ISSN 0103-9741

Monografias em Ciência da Computação
n° 14/06

# Ontology Driven Design Rationale Reuse

**Adriana Pereira de Medeiros**
**Daniel Schwabe**

Departamento de Informática

**PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO**

**RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453-900**

**RIO DE JANEIRO - BRASIL**

Monografias em Ciência da Computação, No. 14/06 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena April, 2006

# Ontology Driven Design Rationale Reuse*

Adriana Pereira de Medeiros

Daniel Schwabe

adri@inf.puc-rio.br, dschwabe@inf.puc-rio.br

**Abstract.** There are several proposals in the literature for representing design rationale. However, most of the existing representation schemas do not describe design rationale using a formal representation language having an expressive semantics that allows performing computable operations to support its reuse. In this paper we present the conceptual architecture of an integrated design environment that implements different operations to support recording design rationale, as well as reuse of design rationale during software design. This environment uses the formal language defined by the Kuaba ontology and the formal models of the artifacts as defined by design methods. They are used to represent the design alternatives considered by designers and the decisions made by them during the design process. This design rationale representation enables a new type of design reuse, where rationales are integrated and re-employed in designing a new artifact.

**Keywords**: Design Rationale, Design Reuse, Ontology, Knowledge Representation.

**Resumo**. Existem várias propostas na literatura para representar *design rationale*. No entanto, a maioria dos esquemas de representação existentes não descrevem *design rationale* usando uma linguagem de representação formal com uma semântica expressiva que permita realizar operações computáveis para apoiar o seu reuso. Neste artigo apresentamos a arquitetura conceitual de um ambiente de design integrado que implementa diferentes operações para apoiar a representação e o reuso de *design rationale* durante o design de software. Este ambiente usa a linguagem formal definida pela ontologia Kuaba e os modelos formais dos artefatos definidos pelos métodos de design. Eles são usados para representar as alternativas de design consideradas pelos projetistas e as decisões tomadas por eles durante o processo de design. Esta representação de *design rationale* possibilita um novo tipo de reuso de design, no qual *rationales* são integrados e re-empregados no design de um novo artefato.

**Palavras-chave**: *Design Rationale*, Reuso de Design, Ontologia, Representação de Conhecimento.

* Trabalho patrocinado pelo Ministério de Ciência e Tecnologia da Presidência da República Federativa do Brasil (e agência de fomento e o número do processo, se aplicável). (Em Inglês: This work has been sponsored by the Ministério de Ciência e Tecnologia da Presidência da República Federativa do Brasil)

**In charge for publications**:

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22453-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3114-1516 Fax: +55 21 3114-1530
E-mail: bib-di@inf.puc-rio.br

# 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 and Lai, 1991]. It includes also information about the other alternatives considered, the tradeoffs evaluated, and the argumentation that led to the decision. This information can be valuable, even critical, to various people who deal with the artifact. 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.

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. Therefore, recording the DR during the design process is critical to allow its reuse.

The research on DR normally involves three main aspects: its capture, its representation and its use. Generally, the representation schema determines the methods used to capture and retrieve the DR, enabling its use in designing new artifacts.

There are several proposals in the literature for representing DR, such as Issue Based Information System (IBIS) [Kunz and Rittel, 1970], Procedural Hierarchy of Issues (PHI) [McCall, 1991], Questions, Options and Criteria (QOC) [MacLean et al., 1991] and Active Design Documents (ADD) [Garcia and Howard, 1992]. Some of them were created specifically for software design. These include the Potts and Bruns model [Potts and Bruns, 1988], the Decision Representation Language (DRL) [Lee and Lai, 1991] and the Design Recommendation and Intent Model (DRIM) [Pena-Mora and Vadhavkar, 1996]. However, the representations generated by these proposals are not described by a formal representation language with an expressive semantics. Consequently, is not possible to perform computable operations on the recorded rationale and, therefore, to reuse DR. Moreover, 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. The DR is represented based only on the descriptions provided by the designers, which results in a generic representation. In other words, it is not possible to leverage the semantics of the artifact provided by the formal model that describes it.

In this work we propose an integrated design environment to support the reuse of DR in a special kind of design domain, which we call "model-based design". Model-based design is a category of design problems that can be seen as an instantiation process of a meta-model. This meta-model represents the formal models used to describe artifacts. 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. An example of such a formal model is the UML specification language [OMG, 2003] used to describe a class diagram.

There are several approaches that regard the design of human-computer interfaces as an instance of model-based design. As such, the approach described here is also applicable, although the examples shown will exemplify hypermedia navigation aspects.

A formally defined DR representation, integrating the formal semantics of the artifacts being designed, can enable the reuse of designs of artifacts by the integration of their rationales. This integration of different rationales is possible only if they are represented using the same representation scheme(s), and the artifacts are built based on the same type of formal model and represent the same application domain. For instance, it is possible to integrate the rationales of two context schemas created with the Object Oriented Hypermedia Design Method (OOHDM) [Schwabe and Rossi, 1998] (formal model) to design the navigation of a Web application for a CD store (same domain), where these rationales are represented as instances of the same ontology.

This formal representation of DR enables a type of design reuse at the highest abstraction level, where rationales can be integrated and re-employed in designing a new artifact. Starting with existing artifacts, the designer can analyze their rationales and decide to integrate them to get a more complete design solution. She or he 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.

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.

