Artificial Intelligence for Engineering Design, Analysis and Manufacturing (2008), *22*, 399–419. Printed in the USA.

doi:10.1017/S0890060408000279

Kuaba approach: Integrating formal semantics and design rationale representation to support design reuse

ADRIANA PEREIRA DE MEDEIROS AND DANIEL SCHWABE

TecWeb Lab, Department of Informatics, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brasil

(RECEIVED May 1, 2007; ACCEPTED May 9, 2008)

Abstract

This article presents Kuaba, a new design rationale representation approach that enables employing design rationale to support reuse of model-based designs, particularly, software design. It is shown that this can be achieved through the adoption of an appropriate vocabulary that allows design rationale representations to be computationally processed. The architecture and implementation of an integrated design environment to support recording design rationale using Kuaba is also shown. The Kuaba approach integrates the design rationale representation model with the formal semantics provided by the metamodel of the design method or modeling language used for describing the artifact being designed. This integration makes the design rationale representations more specific according to the design methods and enables a type of software design reuse at the highest abstraction level, where rationales can be integrated and reemployed in designing a new artifact.

Keywords: Design Rationale Processing; Design Reuse; Design MetaModel; Integrated Support Systems; Ontology

1. INTRODUCTION

Design rationale (DR) is an explanation of why an artifact, or some part of an artifact, is designed the way it is (Lee, 1997). The research developed in the DR area seeks to provide models and tools that allow explicitly recording the reasons behind design decisions.

Currently, there is no consensus in the literature about the definitions of design. Different definitions can be found in Simon (1981), Schön (1983), Goel and Pirolli (1989), Hubka and Eder (1996), and Winograd (1996). In the scope of this work, design is an activity carried out to create an artifact according to a method or process. We specifically deal with model-based design that can be seen as an instantiation process of a metamodel, in which the produced artifacts are models, instances of this metamodel.

DR has a potential value for supporting design reuse, because it prevents the experience and the knowledge invested in a design from being lost. All the experience acquired during a design can be transmitted and augmented by the use of recorded DRs in new designs. Nevertheless, despite much research, DR has not been very much used for software design. One of the reasons is the time consumption and the cost generally required for the capture and representation of DR. This can be explained by the lack of a representation approach that enables the development of an integrated tool that supports the capture, representation and use of DR as part of the software design process.

Generally, the representation approach determines how DR can be captured and used in new designs. Argumentation-based approaches have been extensively used to represent DR, by providing a structure to indicate which decisions were made (or not), and the reasons behind them. Although there are several argumentation-based approaches for representing DR, such as IBIS (Kunz & Rittel, 1970), DRL (Lee, 1991), QOC (MacLean et al., 1991), and TEAM (Lacaze, 2005), most of them generate incomplete or informal representations, not enabling the effective use of DR in the design of new artifacts. Furthermore, when applying them to formally defined artifacts (such as software artifacts), 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

Reprint requests to: Adriana Pereira de Medeiros, TecWeb Lab, Department of Informatics, Pontifícia Universidade Católica do Rio de Janeiro, Rua Marquês de São Vicente 225, 22453-900, Rio de Janeiro, RJ, Brasil. E-mail: adri@inf.puc-rio.br

the artifact provided by the formal model that describes it. This informality severely hinders the computational processing of the recorded rationale and its use in new designs.

This article describes the Kuaba[1] approach (Medeiros, 2006), a new approach for representing DR that integrates the DR representation model with the formal semantics provided by the metamodel of the design method or modeling language used for describing the artifact being designed. The main objective of this approach is to permit the computational processing of DR for supporting the reuse of model-based designs, particularly, software design. This approach proposes the Kuaba ontology (Medeiros et al., 2005) as a formal representation model for DR and the use of the artifacts' formal semantics, as defined by the design methods, as part of the instantiation process of this ontology.

A formally defined DR representation, enriched with the formal semantics of the metamodel that describes the artifacts being designed, allows attributing explicit meaning to the recorded content, making it easier for computers to automatically process and integrate the recorded knowledge. This enables a new type of design reuse, in which rationales can be integrated and reemployed in designing a new artifact.

This article also presents the conceptual architecture of an integrated design environment for supporting the capture, representation, and use of DR using the Kuaba approach. Some layers of this architecture are being implemented in the HyperDE+DR environment (Santos, 2007) that is also presented in this article. Different from other systems to support DR, such as Compendium (Conklin et al., 2003) and DRed (Bracewell et al., 2004), the integrated environment proposed in this work aims at supporting the semiautomatic capture and representation of DR based on the metamodels' formal semantics. In addition, it also aims at supporting design reuse by computationally processing the recorded DR when integrating partial designs into a new design.

In the remainder of this article, we first present the Kuaba approach and address the issue of how the formal semantics of the metamodel that describes the artifacts can be integrated with the representation model for recording DR. Next, we present the conceptual architecture of the integrated design environment and the operations to support designers through processing of these representations. The HyperDE+DR environment that implements part of this architecture is also presented. Finally, we conclude by discussing related work, pointing out further work, and drawing some conclusions.

2. THE KUABA APPROACH

Kuaba is an argumentation-based approach for representing DR. Its main objective is to support the use of DR in the reuse of model-based designs, particularly, software design. The Kuaba approach integrates the DR representation model defined by the Kuaba ontology with the formal semantics of the artifacts provided by the metamodel of the design method or modeling language that describes them. This integration is performed by the use of this formal semantics in the instantiation of the Kuaba ontology described in F-Logic (Kifer & Lausen, 1989), a formal language for ontology specification.

The DR representation using a formal language and incorporating the metamodels' formal semantics allows attributing semantics to the captured DR content and performing inferences and other computable operations on this content. In this way, all knowledge recorded during an artifact design can be computationally processed and used in the design of new artifacts.

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 taken in nor the focus of this article. Kuaba explicitly requires human intervention in defining design steps or operations to produce the final design.

2.1. The Kuaba ontology

The Kuaba ontology formally specifies a knowledge representation model for supporting the recording of DR in model-based designs. It provides a vocabulary and a set of rules described in F-logic, which permits defining computable operations to support the reuse of designs by the processing and integration of their rationales.

The Kuaba ontology vocabulary describes a set of elements (classes, properties, relations and constraints) for representing DR. It extends and enriches the IBIS argumentation structure by explicating the representation of the decisions made during design and their justifications. The extension includes also the integration of this argumentation structure with descriptions of the artifacts produced, and with information about the design history (when decisions were made, who made them, what design method was used, and so forth). Figure 1 shows the elements of the vocabulary defined by the Kuaba ontology, using an UML-like graphical notation to help visualization (see http://www.uml.org). Notice that such object oriented model is used only as a suggestion for presenting the ontology vocabulary; some relations and constraints were hidden to simplify its presentation.

Briefly described, the Kuaba ontology vocabulary represents the reasoning elements used by designers during the design process, their decisions, information about the artifacts that results from these decisions, and information about the formal models used to specify the produced artifacts. Similarly to IBIS, the reasoning elements represent the design problems (questions) that the designer should deal with, the possible solution ideas for these problems and the arguments for or against the presented ideas. These arguments record the experiences and the knowledge that the designers have employed in the artifact design.

In the Kuaba ontology vocabulary, each element has a set of properties and relations that compose the structure of the rationale developed by the designers during the design. For

[1] *Kuaba* means "knowledge" in Tupy-guarany, the language of one of the native peoples in Brazil.

https://doi.org/10.1017/S0890060408000279 Published online by Cambridge University Press

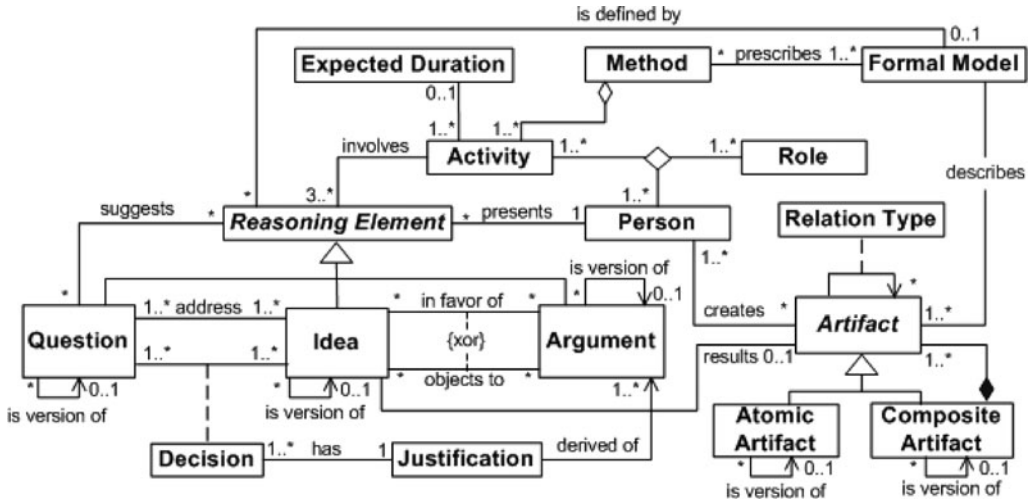

Fig. 1. The elements of the Kuaba ontology vocabulary.

example, the element *Question* has a "type" property, with possible values "AND," "OR," and "XOR." The value "XOR" indicates that all ideas that address this question (i.e., are possible answers) 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.

The acceptance or the rejection of an idea as a solution to a question is recorded by the "Decision" element. Differently from other DR notations, in the Kuaba 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 Figure 1. Thus, the acceptance or rejection of an idea is not an intrinsic property of the "Idea" element. It must be defined with respect to a certain "Question," because the same idea can address more than one question, and be accepted for one and nor for the other. Each decision must have a justification that explains "why" an idea was accepted or rejected as a solution for a particular question. Justification is always derived from one or more arguments presented during the design activity. Although a justification typically includes the arguments for or against the presented ideas, justification and argument are represented as two distinct elements in the Kuaba ontology. This allows to define and represent the final justification of a decision that can, for instance, explicate how the arguments for or against the accepted idea were consolidated.

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. The relation between "Idea" and "Artifact" integrates the artifact description with the DR description, making the rationale representation useful in the evolution of this artifact or in its reuse during the creation of new artifacts. So, it is possible to analyze the design decisions related to the artifact to see all the reasoning used by the designer to achieve the final design of this artifact.

Artifacts are represented by two elements: *atomic artifact* and *composite artifact*. These elements are instantiated according to formal semantics of the artifact being produced. For example, in the metamodel of the Unified Modeling Language (UML; OMG, 2003) a class can be seen as an aggregate of attributes and, therefore, an information item modeled as a class can be represented as a composite artifact.

