Poetry Generation in COLIBRI

Belén Díaz-Agudo, Pablo Gervás, and Pedro A. González-Calero

Dep. Sistemas Informáticos y Programación
Universidad Complutense de Madrid, Spain
{belend, pgervas, pedro}@sip.ucm.es

Abstract. CBROnto is an ontology that incorporates common Case-Based Reasoning (CBR) terminology and serves as a domain-independent framework to design CBR applications. It is the core of COLIBRI, an environment to assist during the design of knowledge intensive CBR systems that combine cases with various knowledge types and reasoning methods. CBROnto captures knowledge about CBR tasks and methods, and aims to unify case specific and general domain knowledge representational needs. CBROnto specifies a modelling framework to describe reusable CBR Problem Solving Methods based on the CBR tasks they solve. This paper describes CBROnto's main ideas and exemplifies them with an application to generate Spanish poetry versions of texts provided by the user.

1 Introduction

Even though any Case-Based Reasoning (CBR) system relies on a set of previous specific experiences, its reasoning power can be improved through the explicit representation and use of general knowledge about the domain. Our approach to CBR is towards integrated knowledge based systems (KBS) that combine case specific knowledge with models of general domain knowledge. Our ongoing work is the development of COLIBRI (Cases and Ontology Libraries Integration for Building Reasoning Infrastructures), an environment to assist during the design of knowledge intensive CBR (KI-CBR) systems [2,4] that combine concrete cases with various knowledge types and reasoning methods.

COLIBRI's architecture is influenced by knowledge engineering approaches such as Role Limiting Methods [11], CommonKADS [12] or Components of Expertise [13], where a KBS is viewed as consisting of separate but interconnected collaborating components. Typically, components of a KBS include domain knowledge and Problem Solving Methods (PSMs), that represent commonly occurring, domain-independent problem-solving strategies.

COLIBRI views KI-CBR systems as consisting of collaborating knowledge components, and distinguishes different types of knowledge [14]. *Ontologies* describe the structure and vocabulary of the *Domain Knowledge* that refers to the actual collection of statements about the domain. *Tasks* correspond to the goals that must be achieved. *PSMs* capture the problem-solving behavior required to perform the goals of a task. And *Inferences* describe the primitive reasoning steps during problem solving.

The core of the COLIBRI architecture is CBROnto, an ontology that incorporates common CBR terminology and problem solving knowledge that serves as a domain-independent framework to design KI-CBR systems [5]. CBROnto is formalized in Loom [10] a Description Logics (DLs) system on top of which COLIBRI is built. From a general perspective CBROnto can be considered a knowledge representation ontology [14] that captures representation primitives commonly used in the case-based representation languages. Our aim is to propose a rich framework to represent cases based on the terminology from the CBROnto together with a reasoning system that works with such representations. We work with a structured case representation where individuals are concept instances and concepts are organized in a hierarchy with inheritance. In our approach, cases are linked within a semantic network of domain knowledge and will be described by using both the domain vocabulary provided by the domain model, and the CBR vocabulary provided by CBROnto. Another facet of CBROnto is as an unifying framework that structures and organizes different types of knowledge in KI-CBR systems according to the role that each one plays. CBROnto terms serve as a bridge that allows the connection between expert knowledge and previously defined domain ontologies, and helps in discovering and modelling the knowledge needed for a CBR system. As a last facet, CBROnto is a task and method ontology whose contents are described in the next section.

2 CBROnto as a Task and Method Ontology

A useful way of describing problem solving behavior is in terms of the tasks to be solved, the goals to be achieved, the methods that will accomplish those tasks, and the domain knowledge that those methods need. A description along these lines is referred to as a *knowledge level description*. Although various authors have applied knowledge level analysis to CBR systems, the most important of these efforts is the well-known CBR task structure developed by Agnar Aamodt and Enric Plaza [1] influenced by the Components of Expertise Methodology [13]. At the highest level of generality, they describe the general CBR cycle by four tasks: *Retrieve* the most similar case/s, *Reuse* its/their knowledge to solve the problem, *Revise* the proposed solution, and *Retain* the experience. The four CBR tasks each involve a number of more specific subtasks. There are methods to solve tasks, that either decompose a task in subtasks or solve it directly. CBROnto includes a task ontology influenced by Aamodt and Plaza's structure at the first level and identifies a number of alternative methods for each task, where each one of the methods sets up different subtasks, that must be solved in their turn. This kind of task-method-subtask analysis is carried on to a level of detail where the tasks are primitive with respect to the available knowledge.