In this paper, we first describe briefly the Kuaba1 ontology, a formal representation language for design rationale that we proposed in [Medeiros, Schwabe and Feijó, 2005], and how this language can be used considering the artifacts formal model to represent DR. Next, we present the conceptual architecture of the integrated design environment that is being built to support designers through processing of formal DR representations. We present also the operations implemented in this environment to support the software design reuse using these representations. Finally, we conclude by discussing related work, pointing out further work, and drawing some conclusions.

## 2 The Kuaba Ontology

Kuaba is a knowledge representation model for DR described in F-logic [Kifer and Lausen, 1989]. Kuaba is composed by a vocabulary and a set of rules that allows describing DRs with an expressive semantics and performing queries, inferences and computable operations over the recorded content, to support the use of DR.

The vocabulary described by Kuaba extends the argumentation structure of IBIS by explicating the representation of the decisions made during design and their justifications, and the relations between the argumentation elements and generated artifacts. Figure 1 shows the elements of the vocabulary defined by the Kuaba ontology, using the UML graphical notation to help visualization. Notice that such object oriented representation is used only as a suggestion of illustration of ontology vocabulary; some relations and constraints were hidden to simplify the presentation.

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

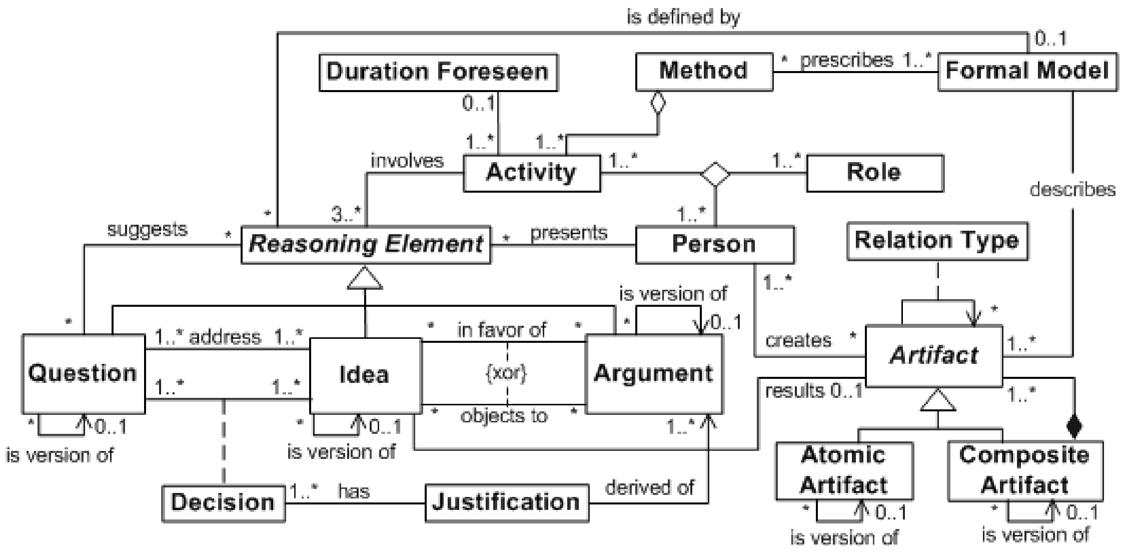


**Figure 1. Kuaba Ontology Vocabulary**

Briefly described, the Kuaba ontology vocabulary represents the reasoning elements used by the designers during the design process; the decisions made by them; information about the artifacts that result from this reasoning; and information about the design activity. 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 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. The Decision element records 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 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; hasArgument->>argument;
     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 Design Rationale with Kuaba

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. Consider the following scenario: a designer needs to model the navigation that will be used by users in a Web application of a CD Store and decides to use the OOHDM method to guide her or his design. When the designer chooses this design method, s/he indirectly determines the formal model(s) that will be used to describe the artifacts. This formal model specifies, to a great extent, the questions and ideas that the designer can propose, since they are pre-defined by this model. Figure 2 shows part of the formal model prescribed by the OOHDM method to specify the navigation of a Web application.

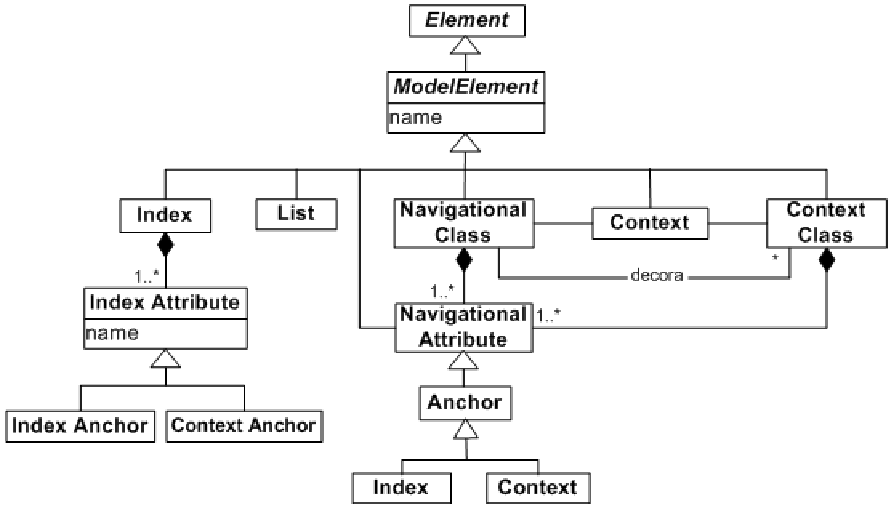


**Figure 2. Partial OOHDM formal model for Navigation**

According the OOHDM formal model, an element in a navigation model is either an index, a list, a navigational class, a context or an InContext class. Contexts define the set of navigation objects that will be explored by the user in each moment. The navigation objects are defined based on the navigational classes related to the context. For example, the context "CDs of the artist Daniela Mercury" represents the set of CDs that will be accessible to the user, after s/he has selected her or his favorite artist. This context is composed by objects of the navigational class CD. This class defines the CD information that the user will have access.