The Kuaba ontology vocabulary also has elements to record the design method used by the designers, the activities prescribed by this method, the people involved in each design activity and their respective roles. A person can present one or more reasoning elements, make decisions about the solution ideas considered and produce the final artifact, according to the formal model that describes it. This formal model is, in turn, the metamodel prescribed by the design method or modeling language used by the designer. Recording this information facilitates the processing and the interpretation of the represented rationale, providing part of the context in which the artifact was designed. This helps to better understand the reasoning employed in a design, because the majority of the questions and solution ideas is defined by the metamodel prescribed by the design method. Moreover, recording information about the artifacts formal models allows attributing semantics to the DR representation, improving the consistency verification and processing of the rationale by a computational environment.

Finally, all reasoning elements (*Question*, *Idea*, and *Argument*) and artifacts have an "is-version-of" relation, representing the fact that any one of them may be based on a previously existing element. This element may be either part of a previous version of the artifact, and therefore, the design is actually evolving it, or part of a different design that is being reused in a new context.

https://doi.org/10.1017/S0890060408000279 Published online by Cambridge University Press

We show below a portion of the Kuaba ontology shown in Figure 1 expressed using F-Logic.

```
// CONCEPTS ------------------------------------------------------
question::reasoning_element.
idea::reasoning_element.
reasoning_element[hasText->STRING; hasCreationDate->STRING;
                  isInvolved->activity; suggests->>question;
                  isPresentedBy->person;isDefinedBy->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].
      …
// RULES (INVERSE RELATIONS) --------------------
FORALL X,Y X[address->>Y]  <-> Y[isAddressedBy->>X].
FORALL X,Y X[concludes->>Y] <-> Y[isConcludedBy->>X].
      …
//  AXIOMS ------------------------------------------------------
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)).
      …
```

The Kuaba ontology portion shown above contains the specification of some rules and axioms used in inferences and validations on the recorded DR. The first rules specify inverse relations of some properties defined in the Kuaba ontology vocabulary. For example, the "isAddressedBy" relation is defined as the inverse of the "address" relation between the *Question* and *Idea* elements. These rules are used by inference engines to guarantee correct answers for the queries formulated by the designers about the recorded rationale. For example, given the relation "idea A address question X" in a DR representation, when necessary, a system can apply the first rule above to infer that question "X" is addressed by idea "A," even though this last relation is not explicitly specified in this representation.

The second rule (axiom) is defined to support decisions for questions of the type "XOR." According to this rule, if an idea associated with a question of type XOR is accepted by the designer, then all other ideas associated with this question must be rejected. An example the use of this rule is presented in the next section. This rule is used for supporting decisions about the acceptance or not of the solution ideas proposed for a particular question during the design. These rules use the values (AND, OR, or XOR) of the *type* property of the *Question* element that defines if the solution ideas proposed for a particular design question are mutually exclusive or not.

The Kuaba ontology also allows specifying rules to validate the represented DR, to verify the well-formedness and completeness of the rationale structure with respect to the decisions made. For example, it is possible formulate questions such as the following: Are there questions without solution (without associated decisions)? Are there ideas that have no arguments? Are there nonconcluded ideas (without associated decisions)? Are there justifications that are not derived from at least one argument? Are there artifacts related to rejected ideas?

3. AN EXAMPLE OF DR REPRESENTATION WITH KUABA

The following example describes the use of the Kuaba approach to represent the DR used by two designers during the design of the conceptual schema shown in Figure 2. This schema models a music CD catalog. The example shows how the formal semantics of the UML metamodel, used for describing class diagrams, is integrated with the representation model defined by the Kuaba ontology.

Normally, the first activity done by a designer in designing a software artifact is the choice of design method or process

https://doi.org/10.1017/S0890060408000279 Published online by Cambridge University Press

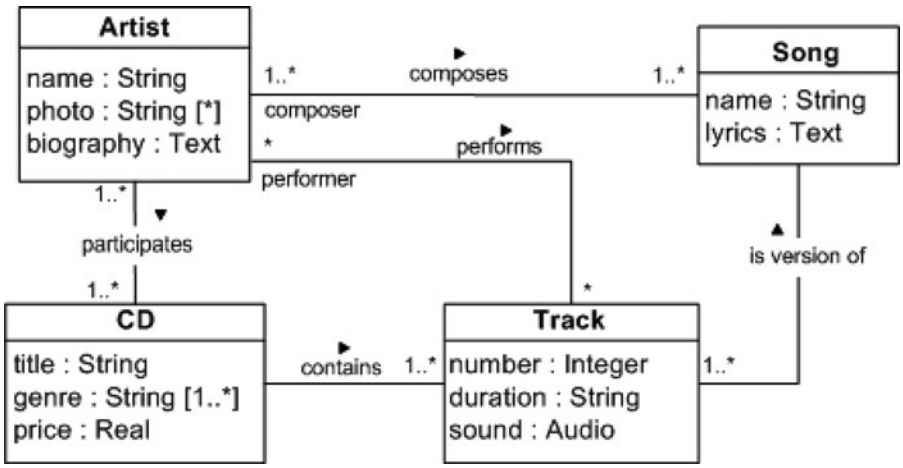

Fig. 2. A conceptual schema modeling a CD catalog.

that will be used to achieve the design. When the designer selects a design method or process, she or he implicitly determines the formal model(s) that will be used to describe the artifact. In this example, let us assume that the designers selected the Unified Process (Jacobson et al., 1999) to do the conceptual schema design. Thus, they implicitly adopted the formal model defined by the UML metamodel to describe this artifact. This information is recorded as part of the DR, as shown in the F-Logic code snippet below. It shows part of an instance of the Kuaba ontology representing general information about the design, such as method used, activity performed, expected duration and people involved.

```
/* Facts */
//Design Activity Information ----------
up:method[hasName->'Unified Software Development Process';
          prescribes->>{uml}].
uml:formal_model[hasName->'Unified Modeling Language (UML)';
                 hasLocalization->'http://www.omg.org/technology/documents/formal/uml.htm';
                 isPrescribedBy->>{up}].
domainModelActivity:activity[hasName->'Designing the domain Model';
                             hasStartDate->'2004-08-25T09:30:00';
                             hasFinishDate->'2004-08-28T10:10:00';
                             hasExpectedDuration->duration1;
                             requires->>{role1};
                             isExecutedBy->>{ana,carlos}].
duration1:expected_duration[hasAmount->3;
                             hasUnitTime->'day'].
ana:person[hasName->'Ana Soares'].
carlos:person[hasName->'Carlos Silva'].
role1:role[hasName->'designer'].
```

Once the unified process has been chosen as design method and the UML metamodel defined as the artifact formal model, the next step in the design is to produce the conceptual schema that, in this example, is represented as an UML class diagram. First, the designers need to identify the domain information items that are relevant for the design, which represent the possible elements of the conceptual model being created. Notice that these information items are determined by the designers' knowledge of the domain. They also could be obtained from domain ontology, or extracted from the DR of a previous phase in the software process, requirements elicitation, which is not addressed in this article. In this example, we consider that the designers initially propose the items *CD*, *Genre*, *Song*, and *Artist*. Next, the designers must decide how each one of them will be modeled using the UML to make up the final artifact, the class

https://doi.org/10.1017/S0890060408000279 Published online by Cambridge University Press

diagram. This decision process is driven by the formal semantics of the UML metamodel for class diagrams that determines, to a great extent, the questions and solution ideas that the designers can propose, because they are predefined by this metamodel. Figure 3 shows part of the UML metamodel (see OMG, 2003, for the complete metamodel).

According to this part of the UML metamodel an element in a class diagram can be a "Class," an "Attribute," an "Association," a "Generalization," or an "Association Class." By the inheritance relation, an element of the type *Class* can be seen as an aggregate of elements of the type *Attribute*. Similarly, an element of the type *Association* can be seen as an aggregate of elements of the type *Association End*, where each *Association End* must specify the class element that participates in the association.

The definitions described in the UML metamodel represent design options that can be used to model the information items of the knowledge domain in which the designers are working. The following rationale examples show how these design options are used in the Kuaba ontology instantiation to represent the DR for the conceptual schema being designed.

3.1. Rationale for CD and genre

The DR representation usually starts with a general question that establishes the problem to be solved. According to the UML metamodel (Fig. 3), 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" element with the instance, What are the model elements? Figure 4 depicts a graphical representation we have created to help visualize instances of the Kuaba ontology. It shows the portion of the DR regarding the solution ideas to model the "Genre" information item. 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.

Once these first ideas for the CD Catalog model elements have been established, the designers must decide how each one of them will be modeled using the UML. This next step is represented in Figure 4 by the "suggests" relation, which determines questions entailed by proposed ideas: How to model CD? How to model Genre? How to model Name?

The possible ideas that address these questions are determined by the UML metamodel 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 metamodel. Strictly speaking, designers consider all the other alternatives proposed by the UML, but for the sake of simplicity we have shown only these two. The properties and types specified in the UML metamodel elements also represent design options that can be used in the Kuaba ontology instantiation to represent DR. The questions Type?, Minimum Multiplicity?, and Maximum Multiplicity? and their corresponding solution ideas are an example.

This example illustrates how the formal semantics of the UML metamodel "drives" the instantiation of the Kuaba ontology providing the design options used to represent the DR of the artifact being designed. Therefore, when the selected design method has a well-defined metamodel it is possible

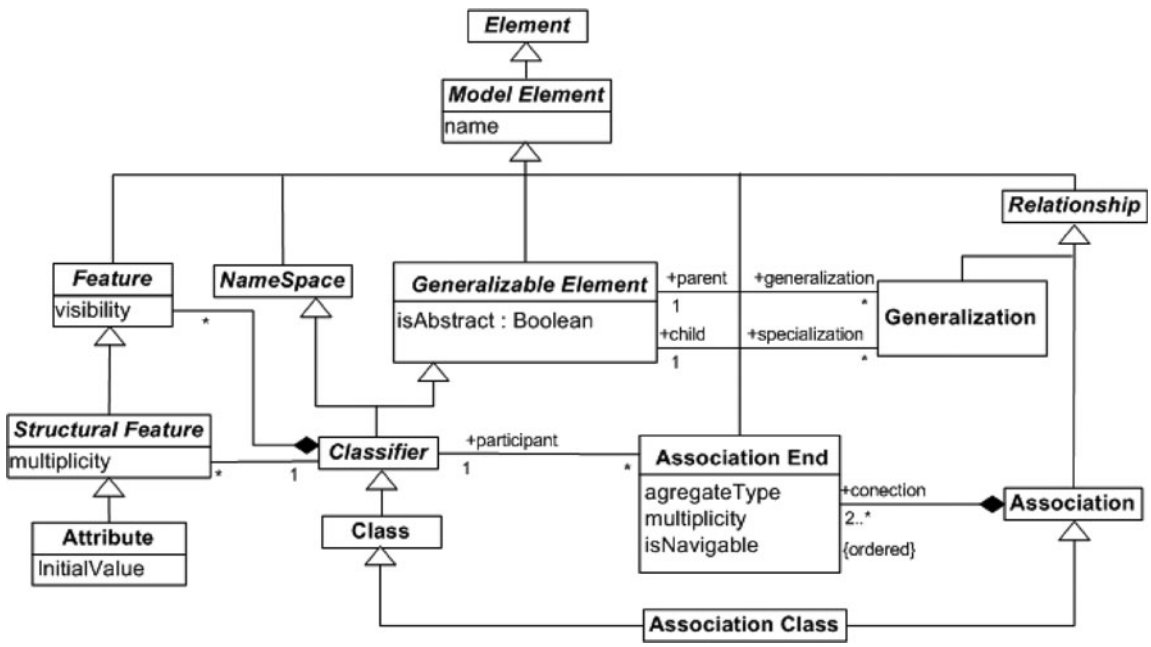

Fig. 3. A partial UML metamodel for class diagrams.