CBROnto includes a library of PSMs associated to the CBROnto tasks. The CBROnto PSMs are described by relating them to terms within its ontology of tasks, methods and domain characteristics. The method ontology includes method description language terms used to formalize PSMs and defines concepts and relationships that are used by the methods. In [5] CBROnto's method description language is described together with a mapping mechanism to bridge the gap between domain knowledge and PSMs based on a DLs classification strategy. The explicit representation of knowledge requirements based on the CBROnto terminology makes easy to identify and solve the PSM's lack of knowledge.

COLIBRI uses an automatic, general and recursive task resolution mechanism that starts with the task to solve, and finds the alternative methods whose competence subsumes this task. Decomposition methods divide the task in subtasks and the resolution process is applied recursively for each subtask. Resolution methods finalize recursion and solve the task:

```
Resolve (iT)
1.Get the method individual to resolve the task: iM
2.Get the method functional specification iFS
3.Get the method requirements iReq
4. If  iM is a decomposition_method,
       Applying iFS answers with the sequence of subtaks
       to solve: iST1, ...., iSTn
       ResolveSeq( iSTn , ResolveSeq( iSTn 1 , ... , ResolveSeq( iST2 , Resolve( iST1))...))
   Else                                  % iM is a resolution method
       Applying iFS with iReq solves the task
```

When designing a new CBR application, COLIBRI offers alternative methods whose competence subsumes this task. The CBR system designer fixes one (or more) preferred method to solve the task in the application, and configure it according to the required behavior. We distinguish between three types of inputs (or requirements) to configure a method:

– The method *knowledge requirements* represent knowledge elements that the method uses and that must be defined before the method can work, such as similarity measures or relevance criteria for retrieval.
– The method *input requirements* are external inputs to the method, i.e., they are not represented as explicit elements integrated within the domain knowledge. These inputs are fixed by the CBR application designer and will be shared by all the method executions.
– The method *parameter requirements* are also external inputs to the method, but they change within different executions of the method (for example, the query). They are specified by the final user of the CBR application.

The following sections introduce some of the CBROnto methods organized around the tasks they resolve, i.e. their competence.

2.1 Retrieval Methods

Retrieval methods are those whose competence is the retrieval task. CBROnto formalizes several retrieval methods that have been described in [3,4] and that are summarized here. Each retrieval method decomposes the retrieval task into one or more of the following subtasks that are solved themselves by methods that depends on the retrieval method:

– *Obtain cases.* Select the initial case set (CS) to apply the following subtasks.
– *Assess similarity.* Assess the similarity between the query and each one of the cases in CS.

– *Select cases.* Select the case or cases to be returned as retrieval result based on the similarity assessment.

The *computational method* computes all the similarity values during the retrieval process. The CBROnto similarity framework allows representing, in a declarative way, several alternatives to compute numerical similarity values for complex case representations, where the similarity knowledge contained in the domain knowledge base participates in the similarity assessment.

The *relevance criteria method* uses the query language to enable the user to describe the current situation and interests. Relevance criteria are defined as the criteria according to which the system asserts that a case is relevant to a problem and more relevant than any other cases. CBROnto uses relevance criteria expressed in first order logic using Loom, which allows to express complex conditions that involve any number of cases interrelated by multiple relationships.

Other method used in the example is the *representational method* that assigns similarity meaning to the path joining two cases in the case organization structure and the domain knowledge base, and retrieval is accomplished by traversing that structure starting from the position of the query. We have applied this choice using an *instance classification method* [4] that uses the subsumption links to define the distance between two individuals. The usefulness of this kind of approach will depend on the knowledge structure where the cases are located.