The access to the objects of a context can be done through an index or through an anchor. An index is a set of ordered objects, where each object has at least an attribute of the type anchor (*seletor*). This anchor creates a link to another object that can be a context object or an object of another index. The previously quoted context "*CDs of the artist Daniela Mercury*" can be accessed, for example, by an index of artists in which one of the objects is "*Daniela Mercury*". Figure 3 shows the context schema that represents this navigation. Note that the "*CDs by Artist*" context includes a group of possible contexts, whose objects are defined by the name of the artist selected by the user in the "*Artists*" index.

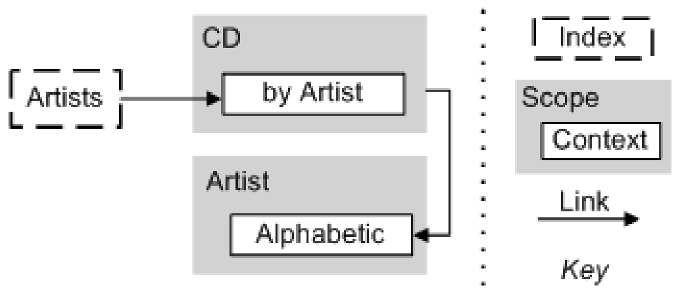


**Figure 3. Context Schema Example. The key to the notation is summarized on the right hand side.**

An InContext class is defined as a "decorator" of a navigational class with information that can only be accessed in a specific context. For instance, we can define an InContext class with an attribute of type anchor to give access to the artist's biography only when the user is browsing a CD in the "*CDs by Artist*" context. In this way, this anchor will be presented, in this context, as part of the CD information, allowing the user to navigate to the "*Artist Alphabetic*" context.

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. Figure 4 shows a graphical representation we have created to help visualizing instances of the Kuaba ontology, showing the portion of the DR regarding the alternatives and decisions that resulted in the context schema shown in Figure 3.

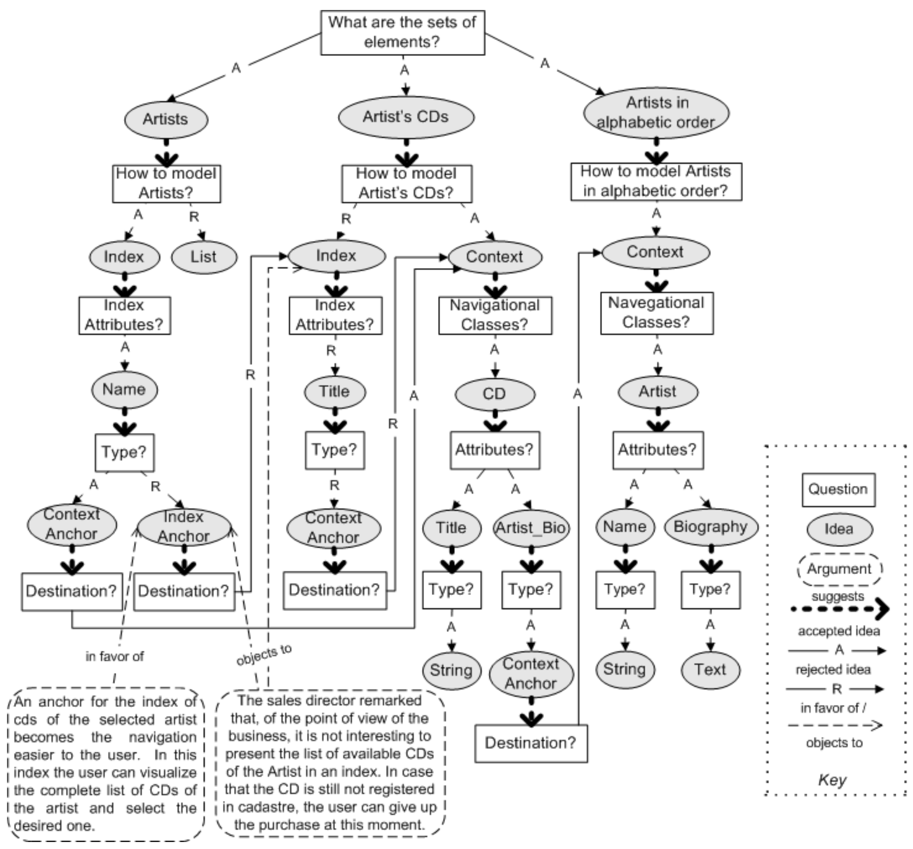

What are the sets of elements?
Artists
Artist's CDs
Artists in alphabetic order
How to model Artists?
How to model Artist's CDs?
How to model Artists in alphabetic order?
Index
List
Context
Index Attributes?
Navigational Classes?
Navegational Classes?
Name
Title
CD
Artist
Type?
Attributes?
Context Anchor
Index Anchor
Artist_Bio
Biography
Destination?
String
Text
in favor of
objects to
An anchor for the index of cds of the selected artist becomes the navigation easier to the user. In this index the user can visualize the complete list of CDs of the artist and select the desired one.
The sales director remarked that, of the point of view of the business, it is not interesting to present the list of available CDs of the Artist in an index. In case that the CD is still not registered in cadastre, the user can give up the purchase at this moment.
Question
Idea
Argument
suggests
accepted idea
A
rejected idea
R
in favor of /
objects to
Key

**Figure 4. A portion of a DR representing the navigation in the Figure 3**

In this representation, the root node is an initial question (represented as rectangles), "*What are the sets of elements?*", which is addressed by the ideas "*Artists*", "*Artist's CDs*" and "*Artist in alphabetic order*", 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.

