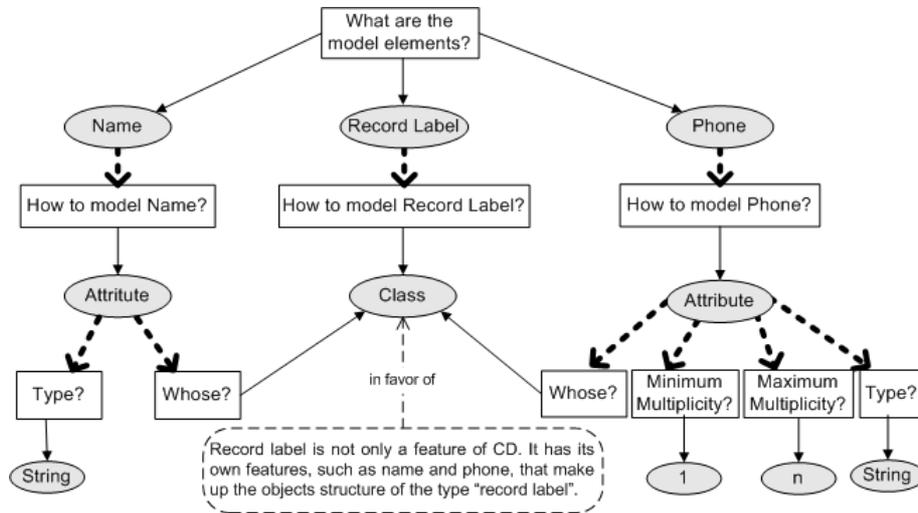


Fig. 5. Original Class Diagram

In this diagram the author decided to model the “*Genre*” information item as an attribute of a CD class and the “*Record Label*” item as a class. Fig. 6 shows the graphical representation of the reasoning elements incorporated into the new design, after the designer has selected the first diagram found. These elements represent the reasoning used by the author of the artifact to model the “*Record Label*” information item in his class diagram. According to this DR representation, the author considered the idea of designing “*Record Label*” as a class and the ideas of designing the “*Name*” and “*Phone*” information items as attributes of this class.



**Fig. 6.** Example of DR about the *Record Label* information item

Notice that the decisions made for the imported artifact design are not incorporated to new design (there are no arrows labeled “A” and “R”). This reflects the fact of the incorporation of an existing DR representation and its modification represents a new design, or a design continuation, whose DR should also be recorded. In this new design, new decisions should be made according to the new designer’s objectives, although the decisions previously taken by the author of the imported DR can be obtained from the DR representation of the original artifact.

In this first example we can suppose that, after analyzing the solution ideas incorporated into the new design, the designer had access to the idea of designing “*Record Label*” as an element of her class diagram, an idea that she had not considered before. We can also assume that the designer decided to modify the DR incorporated into her design including a new solution idea, the idea of designing the element “*Record Label*” as an attribute of a *CD* class.

Observing the decisions shown in Fig. 7 we can conclude that the designer decided to model the “*Record Label*” element as an attribute of the *CD* class and to reject the solution ideas for the “*Name*” and “*Phone*” elements.