Besides the retrieval methods, CBROnto –mainly through its relation hierarchy– allows to represent different *similarity types* depending on the contributing terms, namely it allows defining different similarity types depending on a semantic classification of the attributes –relations– below the CBROnto terms.

2.2 Adaptation and Revision Methods

CBROnto's adaptation methods are based on using domain independent knowledge in the form of transformation operators [8]. Adaptation knowledge is made up of a set of abstract transformation operators (as SUBSTITUTE, ADD and REMOVE) and memory search strategies to find the information in the domain knowledge, needed to apply these operators.

In this paper we describe one adaptation method that is based on substituting some elements in the retrieved case according to the query. Substitutes are searched in the domain model by using memory search strategies. CBROnto provides domain independent strategies and the mechanisms for the domain expert to add specific domain memory search strategies. Besides, other memory search strategies are learned from user's interactions. A memory search strategy goal is to find an item satisfying certain restrictions. That is why some of the methods used to solve the memory search task are shared with the case retrieval task. For example, to find candidates to substitute element i in the solution we can use the computational method with domain specific similarity measures using i as the query, or use the instance classification method to get instances classified near i in the hierarchy, or use the relevance criteria method to retrieve instances satisfying a given criteria of similarity regarding i.

The method of adaptation by substitution leads to the following subtasks:

- *Copy solution task*
- *Modify solution task* leads to the subtasks (cycle):
  - *Find adaptable parts task*
  - *Apply substitution* leads to the subtasks (cycle):
    - *Find substitutes task*
    - *Select substitute task*
    - *Substitute item task*
    - *Validate task*

If the memory search process performed during the find substitutes task does not find acceptable items, the substitute item task will not be performed. After adaptation, the revision task (when manual) allows the user to substitute an item. The learning methods learn both the failed and the successfully applied memory search strategies, and the manually added substitute. Revision Methods are those whose competence is the *Revise task*, that is decomposed in two subtasks: system revision and user revision. Only one of them is mandatory, when both are specified they are solved in sequence.

The revision task leads to the following subtasks (cycle). Note that each loop of the cycle finds one problem and tries to repair it.

- *Evaluation task*
- *Repair task* leads to the subtasks:
  - *Find repair strategy task*
  - *Apply repair strategy task*

In this paper we do not explain user revision method, but exemplify the self revision method where the evaluation and repair tasks are solved by the self evaluation and repair methods, respectively. Self evaluation is based on DLs classification as it compares the classification between the adapted case and the retrieved case. The concepts under which the retrieved case is classified in the domain model are used as declarative descriptions of the properties that should be maintained by the adapted case after transformations. Namely, substitutions must not alter the classification of the case. If they do, the case requires reparation. the repair needs are identified by the automatic classification of the adapted case under a certain type of problem. Repair strategies are linked to the concepts representing adaptation problems, and, thus, can be directly obtained after every classification of the problem case. When it fails the user will be in charge of repairing the case. Our approach is related with the one proposed in [9] where adaptation cases include knowledge about one-step transformations to solve a type of problems.

This generic method depends on the type of problems and repair strategies that are specifically identified and represented for each domain. We are using an idea that is common for other automatic revision methods, namely, the need of an explicit representation of the system task, i.e., the goals that are required for a case to be correct. Our explicit model of the domain allows representing these goals as classification properties over the adapted case, i.e., the case is correct if it is classified according to certain concepts. We typically use the classification of the retrieved case as the goals to be satisfied by the adapted case. The adapted case is initialized to an exact copy of the retrieved case that, classified under the concepts; after the resolution of the modify task it might not be recognized as an instance of some of these concepts. The repair task is in charge of repairing these failures. The semantic definitions of the domain concepts allows to know why the individual has not be recognized as an instance of a certain concept.

The self repair method uses as the correction measure the conjunction of the concept definitions that must be satisfied by a correct case after adaptation. The evaluation method is based on the Loom instance recognition mechanism.