Once these initial ideas for the sets of elements have been established, the designer must decide how each one of them will be modeled using the primitives of the OOHDM method to make up the final artifact, the navigation model. This next step is represented in Figure 4 by the "suggests" relation, which determines questions entailed by ideas – "*How to model Artists?*", "*How to model Artist's CDs?*" and "*How to model Artist in alphabetic order?*".

The possible ideas that address these questions are determined by the OOHDM formal model for defining navigation models – sets of elements can essentially be modeled as an index, a context, or a list. Accordingly, the OOHDM formal model is instantiated into the "*Index*" and "*Context*" ideas linked to the "*How to model Artist's CDs?*" node. Strictly speaking, the designer should consider all the other alternatives proposed by the OOHDM, but for the sake of simplicity here we have shown only these two.

Observe that when the designer considers the ideas of modeling the sets "*Artists*" and "*Artist's CDs*" as indices, the questions "*Index Attributes?*" are immediately suggested. This occurs because an index must be associated to one or more attributes, according the OOHDM formal model shown in Figure 2. These questions are addressed by the ideas that correspond to the respective attributes of these indices. Notice that these attributes can be of type "index anchor" or "context anchor" with different destinations, depending on the navigation that the designer desires. The proposed ideas for the type of the attributes "*Name*" and "*Title*" of the indices "*Artists*" and "*Artist's Cds*", respectively, and the ones that address the question "*Destination?*" in Figure 4, define two different solutions to the navigation in the Web application designed by the designer.

The first navigation solution leads from the "*Artists*" index, through the "*Name*" anchor, to the "*Artist's CDs*" context. This alternative was accepted by the designer (represented by the labels "A" on the arrows between questions and ideas), corresponding to the solution represented in the context schema in Figure 3. Observing the rationale of this solution, shown in Figure 4, we can perceive that the designer decided to include the attribute "*Artist_Bio*" of type "*Context Anchor*" in the navigational class CD. This anchor allows the user to navigate from the information of a CD to the artist's biography in the "*Artists in alphabetic order*" context.

In the second navigation solution, rejected by designer (labeled "R" in Figure 4), the user selects the desired artist in the "*Artists*" index to access a list of CDs of this artist, in the "*Artist's Cds*" index. Then, s/he could select the desired CD in this index to access its detailed information, in the "*Artist's Cds*" context. Observe that the idea "*Artist's CDs*" had two design alternatives (index and context) and both could have been accepted by the designer as solutions to model this set. Figure 5 shows the navigation solution rejected by the designer, according the DR representation shown in Figure 4.

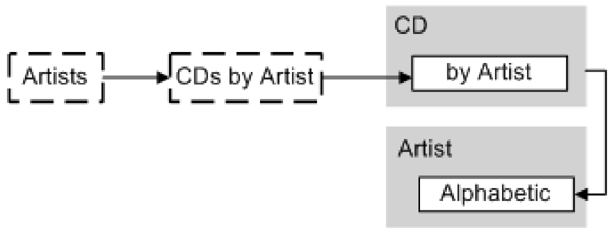


**Figure 5. Context schema of the navigation solution rejected by the designer**

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 design process. After defining the design options for each domain idea, the designer has to record the arguments for and against each option (represented as dashed rectangles in Figure 4), and decide which design alternative will be used in the final artifact. The "Argument" element allows designers to record the experiences and the knowledge that they are employing in the artifact design.

The sub-graph 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". 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 independently accepted as a solution to the question. This kind of information allows us to define rules that can suggest decisions about the acceptance or rejection of the solution ideas proposed. 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 with 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 "Name" as an attribute of the type context anchor in the Artists index was the first one made by the designer, a support system could apply the rule above, and automatically propose that the idea of modeling "Name" as an index anchor 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 question "*Index Attributes?*", rejecting "Name" altogether. In any case, the support environment can apply consistency rules defined by the Kuaba ontology, as well as those expressed by the particular formal artifact representation being used. Therefore, the order of accepting or rejecting an idea does not affect the rationale represented.

## 3 Designs Reuse using Design Rationale

Formalizing DR representation using the language defined with Kuaba enriched with the semantics of the formal model of the artifact as defined by the design method allows design reuse, which is reuse of a new kind. This reuse is possible through the integration of existing DRs to start the design of a new artifact. This type of reuse requires different kinds of operations on the recorded DR. These operations involve matching instances of the Kuaba ontology (DR representations) to compose a more complete solution of design.

In this section we show the conceptual architecture of an integrated environment that supports recording DR, as well as reuse of DR during software design. This environment uses the artifact formal model to suggest design alternatives at each step of the design process. It also records the choices made by the designer using the representation language defined by the Kuaba ontology. In addition, the environment supports also searching for existing DRs, and reusing design alternatives considered in them for the new artifact being designed.