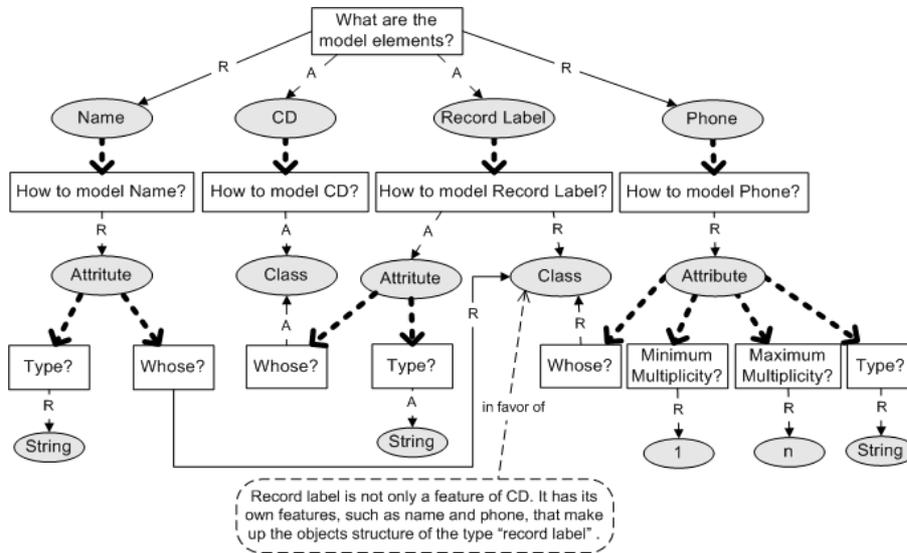


Fig. 7. DR for the new artifact, the “Record Label” attribute definition

Fig. 8 shows the design solution used by the designer after reasoning about the design of the “Record Label” element.

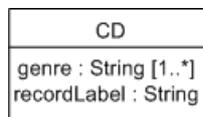
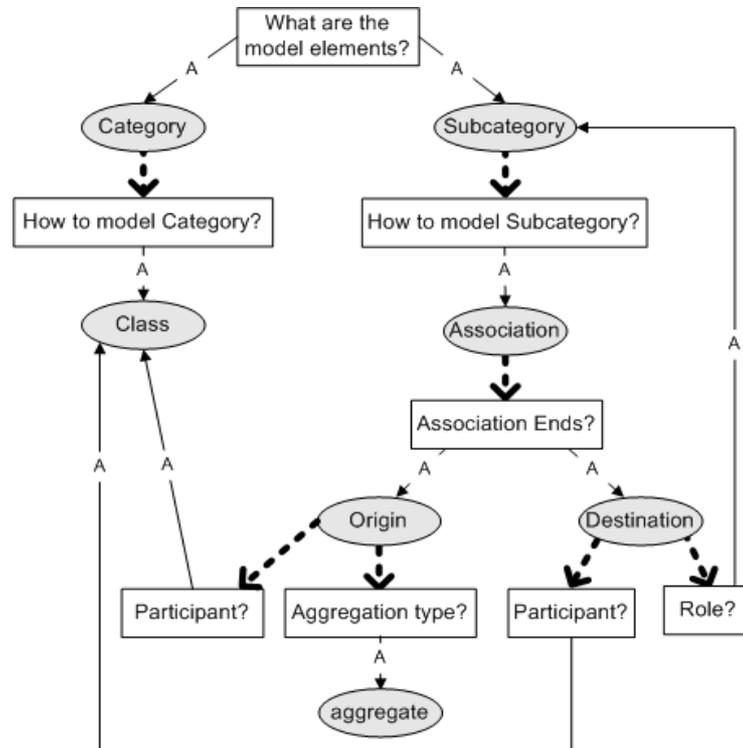


Fig. 8. Newly designed artifact, the “CD” class definition

To exemplify the second way of reusing artifacts, the reuse through integration of existing DR representations, we will use the motivating example presented in section 1. In this second form of reuse, the designer selects the three artifacts found by the search service and, after analyzing the DR of these artifacts, she performs the integration of their respective representations.