3 Implementing Poetry Generation with COLIBRI

Composing poetry is an art not particularly well suited for algorithmic formulation. However, for the specific case of formal poetry, it does have certain overall characteristics that have to be met by any candidate solution. What is particularly interesting from the point of view of illustrating the operation of COLIBRI is the fact that the description of these characteristics involve a complex set of interacting concepts that have to be taken into account.

Another reason involved in the choice of poetry generation as an example of the use of COLIBRI is the existence of previous work along similar lines [6,7] –developed in terms of CBR but not adhering to the CBROnto concepts and the COLIBRI way of chaining them together to form a CBR application– provides a useful reference point from which to discuss the possible advantages and disadvantages of the approach.

The specific process that has been chosen to illustrate this point is conceptually based on a procedure universally employed when not-specially-talented individuals need to personalise a song, for instance, for a birthday, a wedding, or a particular event: pick a song that everybody knows and rewrite the lyrics to suit the situation under consideration. This particular approach to the problem of generating customised lyrics or poetry has the advantage of being easily adapted to a formal CBR architecture. No claims whatsoever regarding the general suitability of this approach for poetry composition in a broad sense should be read into this particular choice.

3.1 Basic Rules of Spanish Poetry

Formal poetry in Spanish is governed by a set of rules that determine a valid verse form and a valid strophic form. A given poem can be analysed by means of these rules in order to establish what strophic form is being used. Another set of rules is applied to analyse (or *scan*) a given verse to count its metrical syllables.

Given that words are divided into syllables and each word has a unique syllable that carries the prosodic stress, the constraints that the rules have to account for are the following:

Metric Syllable Count. Specific strophic forms require different number of syllables to a line. Metric syllables may be run together thereby shortening the syllable count of the line involved. When a word ends in a vowel and the

```
(defconcept Poem :is
  (:and Domain-concept
        (:all has-stanza Stanza)
        (:at-least 1 has-stanza)))

(defconcept Stanza :is
  (:and Domain-concept
        (:all has-line Poem-line)
        (:at-least 1 has-line)))

(defconcept Poem-line :is
  (:and Line
        (:all has-word Word-occurrence)
        (:at-least 1 has-word)
        (:the first-word Word-occurrence)
        (:the rhyme String)
        (:the syllables Number)
        (:all follows-on Poem-line)
        (:at-most 1 follows-on)))
```

```
(defconcept Word-occurrence :is
  (:and Domain-concept
        (:all precedes Word-occurrence)
        (:at-most 1 precedes)
        (:the of-word Word)))

(defconcept Non-final-word-occurrence :is
  (:and Word-occurrence
        (:exactly 1 precedes)))

(defconcept Word :is
  (:and Domain-Concept
        (:the text String)
        (:the syllables Number)
        (:the stress Number)
        (:the rhyme String)
        (:the stVowel Number)
        (:the endVowel Number)
        (:all has-POSTag POSTag)
        (:at-least 1 has-POSTag)))
```

Fig. 1. Structural definitions for the poetry domain.

following word starts with a vowel, the last syllable of the first word and the first syllable of the following word constitute a single syllable. This is known as *synaloepha*, and it is one of the problems that we are facing.

Word Rhyme. Each strophic form requires a different rhyming pattern.

Stanza or Strophic Form. For the purpose of this application only poems of the following regular strophic forms are considered: *cuarteto*, a stanza of four lines of 11 syllables where the two outer lines rhyme together and the two inner lines rhyme together; and *terceto*, a stanza of three lines of 11 syllables where the either the two outer lines rhyme together or the three lines have independent rhymes.

3.2 Poetry Domain Knowledge Ontology

The COLIBRI approach to building KI-CBR systems takes advantage of the explicit representation of domain knowledge. allowing to integrate existing ontologies about a particular domain of application. To our regret, we were unable to locate an existing ontology about formal Spanish poetry. An initial sketch of such an ontology has been developed for purposes of illustration, resulting in a knowledge base containing 86 concepts, 22 relations and 606 individuals.