Since most software design support tools already use some kind of formal description of the artifacts being designed, we propose to extend them to allow their integration with the DR "processor" that is capable of processing representations using Kuaba. The extension enriches the design tools by adding two layers to support the editing and searching of DR. In the edition 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. These layers are being developed initially to the HyperDE [Nunes and Schwabe, 2006], a support environment to the hypermedia applications design using the OOHDM or SHDM [Lima and Schwabe, 2003] methods. The Figure 6 shows the main components of the proposed architecture.

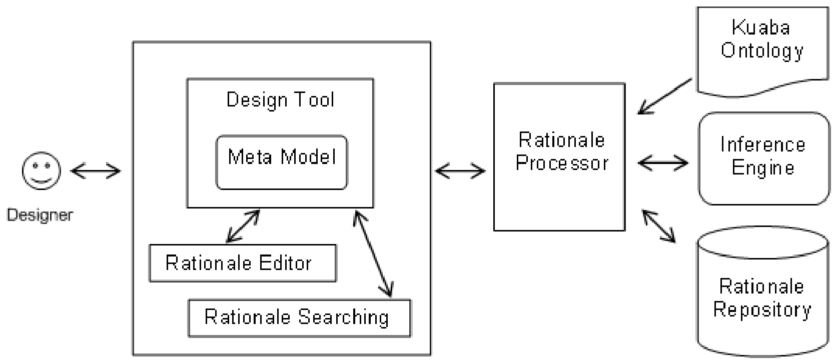


**Figure 6. Architecture of an Environment Supporting DR**

In the proposed architecture, the design tool transfers the design options and the rationale information provided by the designer to the rationale "processor", 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 rationale.

## 3.1 The Rationale Processor

The representation and the integration of DR 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 the operations to create instances of an ontology have already been implemented in several ontology editors such as Protege [Noy et al., 2001]. The operations to perform the integration of instances of the same ontology have not been considered in majority of the proposed systems to support the use of DR. Basically, these operations involve the search, copy, substitution and union of elements of the instances being integrated.

Search operations allow the designer 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 are answers to this question that have arguments in its favor.

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 the other 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, 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.

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[2] that translate F-Logic in tabled Prolog code and process this code in the deductive system XSB[3]. These rules are defined based on the reasoning elements of the Kuaba ontology vocabulary. They consist basically in the recursive processing of the various sub-trees of questions and solution ideas that compose the DR representations.

### 3.1.1 Integrating Design Rationale

Consider again the scenario in which the designer needs to model the navigation of a Web application for a CD store. Since the online stores domain is a common domain in software design, the designer decides to perform a search for existing designs, trying to find similar artifacts, before she or he begins a new design. As a result, s/he finds the artifacts shown in Figures 3 and 7. These artifacts represent different design solutions to model the navigation of Web applications using the OOHDM method.

2 http://flora.sourceforge.net/

3 http://xsb.sourceforge.net/

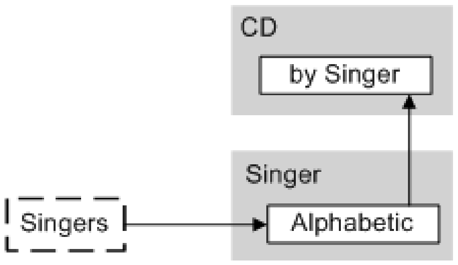


**Figure 7. Context schema of the navigation solution considering singers**

Analyzing the artifacts found, the designer notices that the author of the second artifact, shown in Figure 7, considered the set "*Singers*" instead of considering the set "*Artists*", and decided to use another solution. In this navigation solution, initially the user selects the favorite singer in the "*Singers*" index to access to the information of this singer in the "*Singer Alphabetic*" context. From this information s/he can navigate to the "*CDs by Singer*" context to have access to the detailed information of that CD.

After analyzing the DR representations of these artifacts, shown partially in Figures 4 and 8, the designer decides to integrate the navigation solutions used by other designers to start her or his design with a bigger set of options.