https://doi.org/10.1017/S0890060408000279 Published online by Cambridge University Press

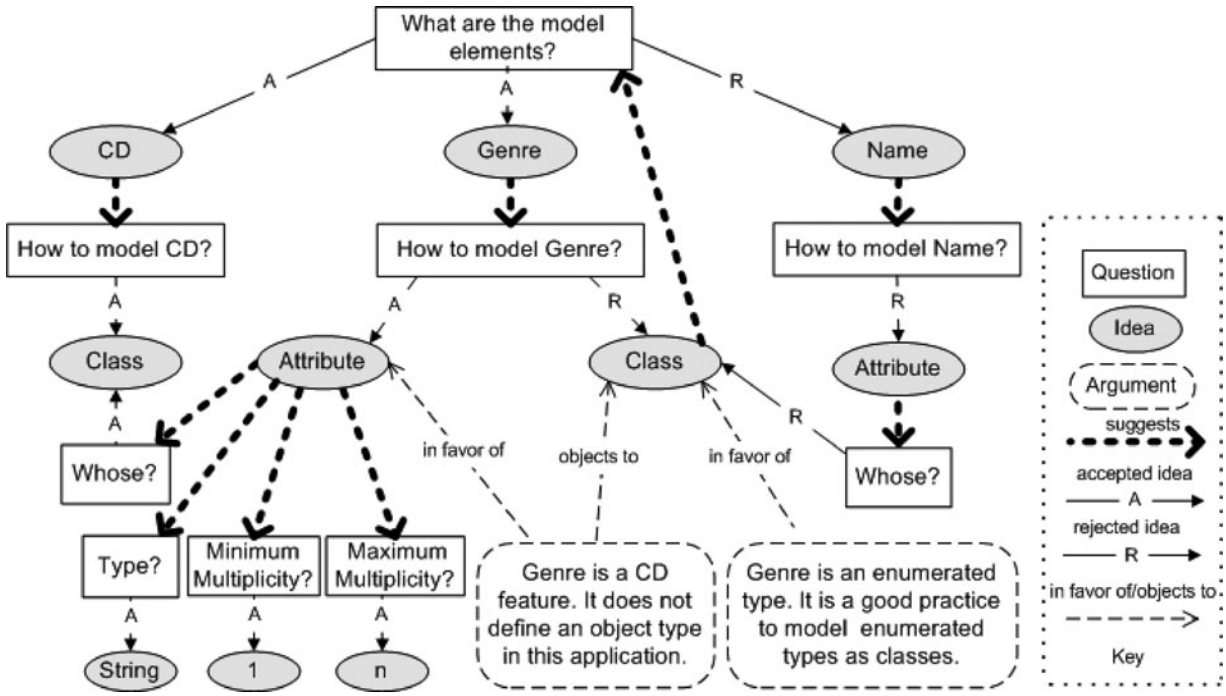

Fig. 4. An example of DR for the Genre element.

automate the instantiation of a large part of the reasoning elements (questions and solution ideas) that must be recorded during the design of an artifact. In this case, the designers only need to provide the information items of the domain and the arguments for and against each design option. Notice that, in the example of the Figure 4, only the solution ideas addressing the initial question and the arguments (represented as dashed rectangles) have informal content provided by the designers.

The portion of the Kuaba ontology instance shown below represents this part of the rationale used for the *Genre* element design expressed using F-Logic.

In this Kuaba instance example, observe that the reasoning elements based on the UML metamodel have the "isDefinedBy" relation that distinguishes them from the reasoning elements provided by the designers.

The UML metamodel defines a class can as an aggregate of attributes. Thus, to model *Genre* as a class is necessary to define its attributes. This is represented by the *suggest* relation between the *Class* idea and the initial question, "What are the model elements?," shown in Figure 4. This relation represents that the designers need to identify other information items related to *Genre*, before making a decision about the

```
/* Facts */
// Reasoning Elements ----------
genre:idea[hasText->'Genre';
        hasCreationDate->'2004-08-25T09:12:06'; address->>{whatElements};
        hasArgument->>{genreArgument}; isInvolved->domainModelActivity;
        isPresentedBy->carlos; suggests->>{hwModelGenre}].
hwModelGenre:question[hasText->'How to model Genre?'; isDefinedBy->uml;
                hasCreationDate->'2004-08-25T09:15:02'; hasType->'XOR';
                isInvolved->domainModelActivity; isPresentedBy->carlos;
                hasDecision->>{genreAttribDecision;genreClassDecision}].
genreAttribute:idea[hasText->'Attribute';
                hasCreationDate->'2004-08-25T09:21:02'; address->>{hwModelGenre};
                hasArgument->>{genreAttribArgument}; isDefinedBy->uml;
                isPresentedBy->ana; isInvolved->domainModelActivity;
                suggests->>{whoseAttrGenre;minMultAttribGenre;maxMultAttribGenre}].
```

https://doi.org/10.1017/S0890060408000279 Published online by Cambridge University Press

```
genreClass:idea[hasText->'Class';
                hasCreationDate->'2004-08-25T09:21:12';
                address->>{hwModelGenre;whoseAttribGenreName}; isDefinedBy->uml;
                hasArgument->>{genreClassArgument1;genreNameAttribArgument2};
                isInvolved->domainModelActivity; isPresentedBy->carlos;
                suggests->>{whatElements}].
genreAttribArgument:argument[hasCreationDate->'2004-08-25T09:21:56';
                             inFavorOf->>{genreAttribute}; objectsTo->>{genreClass};
                             isInvolved->domainModelActivity; isPresentedBy->ana;
                             hasText->'Genre is a feature of CD. It does not define an
                                       object type in this application'].
genreClassArgument:argument[hasCreationDate->'2004-08-25T09:22:00';
                             inFavorOf->>{genreClass};
                             isInvolved->domainModelActivity; isPresentedBy->carlos;
                             hasText->'Genre is an enumerated type. It is a good practice
                                       to model enumerated types as classes'].
```

its design. Otherwise, the design could be inconsistent. Thus, in the rationale shown in Figure 4, the designers also proposed the "Name" information item and considered modeling it as an attribute. Because the *Attribute* idea, in turn, must be associated with a *Class* according to the UML metamodel, the question "Whose?" is suggested, which in turn, will be addressed by the idea corresponding to the class whose attribute it is.

Figure 4 also shows the decisions made, labeling each solution idea to each question with an "A" (for accepted) or "R" (for rejected). Thus, the example represents the fact that the designers decided to accept the *Attribute* idea as a solution to the question "How to model Genre?," in detriment of the *Class* idea. The subgraph of the DR made up of *Question* and *Ideas* is actually an AND/OR graph (Nilsson, 1986) that can be seen as a goal decomposition of the root node, which is always a *Question*. According to Section 2.1, the *Question* element in the Kuaba ontology has a "type" property, whose possible values (AND, OR, and XOR) are used in rules to suggest decisions about the acceptance or not of the proposed solution ideas.

In the representation shown in Figure 4, if the decision to accept the idea of modeling *Genre* as an attribute was the first one made by the designers, a support system can 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 to the question "How to model Genre?" and this question is of type XOR. 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 the rules defined by the Kuaba ontology, as well as those expressed by the metamodel used, to validate the decisions made by the designers, flagging eventual inconsistencies. Therefore, the order of acceptance or rejection of an idea does not affect the represented rationale.

The decisions made about the design options to model *Genre*, expressed in F-logic, are presented below. Notice that the representation of each decision contains a final justification specification. The representation of this justification helps to better understand how the arguments presented during the design influenced or did not influence the final decisions made by the designers.

```
/* Facts */
// Decisions and Justifications ----------
genreAttribDecision:decision[concludes->genreAttribute; isAccepted->'true';
                             hasDate->'2004-08-25T09:28:12'; isMadeBy->>{carlos};
                             hasJustification->genreAttribJustification].
genreAttribJustification:justification[isDerivedOf->>{genreAttribArgument};
                                       hasText->'In this application we only need to record what is
                                                 the genre of a CD. In the conceptual model we
                                                 can represent it defining an attribute genre for the
                                                 CD class.'].
```

https://doi.org/10.1017/S0890060408000279 Published online by Cambridge University Press

```
genreClassDecision:decision[concludes->genreClass; isAccepted->'false';
                            hasDate->'2004-08-25T09:28:12'; isMadeBy->>{carlos};
                            hasJustification->genreClassJustification].
genreClassJustification:justification[isDerivedOf->>{genreClassArgument};
                            hasText->'Important properties were not found to justify the
                                representation of genre as a class. Thus, I decided
                                to model genre as an attribute with multiplicity 1..n
                                to simplify the conceptual schema.' ].
```

Concluding the rationale example for the CD element design, we assume that the designers considered other information items related to a CD, besides *Genre*, for instance, the "Title" and "Price" items. The DR representation for these items is not shown, because it is very similar to the rationale shown in Figure 4 for the *Genre* and *Name* items.

The resulting artifact of the DR represented in Figure 4 is the CD class illustrated in Figure 2. In the Kuaba ontology vocabulary, an artifact can be represented as *atomic artifact* or *composite artifact*. The decision about how to represent an artifact depends on artifact type and its definition in the metamodel used. According to the UML metamodel, a class is an aggregate of attributes. Therefore, the CD class shown in Figure 2 is represented in Kuaba as a composite artifact of several atomic artifacts, which represent its attributes (title, genre, and price).

Below we show a portion of the Kuaba ontology instance representing the CD artifact. This representation contains the "resultsIn" relation, which records the solution idea that originates the CD artifact. Notice that the *Title*, *Price*, and *Genre* items are represented as atomic artifacts, because the designers decided to model them as attributes of CD.

```
/* Facts */
// Artifacts ----------
titleArtifact:atomic_artifact[hasName->'Title attribute';
                            hasCreationDate->'2004-08-26T09:30:52';
                            hasDescription->'Attribute of the CD class that contains the name
                                of the album';
                            isCreatedBy->>{carlos}; isDescribedBy->uml;
                            resultsIn->>{title}].
genreArtifact:atomic_artifact[hasName->'Genre attribute';
                            hasCreationDate->'2004-08-26T09:31:12';
                            hasDescription->'Attribute of the CD class that contains the musical
                                genre of the album';
                            isCreatedBy->>{carlos}; isDescribedBy->uml;
                            resultsIn->>{genre}].
priceArtifact:atomic_artifact[hasName->'Price attribute';
                            hasCreationDate->'2004-08-26T09:32:02';
                            hasDescription->'Attribute of the CD class that contains the price
                                of the album';
                            isCreatedBy->>{carlos}; isDescribedBy->uml;
                            resultsIn->>{price}].
cdArtifact:composite_artifact[hasName->'CD class';
                            hasCreationDate->'2004-08-26T09:48:02';
                            isCreatedBy->>{carlos}; isDescribedBy->uml;
                            compositionOf->>{titleArtifact; genreArtifact; priceArtifact};
                            resultsIn->>{cd};
                            hasDescription->'Main class that contains all attributes that an
                                object CD can have in this application'].
```

https://doi.org/10.1017/S0890060408000279 Published online by Cambridge University Press

3.2. Rationale about Song

Figure 5 shows the DR used by the designers for modeling the Song information item. This DR example shows the use of other design options defined by the UML metamodel in the Kuaba ontology instantiation, besides the options *Class* and *Attribute*.