Figure 1 shows the Loom definitions needed to represent the structure of a poem, a text made up of words, and built up as a series of stanzas, which are groups of a definite number of lines of a specific length in syllables, satisfying a certain rhyme pattern. Going from the parts to the whole, each word is represented as an individual which is an instance of the domain concept *Word* and is described in terms of the following attributes: the name of that particular word (*text*), the number of syllables that the word has (*syllabes*), the position of the stressed syllable of the word counted from the beginning of the word (*stress*), the rhyme of the word (*rhyme*), whether the word begins with a vowel (*stVowel*), whether the word ends in a vowel (*endVowel*), and the part-of-speech tags associated with that word (*has-POSTag*).

```
(defconcept Terceto :is
  (:and Stanza
        (:exactly 3 has-line)
        (:the first-line Endecasilabo)
        (:the second-line Endecasilabo)
        (:the third-line Endecasilabo)))

(defconcept Endecasilabo :is
  (:and Poem-line
        (:fillers syllables 11)))

(defconcept Rhymed-poem-line :is
  (:and Poem-line
        (:exactly 1 rhymes-with)))
```

```
(defconcept Terceto-uno-tres :is
  (:and Terceto
        (:relates rhymes-with first-line
                              third-line)))

(defrelation rhymes-with :is
  (:satisfies (?x ?y)
    (:and (Poem-line ?x)
          (Poem-line ?y)
          (:for-some ?z
            (:and (rhyme ?x ?z)
                  (rhyme ?y ?z))))))
```

Fig. 2. Definition of a *terceto* stanza

In our model of the domain we distinguish between words –instances of *Word*– and particular word occurrences –instances of the domain concept *Word-occurrence*. In the representation of the poems we use a different individual for each occurrence of a particular word, though various individuals may be referring back to the same instance of *Word*. Each occurrence is related to the word it represents through the *of-word* relation, and with the word occurrence that follows it in a line through the *precedes*.

A line is represented as an instance of *Poem-line*, which, in addition to a number of word occurrences, represents the rhyme, the number of syllables, whether the sentence follows on onto the next line of the poem, and, for efficiency reasons, which one is the first word of the line. Each *Stanza* is built up from a number of *Poem-lines*, and each *Poem* is related to the stanzas that make it up.

Using the basic vocabulary we can define different types of stanza such as the one shown in Figure 2. A *Terceto* is defined as a stanza of three lines of eleven syllables. Although not shown in the figure, the relation *has-line* subsumes *first-line*, *second-line* and *third-line*, and, therefore, *Terceto* is a *Stanza* with *at-least* 1 *has-line*. The lines of a *Terceto* are *Endecasilabos* defined as *Poem-line*s where 11 is the value of the attribute *syllables*. Finally, a *Terceto-uno-tres* is a *Terceto* where the two outer lines rhyme together. The model identifies that two lines rhyme together when a common rhyme exists between them.

3.3 The Cases

Cases describe a solved problem of poem composition. We describe cases using the CBROnto case description language and domain knowledge terminology. Although different possibilities can be explored, for the sake of simplicity we choose a case where both description and solution is a given poem.

In COLIBRI the definition of the structure of the cases is part of the process of integrating the domain knowledge within CBROnto, in order to bridge the gap between domain terminology and CBR terminology. Integration is based on classification, domain concepts and relations are marked as subconcepts and subrelations of CBROnto concepts and relations. In this way, the domain-independent PSMs can be applied to the domain-specific information.