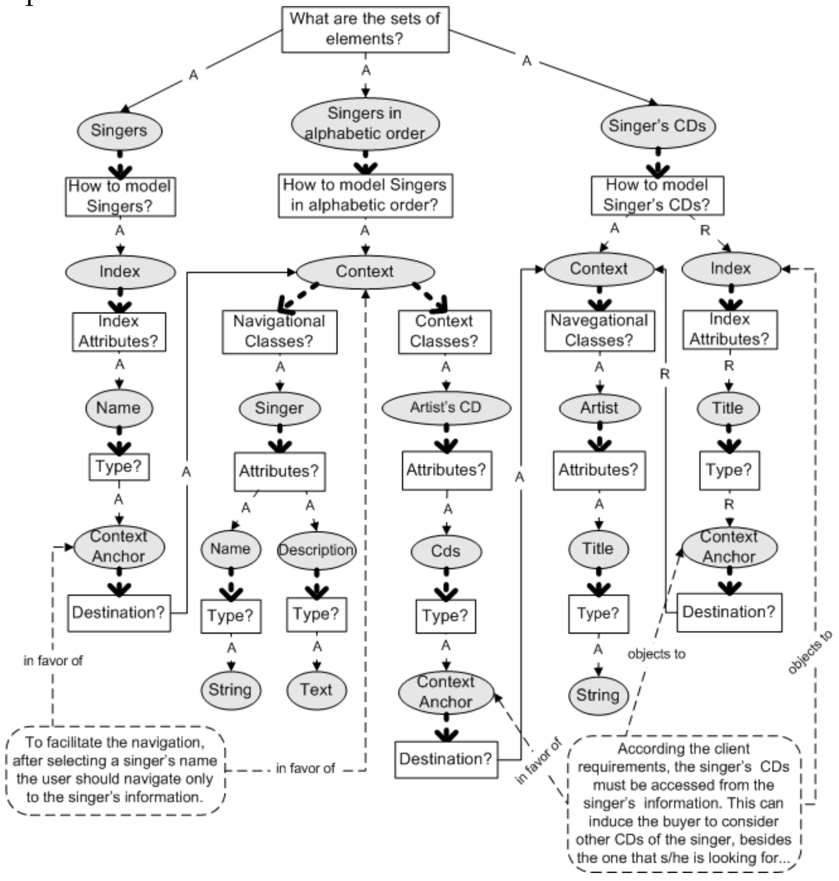

What are the sets of elements?
Singers
Singers in alphabetic order
Singer's CDs
How to model Singers?
How to model Singers in alphabetic order?
How to model Singer's CDs?
Index
Context
Context
Index
Index Attributes?
Navigational Classes?
Context Classes?
Navegational Classes?
Index Attributes?
Name
Singer
Artist's CD
Artist
Title
Type?
Attributes?
Attributes?
Attributes?
Type?
Context Anchor
Name
Description
Cds
Title
Context Anchor
Destination?
Type?
Type?
Type?
Type?
Destination?
String
Text
Context Anchor
String
Destination?
in favor of
in favor of
in favor of
objects to
objects to
To facilitate the navigation, after selecting a singer's name the user should navigate only to the singer's information.
According the client requirements, the singer's CDs must be accessed from the singer's information. This can induce the buyer to consider other CDs of the singer, besides the one that s/he is looking for...

**Figure 8. A portion of a DR representing the navigation in the Figure 7**

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 the domain ideas of the two representations; and the union of the sub-trees of elements (equivalent or not) of these ideas.

The definition of the base representation and the equality specification between domain ideas are performed by the designer. In this example, assume that the designer defined the DR representation shown in Figure 4 as base and specifies that the sets "*Artists*", "*Artists in alphabetic order*" and "*Artists' CDs*" of this representation are equal to the sets "*Singers*", "*Singers in alphabetic order*" and "*Singers' CDs*" of the representation shown in Figure 8. This equality is specified in Flora2 using the predicate ":=:" as

```
artists:=:singers
```

Based on this information, the rationale processor first identifies the equivalent questions and design ideas, applying the equivalence rules shown below. These rules examine the sub-trees of the domain ideas comparing the questions and ideas of the two representations. Questions are considered equal if they are suggested by equivalent ideas, and ideas are considered equal if they have the same text and address equivalent questions.

```
Q1:=:Q2 :- Q1[isSuggestedBy->>I1]@mod1,
           Q2[isSuggestedBy->>I2]@mod2,
            I1:=:I2, not (I1=I2).
I1:=:I2 :- I1[address->>Q1]@mod1, I2[address->>Q2]@mod2,
         I1[hasText->X]@mod1,I2[hasText->Y]@mod2, X=Y,
         Q1:=:Q2, not (Q1=Q2).
```

When the rationale processor finds an equivalent idea in the sub-trees, it simply copies the arguments of this idea from the original representation to the base representation. For example, the "*Context Anchor*" ideas in the sub-trees of the ideas "*Artists*" (Figure 4) and "*Singers*" (Figure 8) are equivalent. Thus, the arguments presented by the designer to the "*Context Anchor*" idea in the representation shown in Figure 8 are copied to the sub-tree of its equivalent idea in the base representation (Figure 4).

After identifying and treating the equivalent ideas, the rationale processor identifies the domain ideas that are different (non-equivalent) in the two representations and copies them with their respective sub-trees to the base representation. The operations used by the processor to treat these ideas are shown below. Observe that the domain ideas are obtained from the root question.

```
?- L=collectset{I|I[address->>whatElementsSets]@mod1},
   L2=collectset{I|I[address->>whatElementsSets]@mod2},
   get_non_equivalent(L,L2,List), copy_idea(List).
```

In the code above, the rationale processor creates two lists (L, L2) with the domain ideas that address the root question "*What are the sets of elements*" of the two DR representations being integrated. The *get_non_equivalent* operation processes these lists returning a new list with the different ideas. The *copy_idea* operation, shown below, copies the domain ideas in this list to the base representation. For each idea copied, notice that the processor copies also its arguments (*copy_argument* operation) and its respective sub-trees, retrieved from the questions suggested by this idea (*copy_question* operation).

```
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),
                                if Idea1[suggests->>Q1]@mod1
                                then
                                  (insert{Idea1[suggests->>{Q1}]@mod2 | Idea1[suggests->>Q1]@mod1},
                                   LQ=collectset{Q2|Q2[isSuggestedBy->>Idea1]@mod1},
                                   copy_question(LQ)))
                              else
                                (insert{Idea1[address->>{Q}]@mod2 | Idea1[address->>Q]@mod1}),
                              copy_idea(Tail).
```