This DR representation shows the "Track" idea as a possible information item related to the *Song* element and the ideas *Attribute*, *Class*, and *Association Class* as possible design options for modeling this item in the class diagram. According to the UML metamodel, shown in Figure 3, an association class is a specialization of the elements *Class* and *Association*. As a specialization of the *Association* element, an association class must have at least two association ends. Thus, the type property of the question "Association Ends?" shown in Figure 5 must be recorded in the rationale representation with value "OR," because more than one solution idea must be accepted for this question. In this example, the decisions of accepting or rejecting the ideas "Destination" and "Origin" must be the same, that is, they must be both accepted or both rejected.

In the UML metamodel, each end proposed for an association must be related to one of the classes participating in the association. This is represented by the questions "Participant?" shown in Figure 5. The ideas that address these questions indicate that the designers considered modeling the *Track* element as an association class between the *CD* and *Song* elements, both modeled as classes. However, this solution idea was rejected according to the argument presented against the idea *Association Class*. Figure 6 shows the rationale behind other decisions made about the design of the *Track* element.

This example shows that the information items proposed by the designers during the conceptual schema design also can be modeled as associations. In this DR, the designers propose the items "Contains" and "Version of" as elements of the class diagram and decide to model them as associations between the elements *CD*, *Track*, and *Song*. The ideas "1" and "n" addressing the "Minimum Multiplicity?" and "Maximum Multiplicity?" questions indicate that a CD can have one or more tracks and a song can be recorded in one or more tracks. Figure 7 shows the resulting class diagram after the decisions made about the modeling of the elements *Song* and *Track*. The DR representation for the attributes of the classes Track and Song is not presented, because it is very similar to the DR representation shown in Figure 4 for the attributes of the CD class.

3.3. Rationale about Artist

Concluding the example of DR representation using the Kuaba approach, this section presents part of the designers' reasoning during the design of the "Artist" information item. Suppose that this reasoning begins with the argument against the idea of modeling the "Singer" element as a class, shown in Figure 8. This could be validated by searching the creation data and hour of each reasoning element included

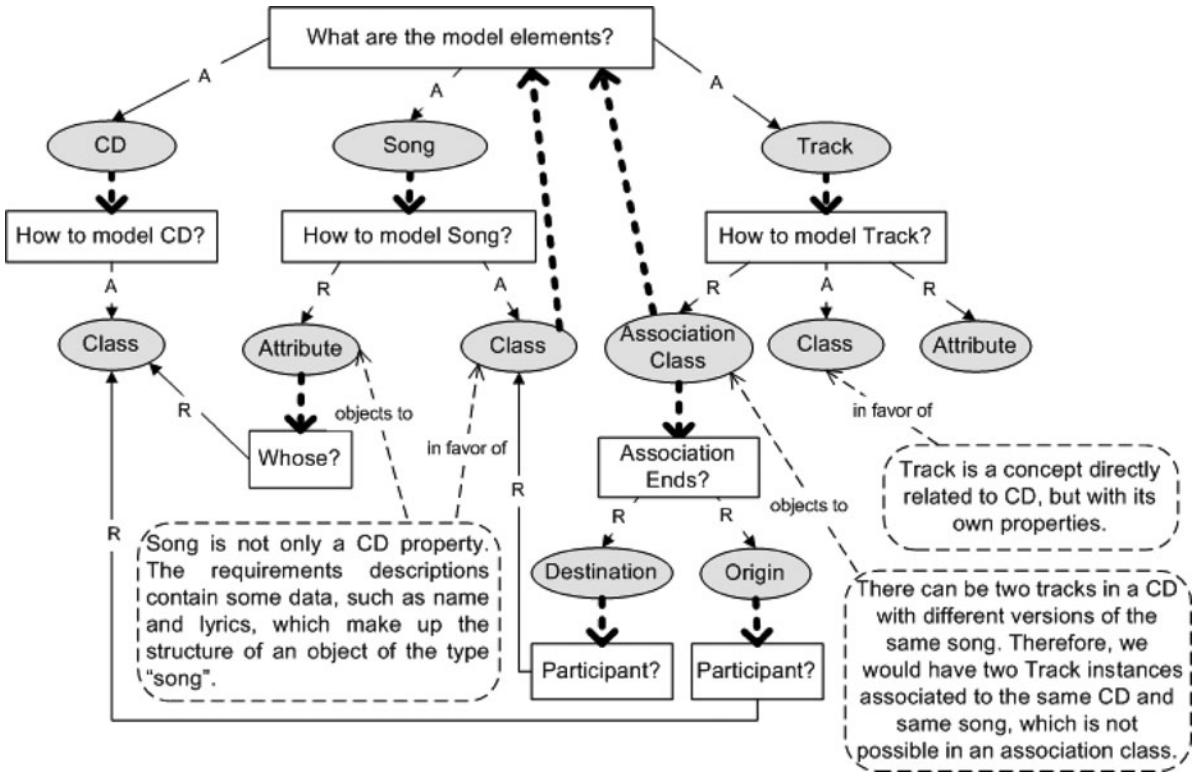

Fig. 5. A partial example of the DR for the Song element.

https://doi.org/10.1017/S0890060408000279 Published online by Cambridge University Press

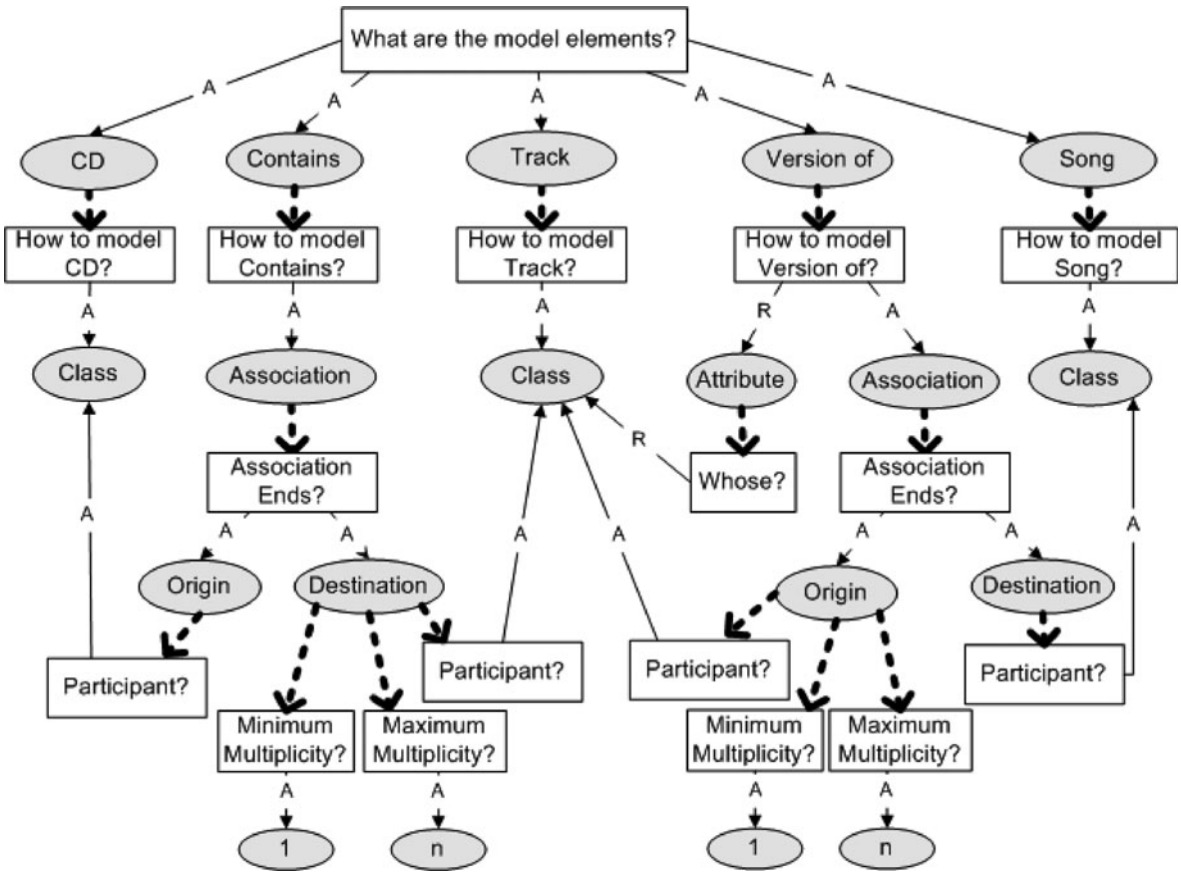

Fig. 6. A partial example of the DR for the Track element.

in the DR representation. The "suggests" relation between this argument and the initial question "What are the model elements?" indicates that the designers needed to propose new information items to model the different participation types of an artist in a CD, for instance, the items *Artist*, *Performer*, and *Composer*.

In this example, we can observe that the designers considered the "Generalization" idea to model the "Participation" information item that suggests the questions "Parent?" and "Child?" According to the UML metamodel (Fig. 3), a generalization is a relation between an element "parent" and at least one element "child" that inherits the properties defined in the "parent" element. Generally, these elements are known in object oriented modeling as "superclass" and "subclass," respectively. However, we use the terms "parent" and "child" adopted in the UML metamodel as instance values for the *Question* element of the Kuaba ontology, because the generalization concept also can be used to represent specializations of associations between classes. Thus, the DR represented in Figure 8 shows that the designers considered the solution of modeling the *Participation* element as a generalization relation between the element *Artist* (parent) and the elements *Performer* and *Composer* (children).

When the designers propose the ideas *Artist*, *Performer*, and *Composer* as elements of the conceptual schema new design problems are suggested. These problems are represented by the questions about how to model these elements in the conceptual schema and how they are related to other model elements.