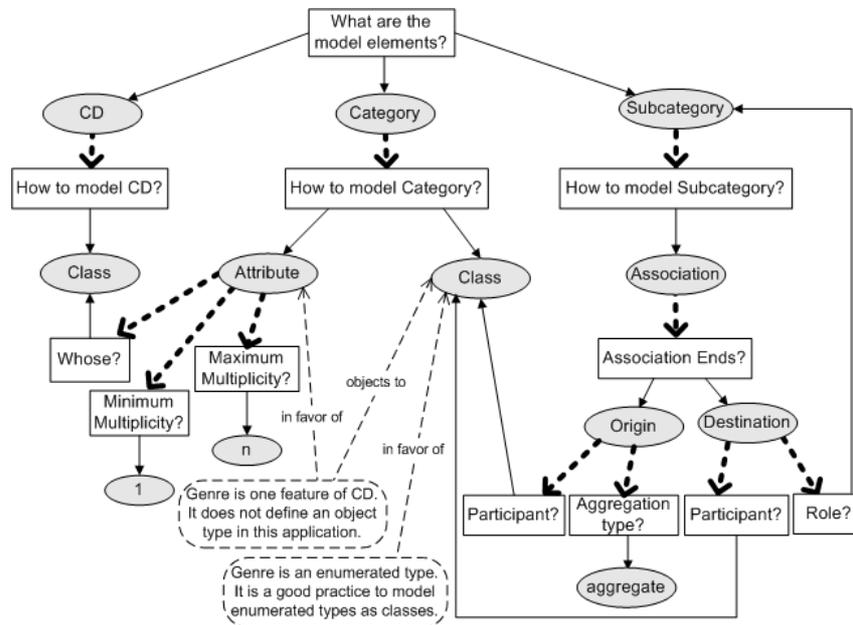
Part of the DR of the first artifact selected by the designer is shown in Fig. 3. Examining this DR, the designer verifies that the author of the select artifact decided to model the “Genre” element as an attribute of the CD class after considering the use of attribute multiplicity to model more than one genre for the same CD.

Continuing her analysis, the designer consults the DR of the third artifact found, shown in Fig. 9, and verifies that the author of this artifact, instead of considering an element “Genre” in the class diagram, considered the element “Category” and decided to model this element as a class. Besides the “Category” element, the author also decided to include the element “Subcategory” in his diagram to model the concept of subcategories as an aggregation of the Category class.



**Fig. 9.** Example of DR about the Category Element

Next, the designer establishes that the “*Genre*” and “*Category*” elements in each of the DR representations are actually the same concept in the CD domain, and formally specifies this identity so that a computational environment can integrate the reasoning elements related to them. After specifying the identity between these elements, the designer defines that the representation of the second artifact (Fig. 9) will be used as the basis for the integrated representation, and performs the integration of the DR representations of the selected artifacts. Fig. 10 shows the graphic representation of part of the DR that was recorded after the integration of the rationales exemplified in figures 3 and 9.



**Fig. 10.** Example of integrated DR

Observing the DR integration result shown in Fig. 10, we can notice that the solution idea “*Attribute*” that addresses the question “*How to model Genre?*” in Fig. 3 and all other questions, solution ideas and arguments represented there, were automatically included in the sub-tree of the question “*How to model Category?*”. With this integrated DR, the designer can begin her design evaluating the solution ideas already used taking advantage of the experiences of other designers, expressed in the arguments and decision justifications included in the DR. Based on this knowledge, she can modify the DR obtained and make new decisions about how her new class diagram will be designed.

The DR represented with the Kuaba ontology vocabulary could also be used in scenarios that involve cooperation. We believe that the support operations necessary for the reuse of DR in cooperation scenarios are the same operations necessary in the scenarios presented previously. In other words, the operations just described can be applied in exactly the same way when integrating partial solutions produced by team members in a cooperative design effort. We will discuss the necessary operations on DR representations next.

### 3.1 Operations

The use of the DR representations requires different kinds of operations on the recorded content. The explicit and semantic representation of DR in a language formally defined and specifically designed for the description of ontologies, allows these operations to be computable by engines to support the design of new artifacts.