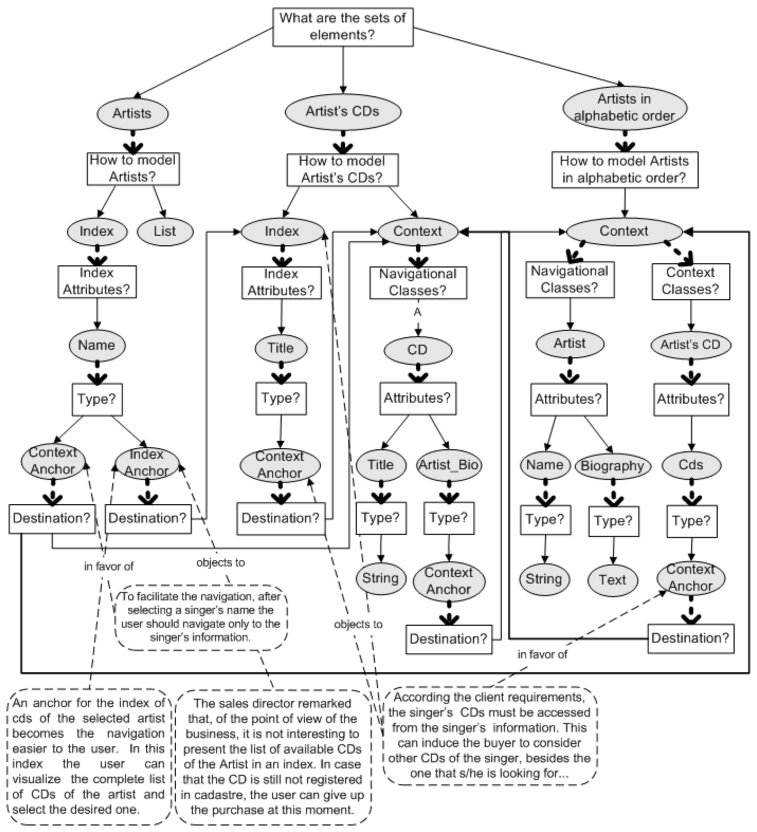
Finishing the integration, the rationale processor treats the design ideas based on formal model of the artifact that are not equivalent. For this, it retrieves again the sub-trees of the domain ideas and copies the design ideas not considered so far to the base representation. The code shown below is an example of the operations used by the processor to perform this treatment.

```
?- L=collectset{I|I[address->>Q[isSuggestedBy->>singers]]@mod1},
   L2=collectset{I|I[address->>Q[isSuggestedBy->>artists]]@mod2},
   get_non_equivalent(L,L2,List), transfer_idea(List, artists).
```

Observe that the sub-trees treated by the rationale processor are composed by the ideas that address the questions suggested by the domain ideas "*Singers*" and "*Artists*", respectively. The operation *get_non_equivalent* is performed again returning the list of different design ideas that is processed by the operation *transfer_idea*. The parameter "*artists*" represents the domain idea of the base representation that will receive the sub-tree of questions and design ideas considered to the idea "*Singers*". The association of the copied sub-tree with the domain idea of the base representation is performed by the code in bold in the *transfer_idea* operation shown below.

```
transfer_idea([], _).
transfer_idea([Idea1|Tail], Base) :-
          insert{(Idea1:idea,Idea1[hasText->X, hasCreationDate->Y, isInvolved->Z])@mod2 |
                  Idea1[hasText->X, hasCreationDate->Y, isInvolved->Z]@mod1},
          insertall{Idea1[address->>{Q}]@mod2 | Q[isSuggestedBy->>Base]@mod2},
          insertall{Idea1[suggests->>{Q1}]@mod2 | Idea1[suggests->>Q1]@mod1},
          insertall{Idea1[hasArgument->>{A}]@mod2 |Idea1[hasArgument->>A]@mod1},
          LQ=collectset{Q2|Q2[isSuggestedBy->>Idea1]@mod1},
          copy_question(LQ),
          LA=collectset{A1|A1[inFavorOf->>Idea1]@mod1;A1[objectsTo->>Idea1]@mod1},
          copy_argument(LA,Idea1),
          transfer_idea(Tail, Base).
```

Observing the rationale representations shown in Figures 4 and 8 we can notice, for instance, that the navigation options from the ideas "*Context Anchor*" in the sub-trees of the ideas "*Artists*" and "*Singers*" are different. In the first representation (Figure 4), the destination of this anchor is the set "*Artists' CDs*" modeled as a context. In the second one the destination is the "*Singers in alphabetic order*" context. Since these navigation options represent different solution ideas, the rationale processor copies this last option to the integrated DR, as shown in Figure 9.



To facilitate the navigation, after selecting a singer's name the user should navigate only to the singer's information.

An anchor for the index of cds of the selected artist becomes the navigation easier to the user. In this index the user can visualize the complete list of CDs of the artist and select the desired one.

The sales director remarked that, of the point of view of the business, it is not interesting to present the list of available CDs of the Artist in an index. In case that the CD is still not registered in cadastre, the user can give up the purchase at this moment.

According the client requirements, the singer's CDs must be accessed from the singer's information. This can induce the buyer to consider other CDs of the singer, besides the one that s/he is looking for...

**Figure 9. Example of an integrated DR**

In the integrated DR, all navigation options considered in the artifacts shown in Figures 3, 5 and 7 were joined in one unique DR representation. Notice that the decisions made for the reused artifacts and their justifications 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 her or his objectives.