In Figure 8, the relation "suggests" between the *Class* idea proposed as a solution for modeling *Performer* and the initial question indicates that it was necessary for the designers to define new information items, for instance, "Photo." This item could be modeled as an attribute of the *Performer* element, which could justify the modeling of this element as a "child" (subclass) of *Artist*. However, the rationale shows that the designers decided to model *Photo* as an attribute of

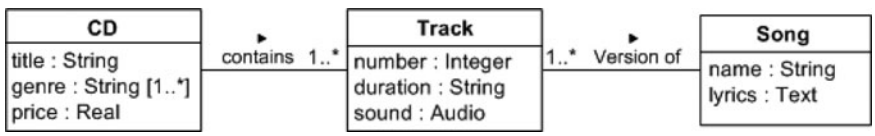

Fig. 7. A partial class diagram with the Song and Track elements.

https://doi.org/10.1017/S0890060408000279 Published online by Cambridge University Press

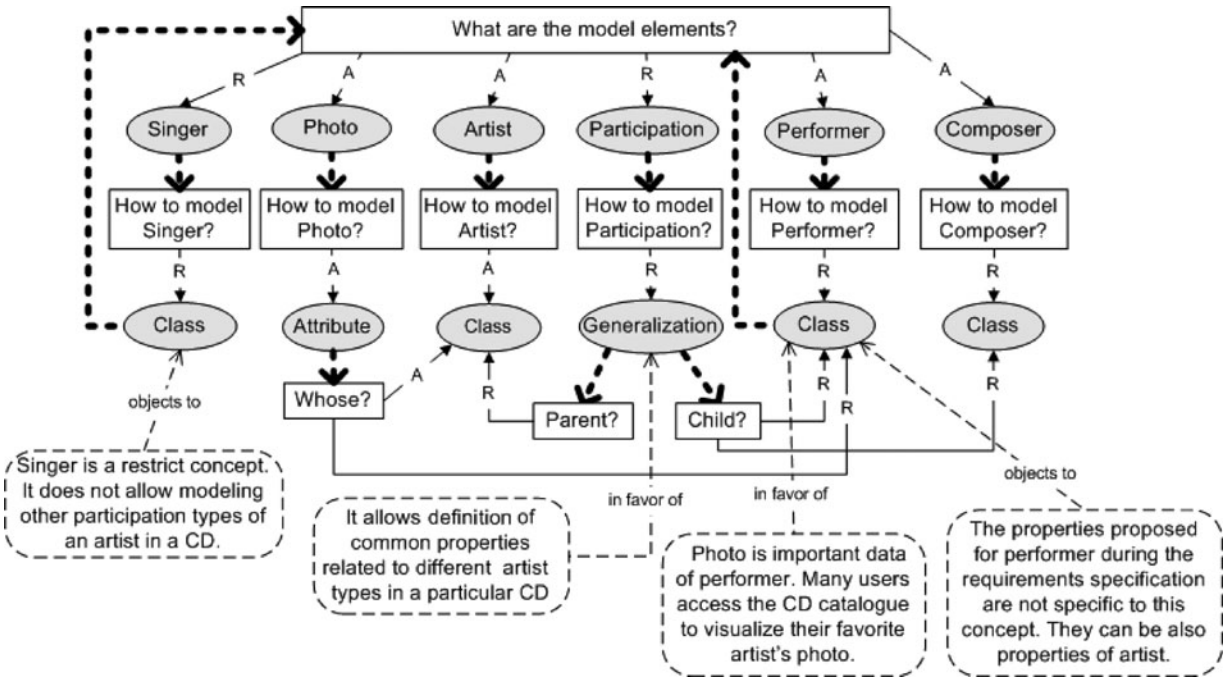

Fig. 8. An example of part of the rationale about the Artist and Performer elements.

Artist element, designed as a class. So, the idea of modeling the *Performer* element as a specialization class of *Artist* was rejected by the designers.

Notice that it is not possible understand the arguments presented for the *Class* idea in the subtree of the *Performer* element without also considering the questions "How to model Performer?" and "Whose?" addressed by this idea. Although the existing relation between the arguments and these questions is not graphically represented in Figure 8, it is recorded, as shown below.

```
/* Facts */
// Reasoning Elements ----------
performerClass:idea[hasText->'Class';
                    hasCreationDate->'2004-08-26T19:21:13';
                    address->>{hwModelPerformer; whoseAttribPhoto};
                    isDefinedBy->uml; suggests->>{whatElements};
                    hasArgument->>{performerClassArgument1; performerClassArgument2};
                    isInvolved->domainModelActivity; isPresentedBy->carlos].
performerClassArgument1:argument[hasCreationDate->'2004-08-26T19:22:56';
                                 inFavorOf->>{performerClass};
                                 isInvolved->domainModelActivity; isPresentedBy->ana;
                                 hasText->'Photo is an important data of performer.  Many
                                          users access the CD catalogue to see the photo
                                          of their favorite artist';
                                  considers->whoseAttribPhoto].
performerClassArgument2:argument[hasCreationDate->'2004-08-26T19:24:00';
                                 objectsTo->>{performerClass};
                                 isInvolved->domainModelActivity; isPresentedBy->carlos;
                                 hasText->'The properties proposed for performer during the
                                          requirements specification are not specific of this
                                          concept. They can be also properties of artist.';
                                  considers->hwModelPerformer].
```

https://doi.org/10.1017/S0890060408000279 Published online by Cambridge University Press

This example shows the relation "considers" represented for the two arguments associated to the *Class* idea in the subtree of the *Performer* element. This relation allows representing that each argument is valid only for one of the questions addressed by this idea. In this way, the first argument (*performerClassArgument1*) is valid only when the *Class* idea addresses the "Whose?" question, and the second one is valid only when this idea addresses the question "How to model Performer?"

4. DESIGN REUSE USING RATIONALE

Formalizing DR representation using the model defined by Kuaba enriched with the formal semantics of the metamodel that describes the artifacts makes the DR more specific according to the design methods and allows design reuse, which is a new kind of reuse. It is achieved through the integration of existing DRs when beginning the design of a new artifact. This type of reuse requires different kinds of computable operations on the recorded DR. These operations involve matching instances of the Kuaba ontology (DR representations) to compose a more complete solution of design.

The operations on the recorded DR are performed by the rationale processor, one of the components of the conceptual architecture defined for the integrated design environment proposed in this work. This environment aims at making the capture, representation and use of DR part of the design process, and allowing the computational processing of DRs to support reuse. Figure 9 shows the conceptual architecture of this integrated environment.

Because most software design support tools already use some kind of formal description (metamodel) of the artifacts being designed, we propose to extend them to allow their integration with the DR processor developed in this work. As we can observe in Figure 9, this extension enriches the design tools by adding two layers to support the editing and searching of DR. In the editing layer, the designer informs the arguments for and against the design alternatives considered, and the justifications for the decisions made. In the search layer, s/he searches existing designs with their rationales, formulates questions about the designs found, and starts the integration of rationales. In this layer the designer can also graphically visualize the rationale of the artifact being designed, or the rationale of the designs being reused in her or his design.

The extension of a software design tool to support DR using the Kuaba approach allows the designers capturing and representing DR while designing the software artifacts. In this extended tool, a large part of the rationale (questions and design ideas) is automatically obtained from the design metamodel used. Therefore, the effort required from the designers to record their rationale can be reduced, because they need to inform manually only their arguments and justifications instead of informing all rationale structure.

In the proposed architecture, the design tool transfers the design options and the rationale information provided by the designer to the rationale processor. This processor is responsible for creating the DR representations and processing the rationales integration, when requested by the designer. In a future version, the design tool will also be capable of generating the artifact based on the modifications and decisions made by the designer over the integrated DR.

4.1. The rationale processor

The representation and the integration of DRs involves different types of operations such as queries, operations to create instances of the Kuaba ontology, and operations to match or integrate elements of two or more instances of this ontology.

Queries and operations to create instances of ontology have already been implemented in several ontology editors such as Protégé (Noy et al., 2001). The operations to perform the integration of instances of the same ontology have not been considered in majority of the systems proposed to support DR. Basically, these operations involve the search, copy,

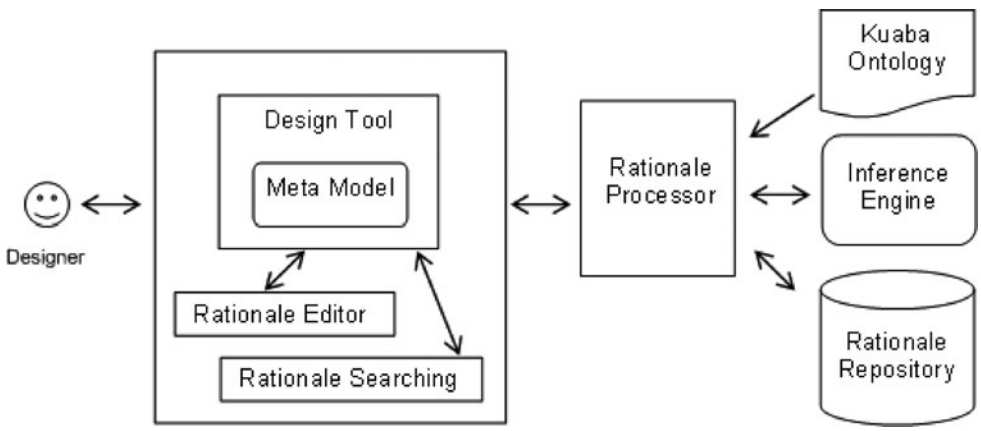

Fig. 9. The conceptual architecture of an environment supporting DR.

https://doi.org/10.1017/S0890060408000279 Published online by Cambridge University Press

substitution and union of elements of the instances being integrated.

Search operations allow the designer to select which elements of the representations considered for the integration will be included in the new design. For example, the designer could provide a question, and request that the search tool recover only the ideas that address this question and have arguments for them.

Substitution operations allow the designer substitute an element in one representation by a corresponding element in another representation. This operation can be used, for example, when the designer wants to use the DR of one representation, but needs to substitute one of its elements by an element specified in another representation.

Copy operations allow the designer copy elements of one representation into another.

Finally, union operations allow joining the reasoning elements described in the representations involved in the integration to generate a new design. 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, the designer could define the *Question* element that would be the root of the union of the representations. Or still, the designer could 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.

Currently, the rationale processor implements the complete union of two DR representations. This operation consists of a set of rules implemented in the Flora-2 language (see http://flora.sourceforge.net) that translates F-Logic into tabled Prolog code and processes this code in the XSB deductive system (see http://xsb.sourceforge.net/). These rules consist basically in the recursive processing of the various subtrees of questions and solution ideas that compose the DR representations.

4.2. An example of DR integration

To exemplify the design reuse by DR integration, consider the scenario where a designer wants to design a class diagram to represent information that will be used in a CD Store application. Because 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 he or she begins a new design. As a result, s/he finds some 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, recorded in the available DR representations. Thus, the designer selects two artifacts representing different solution alternatives to model information about how an artist participates in a CD. According to the DR representations of these artifacts, this participation can be modeled in different ways, as Figure 10 shows.