The operations over DR representations can be grouped into queries, operations for manipulating an existing DR representation and operations for integrating of two or more DR representations (instances of the Kuaba ontology)

The first group allows formulating relevant questions about the design process and about the produced artifact. For example, we can formulate questions such as “*What were the solution ideas considered during the Genre artifact design?*”, “*Why the decision of designing the Genre artifact as an attribute was made?*”, “*Who presented the solution idea adopted for the Genre artifact?*”, or still “*Is artifact X related to artifact Y?*” The queries are performed according to the relations semantics and constraints defined by the Kuaba ontology vocabulary.

The operations for the manipulation of the existing DR representations are basically the same operations necessary to represent the reasoning elements (*Question, Idea* and *Argument*) produced by the designer during the design of a new artifact. These operations involve the creation and destruction of instances of the classes and properties defined in the Kuaba ontology. They are implemented in the majority of available ontology manipulation tools, such as Protégé [13].

The operations for the integration of two or more DR representations involve more specific treatments. These operations resemble the operations necessary to perform ontology alignment [14]. However, the operations for the integration of the DR representations defined in this work differ from the usual alignment operations, since they involve matching instances of the same ontology (Kuaba) and not the matching of taxonomies and instances defined in different ontologies. The types of operations identified for the integration of two or more DR representations are: search, copy, union and substitution.

Search operations allow the designer select which elements of the representations considered for the integration will be included in the integrated representation. For example, the designer could provide a question, and request that the search tool recover only the ideas that have arguments in its favor that are answers to this question.

Union operations allow joining the reasoning elements described in the representations considered in the integration to generate a new representation. These operations should take into account the identity specifications and the definition of the base representation previously defined by the designer. These operations can be implemented in different ways, allowing the designer to determine how the union of elements will be performed. One way would be to permit the designer to specify which parts of the representations considered should be integrated. For example, she could define the *Question* element that would be the root of the union of the representations. Or still, to allow her to restrict the elements considered during the integration, such as, for instance, requiring the union to consider only the ideas that were accepted in their respective representations.

Finally, the substitution operations allow the designer substitute an element in one representation by a corresponding element from another representation. This operation can be used for example when the designer desires to use the DR of a single representation, but she desires to substitute one of its elements by an element specified in the other representation.

Some of these operations make use of inference rules to automate part of the designer work in recording the DR of the artifact that she is building.

## 4 Related Work

Procedural Hierarchy of Issues (PHI) extends IBIS by simplifying the relations among issues by using the “serve” relationship only. In addition, it provides two methods to deal with design issues: deliberation (to give answers to the issue) and decomposition (to break down the issue into a variety of subissues). Differently the Kuaba ontology, PHI does not represent explicitly the decisions made during design and its justifications. Plus, it is not integrated with the artifacts descriptions and other information about the design process.

The Potts and Bruns [15] model relates entities in existing software engineering methods to IBIS-based deliberation. This model was extended by [16] in the creation of the Decision Representation Language (DRL). Similarly to the Kuaba ontology, the key difference from other DR representations, namely the integration with software engineering methods, is achieved through derivation of artifacts from alternatives. However, the Potts and Bruns model and the Kuaba ontology differ in the way they use software engineering methods. In Potts and Bruns model, the generic model’s entities are refined to accommodate a particular design method’s vocabulary for deriving new artifacts. For example, a new entity specific to the used design method is incorporated into the IBIS model. In Kuaba ontology the vocabulary of the design method is used in the creation of instances of the reasoning elements (*Question* and *Idea*), which allows to automate the generation of part of the values that would be informed by the users during design.

Although the works presented in [17] and [18] approach DR specifically for software engineering and focus on the (re)use of DR, they are not directly comparable to our work because they do not address the integration of DR representations to create new software artifacts. Furthermore, these works do not consider computable operations as a support for the reuse of software artifacts.

## 5 Conclusions

In this paper we have proposed a new way of reusing artifacts in Software Design domain. To permit a more effective reuse of DR, we have presented the use of the formal models of artifacts to represent DR using the vocabulary defined in Kuaba ontology. This vocabulary represented in a specific ontology specification language allows formulating queries on the recorded DR and to define a set of rules and operations to support the use of DR in designing new software artifacts.