## 4 Related Work

The Kuaba ontology provides an argumentation-based representation model for DR. Different from other argumentation-based models, as IBIS, PHI and DRL, Kuaba allows represent explicitly the decisions made by the designers, including in its vocabulary a specific element to describe decisions. Furthermore, Kuaba integrates these decisions with the argumentation used by the designers during the design process and with the descriptions of the artifacts that results from them.

Compared with the approach used by the ADD system, Kuaba can be also considered a model-based approach, since it uses the formal model of the artifact to record DR. However, ADD is used to capture DR in routine, parametric designs where the designer defines the values that specify the artifact being designed. Differently from ADD, Kuaba represents DR in software designs which focus is in the structural aspects of the artifacts.

Considering the DR approaches proposed specifically for software design, we can say that Kuaba is similar to the Potts and Bruns model, that was extended by [Lee, 1991] in the creation of the Decision Representation Language (DRL). Both integrate the DR representation with software engineering methods. The Potts and Bruns model and the Kuaba ontology differ in the way they use software engineering methods. In that model, the generic model entities are refined to accommodate a particular design method's 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 ontology vocabulary 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 are informed by designers during the design.

In [Pena-Mora and Vadhavkar, 1996], the author proposed a framework combining DRIM 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 and Brown, 2004] has an architecture similar to the architecture of the integrated support environment presented in this work. However, SEURAT supports the use of rationale only to identify inconsistencies during the software maintenance process. It does not consider computable operations over the DR as a support for the reuse of software artifacts.

## 5 Conclusion

In this paper we have proposed a new way of reusing designs in Software Design domain with the support of an integrated design environment. The proposed environment integrates the software design tools with a rationale processor capable of generating and integrating DR representations during the design process. The reuse of DR in the design of new artifacts is supported by different operations implemented as rules in the Flora-2 language.

To permit a more effective reuse of DR, we have also proposed the use of the formal models of artifacts to represent DR using the vocabulary defined in Kuaba ontology. We believe that the use of formal models of artifacts 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 development of the layers of edition and searching for the HyperDE environment and the integration of this environment with the rationale processor to validate the reuse of software designs through the integration of existing DR representations, with semi-automated support.

## ACKNOWLEDGMENTS

Daniel Schwabe was partially supported by a grant from CNPq - Brazil.

## References

BURGE, J.; BROWN, D.C. **An Integrated Approach for Software Design Checking Using Rationale**. Design Computing and Cognition 2004, Kluwer Academic Publishers, 557-576, 2004.

GARCIA, A. C. B.; HOWARD, C. H. **Acquiring design knowledge through design decision justification**. Artif. Intell. for Eng. Design, Analysis and Manuf., 6(1), 59-71, 1992.

KIFER, M.; LAUSEN, G. **F-Logic: A Higher-Order Language for Reasoning about Objects, Inheritance and Scheme**. ACM SIGMOD May , 134-146, 1989.

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.

LEE, J. **Extending the Potts and Bruns Model for Recording Design Rationale**. In Proceedings of the 13th International Conference on Software Engineering, Austin, TX, 114-125, 1991.

LEE, J.; LAI, K. **What's in Design Rationale**. *Human-Comput. Interaction, 6* (3-4), 251-280, 1991.

LIMA, F.; SCHWABE, D. **Application Modeling for The Semantic Web**. *Proc. LA Web 2003,* IEEE-CS Press (2003)

MCCALL, R. J: **PHI: A conceptual foundation for design hypermedia**. Design Studies, No.12 (1), 30-41, 1991.

MACLEAN, A.; YOUNG, R.; BELLOTTI, V. and MORAN, T. **Questions, Options, and Criteria: Elements of Design Space Analysis**. Human-Computer Interaction, No. 6 (3-4), 201-250, 1991.

MEDEIROS, A. P.; SCHWABE, D.; FEIJÓ, B. **Kuaba Ontology: Design Rationale Representation and Reuse in Model-Based Designs.** In: Proceedings of the 24th International Conference on Conceptual Modeling (ER2005), Klagenfurt, Austria, Lecture Notes in Computer Science 3716, Springer-Verlag, ISBN 3-540-29389-2, p. 241-255, 2005.

NILSSON, N. **Principles of Artificial Intelligence**. Morgan Kaufman Publishers, 476p, 1986.

NOY, N. F.; Sintek, M.; Decker, S.; Crubézy, M.; Fergerson, R. W. and Musen, M. A. **Creating Semantic Web Contents with Protégé-2000**. IEEE Intelligent Systems Vol. 16, No 2, Special Issue on Semantic Web, 60-71, 2001.

NUNES, D. A.; Schwabe, D. **Rapid Prototyping of Web Applications using Domain Specific Languages and Model Driven Design**. *Submitted to WWW 2006.* Available at http://server2.tecweb.inf.puc-rio.br:8000/projects/hyperde/trac.cgi/attachment/wiki/DownloadArchives/MDD%20DSL%20RapidPrototyping%20.pdf

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

PENA-MORA, F.; VADHAVKAR, S.: **Augmenting Design Patterns with Design Rationale**. Artificial Intelligence for Engineering Design, Analysis, and Manufacturing, 93-108, 1996.

POTTS, C.; BRUNS, G. **Recording the Reasons for Design Decisions**. In Proceedings of 10th International Conference on Software Engineering, Singapore, 418-427, 1988.

SCHWABE, D.; ROSSI, G. **An object-oriented approach to Web-based application design**. Theory and Practice of Object Systems (TAPOS), 207-225, 1998.