The DR representation of the first artifact, depicted in Figure 11, shows that the author considered two design solutions for modeling the artist's performance in a CD: modeling it as a simple association between the classes "CD" and "Musician" (Fig. 10a) or as an association class between these two classes (Fig. 10b). Examining the decisions recorded in this DR (labels A and R), the designer verifies that the author decided to model *Performance* as an association class with an attribute "type" related to the classes *CD* and *Musician*.

Analyzing the DR of the second artifact, shown in Figure 12, the designer verifies that the author instead of considering the *Musician* element in the class diagram considered the *Artist* element and decided to model this element as a class. In addition, the author also decided to model *Participation* as a generalization and the item *Participates* as an association between the elements *Artist* and *CD* (Fig. 10c).

After analyzing the solution ideas and the decisions made by the authors of these artifacts, the designer decided to integrate the DRs shown in Figures 11 and 12 to start her or his design with a larger set of options. This integration is possible because the designs being reused are described by the formal semantics of the same metamodel (UML) and represent the same application domain (CD catalog).

The integration of two DR representations involves the definition of the representation that will be used as the basis for the integrated DR, the equality specification between equivalent elements in the two representations, and the union of

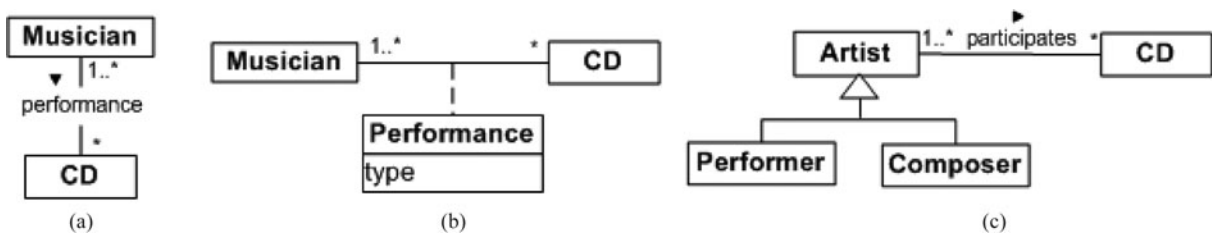

Fig. 10. Design options for modeling an artist's participation in a CD.

https://doi.org/10.1017/S0890060408000279 Published online by Cambridge University Press

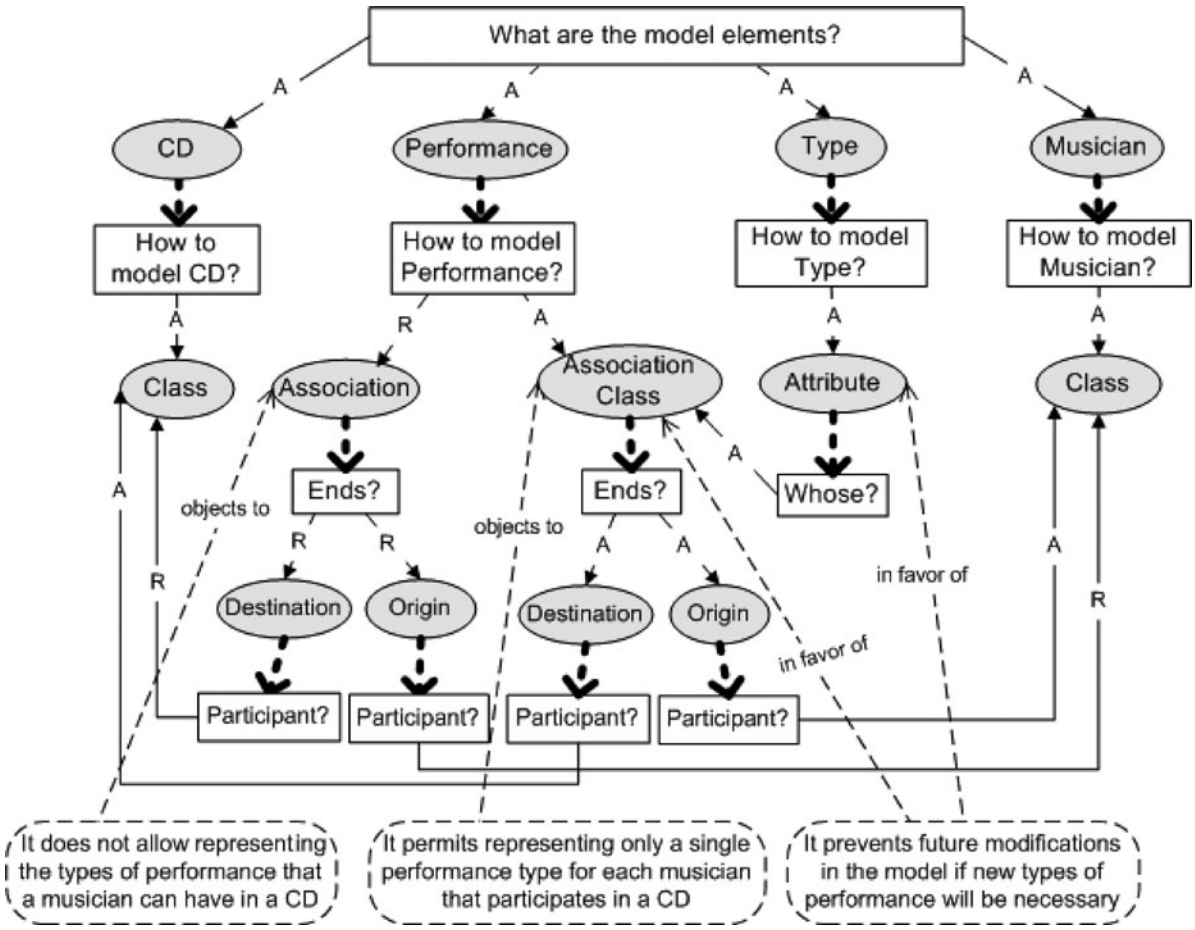

Fig. 11. An example of DR about the Musician's performance modeling (Fig. 10a and b).

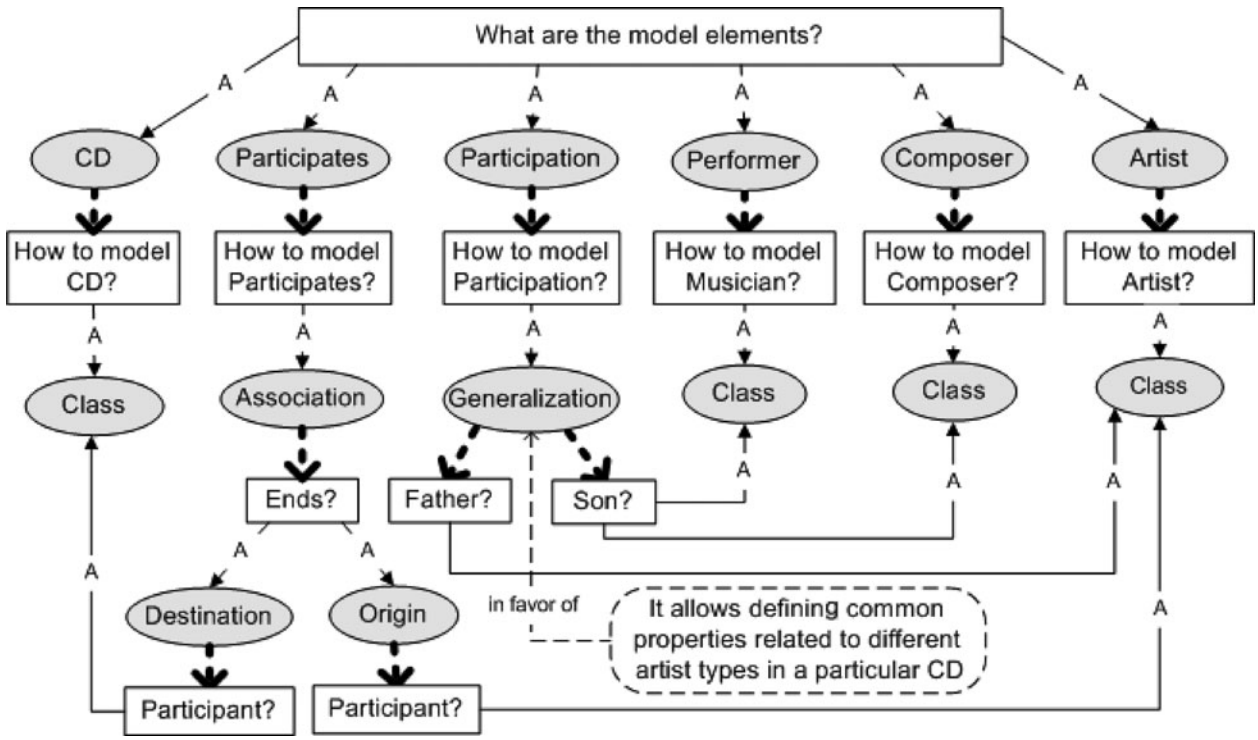

Fig. 12. An example of DR about the elements Artist and Participation (Fig. 10c).

https://doi.org/10.1017/S0890060408000279 Published online by Cambridge University Press

the reasoning elements (equivalent or not) of these DR representations.

The definition of the base representation and the equality specification between domain ideas are performed by the designer. In this example, we assume that the designer defined the DR representation shown in Figure 12 as the base, and specified the identity of the elements *Musician* and *Artist*, as well as the equality between the elements *Performance* and *Participation* of the DRs that will be integrated (Figs. 11 and 12). These equalities are specified in Flora2 using the predicate ": = :" as

```
Musician:=:Artist.
Performance:=:Participation
```

Based on this equality specification, the rationale processor first identifies the equivalent ideas, applying the equivalence rules shown below. According to these rules, questions are considered equal if they have the same text (or they are in the equality specification defined by the designer) and are suggested by equivalent ideas. Domain ideas are considered equal if they have the same text (or they are in the equality specification) and address equivalent questions. Finally, design ideas are considered equal if they have the same text and address at least one common equivalent question. When the rationale processor finds an equivalent idea among the DR representations being integrated, it simply copies the arguments of this idea from the original representation, adding them to the base representation.