One of the problems related to DR representation is the use of the formalism by the people involved in design process. We believe that the use of formal models of artifacts in a software development environment can facilitate the DR capture, since it permits automating part of generation of DR representations. Therefore, the large amount of data produced in DR representations of actual designs is significantly hidden from the designer through the use of automated support.

Our current research includes: the implementation of the operations defined in this paper to validate the reuse of software artifacts through the integration of existing DR representations; and the investigation of the use of the Kuaba ontology to represent

DR also in domains where there are no well defined formal models to describe artifacts. The operations are being implemented using the Flora-2 language<sup>2</sup> that translates F-Logic into tabled Prolog code.

## References

1. Lee, J.: Design Rationale Systems: Understanding the Issues. IEEE Expert Volume 12, No. 13 (1997) 78-85
2. Kunz, W., Rittel, H. W. J.: Issues as Elements of Information Systems. Institute of Urban and Regional Development Working Paper 131, Univ of California, Berkeley, CA, (1970)
3. McCall, R. J.: PHI: A conceptual foundation for design hypermedia. Design Studies, No.12 (1) (1991) 30-41
4. MacLean, A. et al.: Questions, Options, and Criteria: Elements of Design Space Analysis. Human-Comput. Interaction, No. 6 (3-4) (1991) 201-250
5. Lee, J., Lai, K.: What's in Design Rationale. Human-Comput. Interaction, No. 6 (3-4) (1991) 251-280
6. OMG: Unified Modeling Language Specification version 1.5. March (2003)
7. Medeiros, A. P., Schwabe, D., Feijó, B.: A Design Rationale Representation for Model-Based Designs in Software Engineering. In Proceedings of the CAiSE'05 Forum, Porto, Portugal, June (2005) 163-168
8. Gruber, T. R.: A Translation Approach to Portable Ontologies. Knowledge Acquisition, No. 5 (1993) 199-220
9. W3C: OWL Web Ontology Language Overview. W3C Recommendation. February (2004)
10. Kifer, M., Lausen, G.: F-Logic: A Higher-Order Language for Reasoning about Objects, Inheritance and Scheme. ACM SIGMOD May (1989) 134-146
11. Jacobson, I., Booch, G., Rumbaugh, J.: The Unified Software Development Process. Reading, Mass.: Addison-Wesley, (1999) 463p
12. Nilsson, N.: Principles of Artificial Intelligence. Morgan Kaufman Publishers (1986) 476p
13. Noy, N. F. et al.: Creating Semantic Web Contents with Protégé-2000. IEEE Intelligent Systems Vol. 16, No 2, Special Issue on Semantic Web, March/April (2001) 60-71
14. Doan, A. et al.: Learning to Map between Ontologies on the Semantic Web. In Proceedings of the 11<sup>th</sup> World Wide Web Conference (WWW2002), Honolulu, Hawaii, USA, May (2002)
15. Potts, C., Bruns, G.: Recording the Reasons for Design Decisions. In Proceedings of 10<sup>th</sup> International Conference on Software Engineering, Singapore (1988) 418-427
16. Lee, J.: Extending the Potts and Bruns Model for Recording Design Rationale. In Proceedings of the 13<sup>th</sup> International Conference on Software Engineering, Austin, TX (1991) 114-125
17. Pena-Mora, F., Vadhavkar, S.: Augmenting Design Patterns with Design Rationale. Artificial Intelligence for Engineering Design, Analysis, and Manufacturing (1996) 93-108
18. Burge, J., Brown, D. C.: Rationale Support for Maintenance of Large Scale Systems. Workshop on Evolution of Large Scale Industrial Software Applications (ELISA), ICMS'03, Amsterdam, NL (2003)

---

<sup>2</sup> <http://flora.sourceforge.net/>