After identifying and treating the equivalent ideas, the rationale processor identifies the questions that are different (nonequivalent) in the two representations and copies (adds) them to the base representation. The operations used by the processor to treat these questions are shown below.

```
?- L=collectset{Q|Q[isSuggestedBy->>I]@mod1},
   L2=collectset{Q|Q[isSuggestedBy->>I]@mod2},
   get_non_equivalent_question(L,L2,List), copy_question(List).
```

In the code above, the rationale processor creates two lists (L, L2) with the questions of the two DR representations being integrated. Note that the questions retrieved must be suggested by some idea. This restriction allows distinguishing them from the initial questions of each representation, which are always equal because they are defined by the metamodel used. The *get_non_equivalent_question* operation processes these lists returning a new list with the different questions. The *copy_question* operation, shown below, copies the questions from this list to the base representation. For example, the questions "*How to model type?*" and "Whose?" in the subtree of the idea "Type" (Fig. 11) are copied to the base representation. If one of these questions is addressed by an idea equivalent to an idea already existent in the base representation, the processor updates this representation making the existing idea address the question

```
equivalent_question(Question1, Question2) :- Question1[hasText->_TQ1, isSuggestedBy->>_I1]@mod1,
                                             Question2[hasText->_TQ2, isSuggestedBy->>_I2]@mod2,
                                             (_TQ1=_TQ2;Question1:=:Question2),
                                             ( equivalent_domainIdea(_I1,_I2);
                                               equivalent_designIdea(_I1,_I2);
                                               _I1:=:_I2 ).

equivalent_domainIdea(Idea1, Idea2) :- Idea1[not (isDefinedBy->_X),hasText->_TI1,
                                             address->>_Q1[hasText->_TQ1]]@mod1,
                                       Idea2[not (isDefinedBy->_Y),hasText->_TI2,
                                             address->>_Q2[hasText->_TQ2]]@mod2,
                                       (_TI1=_TI2;Idea1:=:Idea2),
                                       (_TQ1=_TQ2;_Q1:=:_Q2), not(_Q1=_Q2).

equivalent_designIdea(Idea1, Idea2) :- Idea1[hasText->_TI1]@mod1,
                                       Idea2[hasText->_TI2]@mod2,
                                       _TI1=_TI2,
                                       LQ1=collectset{Q1|Q1[isAddressedBy->>Idea1]@mod1},
                                       LQ2=collectset{Q2|Q2[isAddressedBy->>Idea2]@mod2},
                                       get_equivalent_question(LQ1,LQ2,List),
                                       not(List = []).
```

https://doi.org/10.1017/S0890060408000279 Published online by Cambridge University Press

just copied. This is achieved by the "identify_equivalent_idea" operation.

The rationale processor first creates two lists (L, L2) with the ideas of the two DR representations being integrated.

```
copy_question([]).
copy_question([Question1|Tail]) :-
        insert{(Question1:question, Question1[hasText->X, hasCreationDate->Y,
              hasType->T, isInvolved->Z, isDefinedBy->W])@mod2 |
              Question1[hasText->X, hasCreationDate->Y, hasType->T,
              isInvolved->Z,isDefinedBy->W]@mod1},
        LI1=collectset{I|I[address->>Question1]@mod1},
        LI2=collectset{I|I:idea@mod2},
        identify_equivalent_idea(LI2,LI1,Question1),
        copy_question(Tail).
```

After treating the nonequivalent questions, the rationale processor identifies the ideas that are different (nonequivalent) in the two representations and copies them to the base representation. The operations used by the processor to treat these ideas are shown below.

```
?- L=collectset{I|I:idea@mod1},
   L2=collectset{I|I:idea@mod2},
   get_non_equivalent_idea(L,L2,List), copy_idea(List).
```

Then, the *get_non_equivalent_idea* operation processes these lists returning a new list with the different ideas. The *copy_idea* operation, shown below, copies the ideas in this list to the base representation. For example, the ideas *Type* and *Attribute* as well as the ideas *Association Class*, *Origin*, and *Destination* in the subtree of the element *Performance* (Fig. 11) are copied to the base representation (Fig. 12). For each idea copied, the processor copies also its arguments (*copy_argument* operation) and treats questions equivalent to the questions addressed by it in the original representation (*identify_equivalent_question* operation). Figure 13 shows part of the resulting integrated DR.

```
copy_idea([]).
copy_idea([Idea1|Tail]) :-
        if not(Idea1:idea@mod2) then
          (insert{(Idea1:idea,Idea1[hasText->X, hasCreationDate->Y, isInvolved->Z])@mod2 |
                Idea1[hasText->X, hasCreationDate->Y, isInvolved->Z]@mod1},
           insert{Idea1[address->>{Q}]@mod2 | Idea1[address->>Q]@mod1},
           insert{Idea1[hasArgument->>{A}]@mod2 | Idea1[hasArgument->>A]@mod1},
           LA=collectset{A1|A1[inFavorOf->>Idea1]@mod1;A1[objectsTo->>Idea1]@mod1},
           copy_argument(LA,Idea1),
           LQ1=collectset{Q1|Q1[isAddressedBy->>Idea1]@mod1},
           LQ2=collectset{Q2|Q2:question@mod2},
           identify_equivalent_question(LQ2, LQ1, Idea1),
           if Idea1[suggests->>_X]@mod1 then
             (insert{Idea1[suggests->>{Q1}]@mod2 | Idea1[suggests->>Q1]@mod1}),
              if Idea1[isDefinedBy->M]@mod1
              then
                 (insert{Idea1[isDefinedBy->M]@mod2 | Idea1[isDefinedBy->M]@mod1}))
              else
                 (insert{Idea1[address->>{Q}]@mod2 | Idea1[address->>Q]@mod1}),
          copy_idea(Tail).
```

https://doi.org/10.1017/S0890060408000279 Published online by Cambridge University Press

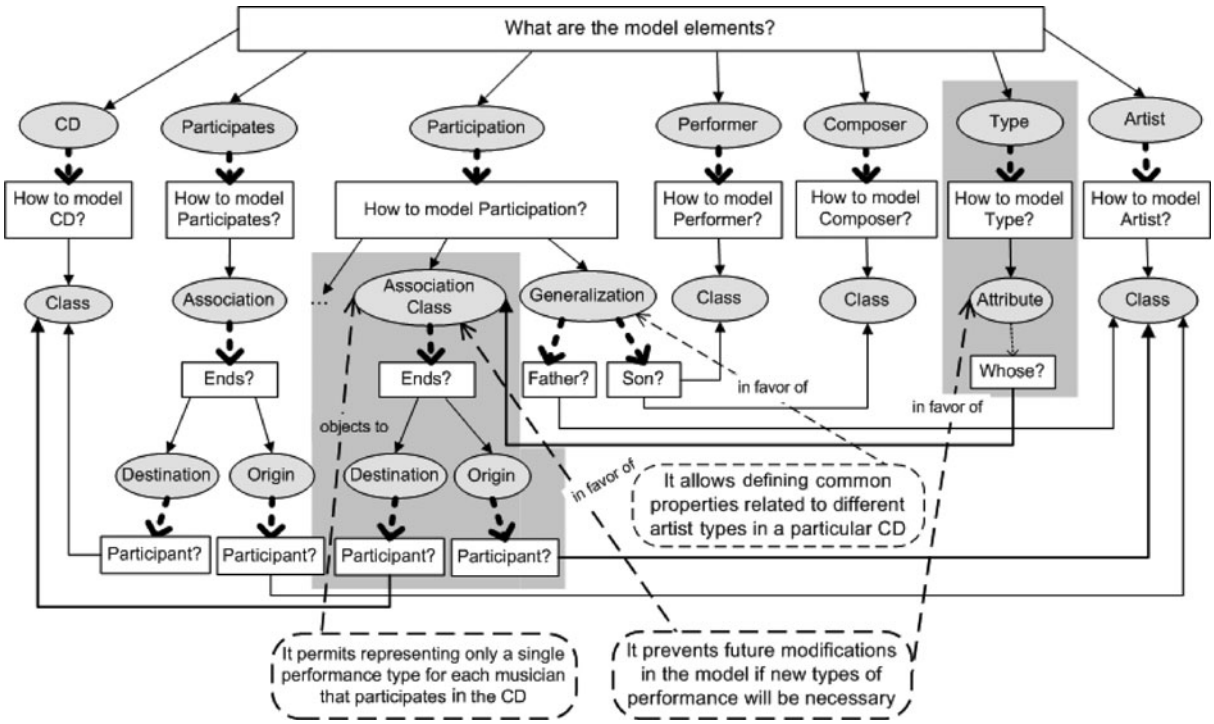

Fig. 13. A partial example of the integrated DR. Newly included elements are shown in gray background.

Finishing the integration, the processor verifies the base representation and makes the necessary modifications to guarantee the consistency of the relations between questions and ideas in the integrated representation. For instance, in Figure 13, the idea *Association Class* is associated to the question "How to model Participation?" that is equivalent to the question "How to model Performance?" in the original representation (Fig. 11).

In the integrated DR (Fig. 13), all design options in Figure 10 were joined in one unique DR representation. Notice that the decisions made for the reused artifacts, shown in Figures 11 and 12, are not incorporated to this representation. This reflects the fact of this integrated DR represents a new design, in which the designer can do modifications and make new decisions according to her or his objectives, creating a new artifact.

5. THE HYPERDE+DR ENVIRONMENT

HyperDE+DR is an extension of the HyperDE environment (Nunes & Schwabe, 2006), in which the layers of editing and searching of DR proposed in the architecture illustrated in Figure 9 are being initially developed. HyperDE is a support environment for hypermedia application development using the design methods OOHDM (Object Oriented Hypermedia Design Method; Schwabe & Rossi, 1998) and SHDM (Semantic Hypermedia Design Method; Lima & Schwabe, 2003). In its current version, the designers dynamically develop their applications, in an interactive way, informing the items that compose the previously elaborated navigation model of their application. These items are recorded as instances of the metamodel used by the HyperDE environment, which is based on the metamodels that describe the primitives of the OOHDM and SHDM methods. In HyperDE+DR, this metamodel was extended with the representation model described by Kuaba ontology. This integration permits to capture and track part of the reasoning developed by the designer during the use of the environment to construct the hypermedia application.

The HyperDE+DR environment was designed for capturing part of the DR behind the design solutions used by the designers during the hypermedia application construction, because they have previously elaborated the navigational model and the context schema according to the OOHDM or SHDM method. Therefore, HyperDE+DR follows the current functioning of the environment HyperDE, on which it is based: the user builds the application after having concluded the previous steps of modeling (requirements gathering, conceptual model, navigational model, navigation context schema). In a future version, the environment will also support the design of the models. Therefore, HyperDE+DR is only a first experiment in the implementation of the layers of editing and searching of DR (Fig. 9) and automation of part of the rationale capture (questions and design ideas) enabled by the use of the metamodels' formal semantics.

In the HyperDE+DR, the designers construct their hypermedia applications by informing the set of objects (application concepts) that will be explored during the navigation and the way how they will be organized for navigation according to the primitives of the OOHDM and SHDM

https://doi.org/10.1017/S0890060408000279 Published online by Cambridge University Press

methods. This information is used by the environment to automatically generate some questions and ideas that must compose the application DR. The decision linking each question and the ideas that address it is also automatically generated. The explicit interaction of the designer is required only to add arguments for or against the created ideas and justify the decisions made. All arguments provided are included in the list of existing arguments and, thus, they can be reused for other ideas. The goal is that this process be (semi) automatic, once there is a database of predefined arguments according to the experience in using the primitives of the OOHDM and SHDM methods. The idea is to reduce the effort required from the users for capturing DR.

To support the design reuse through the use of DRs, the HyperDE+DR environment allows importing the recorded DRs. After selecting the application repository in which the union of the rationales will be done, the designer chooses the reasoning elements that must be imported. Besides selecting the elements that will be imported, the designer must also inform the identities between these elements and those in the application being developed. The identity specification is necessary when elements with different names represent the same concepts in the two applications being integrated.

As a further development of this implementation, the communication of the HyperDE+DR with the existing rationale processor is still being investigated. If this communication is not possible, the implementation of the computable operations proposed in this work will be reimplemented in the HyperDE+DR environment to support the rationales integration, currently is performed by the stand alone rationale processor.

6. RELATED WORK

The Kuaba approach provides an argumentation-based representation model for DR. Different from other argumentation-based models, such as IBIS, DRL, QOC e TEAM, the vocabulary used by Kuaba allows explicitly representing the decisions made by designers, including a specific element to describe decisions. Furthermore, Kuaba integrates these decisions with the argumentation used by designers during the design and with the artifact descriptions that result from these decisions, making the DR representation more complete.

Considering the DR approaches proposed for software design, Kuaba is similar to the Potts and Bruns model (Potts & Bruns, 1988), which was extended by Lee (1991) in defining DRL. Both integrate the DR representation with semantics provided by the software design methods. However, the Potts and Bruns model and the Kuaba approach differ in the way they use this semantics. In Potts and Bruns, the generic model entities are refined to accommodate a particular design method vocabulary for deriving new artifacts. For example, a new entity specific to the design method used is incorporated into the IBIS model. In the Kuaba approach the formal semantics of the metamodel prescribed by the design method is used in the instantiation of the reasoning elements (*Question* and *Idea*), which allows to automate the generation of part of the rationale that is informed by designers during the artifact design. In other words, Kuaba works at the metamodel level, whereas the Potts and Bruns approach works at the model level, requiring a change in the vocabulary for each different software design method.

Many systems have been also proposed to support DR. However, they usually do not permit the DR processing by computable operations. In Peña-Mora and Vadhavkar (1997), the authors proposed a framework combining DRIM (Design Recommendation and Intent Model) with design patterns to offer active assistance to software designers in designing reusable software systems. This combined approach leads to the "patterns by intent" approach. This approach refers to the process of selecting patterns based on their initial intents and then refining the choice of the patterns by specific constraints. Although this approach focuses on software reuse, it does not address the integration of rationales proposed in this work to create new software artifacts. Basically, the framework supports the reuse of components by recording and allowing the retrieval of decisions made during the software design process.

The SEURAT (Software Engineering Using Rationale) system (Burge & Brown, 2004) has architecture similar to the architecture of the integrated support environment presented in this work. However, SEURAT supports the use of rationale to identify inconsistencies during the software maintenance process. Differently from the environment proposed here, SEURAT does not consider computable operations over the DR as a support for the design reuse. Similarly to the Kuaba approach, the SEURAT system also propose the integration of the software design tools with tools for supporting the capture and representation of DR. But, SEURAT does not contain a component equivalent to the rationale processor developed in this work, capable of processing the rationales integration to support reuse.

Compendium and DRed are argumentation based systems that allow DR to be captured graphically. In both, all the reasoning elements (questions, ideas, arguments, decisions) are created, positioned, and linked manually by the user. In the integrated design environment proposed in this article, a great part of the rationale (questions and design ideas) is automatically obtained from the design metamodel used while the designers are designing the artifacts. In other words, the rationale is captured as the design proceeds according to the artifact design options selected by the designers at each moment in the design tool. Therefore, the designers do not need to know the elements of the Kuaba representation model. They need inform only their arguments and the justifications for the decisions made. Furthermore, neither one of them support computations (or reasoning) over the recorded rationales, as Kuaba does.

7. CONCLUSIONS

In this article we have proposed a new approach for the DR representation to support the reuse of model-based designs,

https://doi.org/10.1017/S0890060408000279 Published online by Cambridge University Press

particularly, software design. To allow a more effective use of DR to support design reuse, this approach integrates the DR representation model defined by the Kuaba ontology with the formal semantics provided by the metamodel of the design method or modeling language used for describing the artifact being designed. This integration makes the DR representations more specific according to the design methods and enables a new type of design reuse, where rationales can be integrated and re-employed in designing a new artifact.

The examples of DR representation considering different design metamodels show that the Kuaba representation approach can also facilitate the DR capture, because it allows automating part of generation of DR. 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. This automated support can reduce the overheads of DR authorship, because a great part of the rationale (questions and design ideas) is automatically obtained from the design metamodel used. However, the implementation of this automated support is still in its early days, and empirical evaluation must and will follow.

In addition, we have also proposed the conceptual architecture of an integrated design environment for supporting the capture, representation, and processing of DR. The proposed environment integrates the software design tools with a rationale processor capable of generating and integrating DR representations during the design process using a set of rules and computable operations. Thus, the design reuse is supported by the computational processing of the recorded DR in the new artifacts production.

Our current research includes the investigation of the use of the Kuaba approach to represent DR considering other activities of the software development process, the implementation of the proposed conceptual architecture in the HyperDE+DR environment and in other software design tools, and the investigation of visualizations techniques to support the presentation of DR using the Kuaba graph. We are also investigating the use of Kuaba in other domains, with different kinds of metamodels, such as engineering and geophysics. The goal here is to further experiment with the semantics of the metamodels and the degree of automation it enables within the support environment. A first study has been done in a large oil company, in which we are using the Kuaba approach to capture the rationale behind the best practices used by the company in oil well engineering. This initial rationale will be used to support the formalization of the metamodels used in these domains.

REFERENCES

Bracewell, R.H., Ahmed, S., & Wallace, K.M. (2004). DRed and design folders: a way of capturing, storing and passing on knowledge generated during design projects. *Design Automation Conf., ASME Design Engineering Technical Conf.*, Salt Lake City, UT.

Burge, J., & Brown, D.C. (2004). An integrated approach for software design checking using rationale. *Design Computing and Cognition*, pp. 557–576. New York: Kluwer Academic.

Conklin, J., Selvin, A., Buckingham, S.S., & Sierhuis, M. (2003). Facilitated hypertext for collective sensemaking: 15 years on from gIBIS. *Proc. LAP'03: 8th Int. Working Conf. Language-Action Perspective on Communication Modelling*, Tilburg, The Netherlands.

Goel, V., & Pirolli, P. (1989). Motivating the notion of generic design within information processing theory: the design problem space. *AI Magazine 10*, 19–36.

Hubka, V., & Eder, E.W. (1996). *Design Science: Introduction to Needs, Scope and Organization of Engineering Design Knowledge*, 2nd ed. London: Springer–Verlag London Limited.

Jacobson, I., Booch, G., & Rumbaugh, J. (1999). *The Unified Software Development Process*. Reading, MA: Addison–Wesley.

Kifer, M., & Lausen, G. (1989). F-Logic: a higher-order language for reasoning about objects, inheritance and scheme. *ACM SIGMOD*, pp. 134–146.

Kunz, W., & Rittel, H.W.J. (1970). *Issues as Elements of Information Systems*, Working Paper 131. Berkeley, CA: University of California, Berkeley, Institute of Urban and Regional Development.

Lacaze, X. (2005). *Conception rationalisée pour les systèmes interactifs—une notation semi formelle et un environnement d'édition pour une modélisation des alternatives de conception*. PhD Thesis. Université Toulouse I.

Lee, J. (1991). Extending the Potts and Bruns model for recording design rationale. *Proc. 13th Int. Conf. Software Engineering*, pp. 114–125, Austin, TX.

Lee, J. (1997). Design rationale systems: understanding the issues. *IEEE Expert 12(13)*, 78–85.

Lima, F., & Schwabe, D. (2003). Application modeling for the semantic web. *Proc. LA Web 2003*, p. 93. Taiwan: IEEE-CS Press.

MacLean, A., Young, R., Bellotti, V., & Moran, T. (1991). Questions, options, and criteria: elements of design space analysis. *Human–Computer Interaction 6(3–4)*, 201–250.

Medeiros, A.P. (2006). *Kuaba: an approach for representation of design rationale for the reuse of model-based designs.* PhD. Thesis. PUC Rio de Janeiro, Department of Informatics.

Medeiros, A.P., Schwabe, D., & Feijó, B. (2005). Kuaba ontology: design rationale representation and reuse in model-based designs. In *Proc. 24th Int. Conf. Conceptual Modeling (ER2005), Lecture Notes in Computer Science*, Vol. *3716*, pp. 241–255. Berlin: Springer–Verlag.

Nilsson, N. (1986). *Principles of Artificial Intelligence*. San Mateo, CA: Morgan–Kaufman.

Noy, N.F., Sintek, M.; Decker, S., Crubézy, M., Fergerson, R.W., & Musen, M.A. (2001). Creating semantic web contents with Protégé-2000. *IEEE Intelligent Systems 16(2)*, 60–71.

Nunes, D.A., & Schwabe, D. (2006). Rapid prototyping of web applications combining domain specific languages and model driven design. *Proc. 15th Int. Conf. World Wide Web*, pp. 889–890. New York: ACM Press.

OMG. (2003). *Unified Modeling Language Specification. Version 1.5.*

Peña-Mora, F., & Vadhavkar, S. (1997). Augmenting design patterns with design rationale. *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing 11(2)*, 93–108.

Potts, C., & Bruns, G. (1988). Recording the reasons for design decisions. *Proc. 10th Int. Conf. Software Engineering*, pp. 418–427, Singapore.

Santos, D.R.G. (2007). *Support for recording and using design rationale for web application design, Rio de Janeiro*. Master's Dissertation. PUC Rio de Janeiro, Department of Informatics.

Schön, D. (1983). *The Reflective Practitioner: How Professionals Think in Action*. New York: Basic Books.

Schwabe, D., & Rossi, G. (1998). An object-oriented approach to Web-based application design. *Theory and Practice of Object Systems (TAPOS)*, 207–225.

Simon, H.A. (1981). *The Sciences of the Artificial*, 2nd ed. Cambridge, MA: MIT Press.

Winograd, T. (1996). *Bringing Design to Software*. Reading, MA: Addison–Wesley.

Adriana Pereira de Medeiros is a Researcher with the TecWeb Laboratory Research Group, Department of Informatics, Catholic University, Rio de Janeiro. She received a PhD in computer science. Adriana has worked in software development projects for 4 years and has been teaching since

https://doi.org/10.1017/S0890060408000279 Published online by Cambridge University Press

2001. She has been investigating the use of DR to support the reuse of model based designs. Dr. Pereira de Medeiros' main research interests include software engineering, DR, ontology, knowledge management, metamodeling, and Web engineering.

Daniel Schwabe is a Professor of informatics at the Catholic University, Rio de Janeiro. He attained his PhD in computer science from UCLA in 1981. Daniel has been investigating the design and implementation of information systems seen as part of human–machine teams that perform tasks to solve problems, integrating formal and informal knowledge. The former is expressed through computational models, often integrated with the latter through hypermedia models. Current Web-based applications are a particular case of this type of system. Dr. Schwabe has investigated the use of DR as a natural way to record reuse design knowledge, which seamlessly integrates with the underlying domain model.

https://doi.org/10.1017/S0890060408000279 Published online by Cambridge University Press