slgt2:PoetryCase						
description	poe-slgt2:Poem					
solution	poe-slgt2:Poem					
	has-stanza	st1-poe-slgt2:Stanza				
		first-line	l1-st1-poe-slgt2:Poem-line			
			first-word	no221:Word-occurrence		
			rhyme	ada		
			syllables	11		
			follows-on	l2-st1-poe-slgt2:Poem-line		
			has-word	no221:Word-occurrence		
			has-word	so_lo243:Word-occurrence		
			has-word	en413:Word-occurrence		
			has-word	plata485:Word-occurrence		
				precedes	o484:Word-ocurrence	
				of-word	plata:Word	
					text	plata
					syllables	2
					stress	1
					rhyme	ata
					stVowel	0
					endVowel	1
					has-POSTag	ADJGMS:POSTag
					has-POSTag	ADJGFS:POSTag
					has-POSTag	NCFS:POSTag
					has-POSTag	ADJGMP:POSTag
					has-POSTag	ADJGFP:POSTag
			has-word	o484:Word-occurrence		
			has-word	viola321:Word-occurrence		
			has-word	truncada122:Word-occurrence		
		second-line	l2-st1-poe-slgt2:Poem-line			
		third-line	l3-st1-poe-slgt2:Poem-line			

Fig. 3. Case Representation Example

The first thing to do in the design phase is to define a new type of case, i.e an specialization of the concept *Case*, and choose which concept within the domain model will represent a case description (mandatory) and which one will represent a solution (optional). Figure 3 shows a partial view of a case representing a poem (Each case that is added to the system adds an average of 50 individuals to the knowledge base). In the example, the new case type is *PoetryCase* whose *description* and *solution* are both the same instance of the *Poem*:

no sólo en plata o viola truncada
se vuelva mas tú y ello juntamente
en tierra en humo en polvo en sombra en nada [1]

Although no semantic information about the attributes is used here, the integration phase could also determine that, for instance, *precedes* and *follows-on* relations are a kind of *before* attribute, or that the relations *has-stanza*, *has-word*, and *has-line* are a kind of *has-part* attribute. That integration would provide the semantic roles to be used in the predefined *similarity types*. For adaptation purposes, we could also classify *follows-on* under *depends-on* CBROnto relation,

[1] not just to silver or limp violets // will turn, but you and all of it as well // to earth, smoke, dust, to gloom, to nothingness

to indicate that if a line which follows onto the next is modified, then the next one may be affected, and its adaptation should be considered.

3.4 Retrieval

The query is given as a sequence of words that we want to inspire our poem, and it is represented as an instance of *PoetryCase* with *description* but, obviously, without *solution*, consisting of just one line with the given *Word-occurrence*s.

In the example, the cases with the largest number of POS tag in common with the query should be retrieved. This way, it will be easy to substitute words in the retrieved poem with words from the query without loosing syntactic correctness. Therefore, the *obtain cases* retrieval subtask is easily defined as a LOOM query –a relevance criterion– that retrieves poems based on this requirement.

To solve the *select cases* retrieval subtask we select the cases to be retrieved by computing the similarity between the query and every retrieved case. For the example, we associate a similarity measure with the concept *PoetryCase* that collects the similarity among the *Words* of the *description*s of two poems. A similarity measure for the concept *Word* is also needed, taking into consideration all of the word attributes.

Given this configuration of the tasks, if we –carefully– choose as reference words to inspire our poem "*una boca ardiente pase techo y suelo*" we will retrieve the poem used as example in the previous section.

3.5 Adaptation

The high level idea of the adaptation process is to substitute as many words from the poem with words from the query, if possible in the same order as appearing in the query, without loosing the syntactic structure of the poem lines. We assume that the query is a meaningful sentence and, therefore, if we can accommodate those words into the poem in a similar order it is plausible to think that the new poem will reflect, to a certain extent, the original message in the query. In order to maintain the syntactic correctness of the poem, and taking into account that the system has no additional syntactic knowledge, we constrain substitutions to words with exactly the same POS tag. The adaptation algorithm runs, then, as follows: for every word in the poem, the first word in the query with the same POS tag is chosen as its substitute, if none exists then the word in the poem remains unchanged; for every substitution, the word from the query is removed so it is used only once. This process iterates until a whole cycle is done without substitutions or all the words in the query have been included in the poem. In the example this process results, in just one cycle, in:

no sólo en *boca y* viola *ardiente*
se *pase* mas tú y ello juntamente
en tierra en *techo* en *suelo* en sombra en nada [2]

[2] not just to mouth or burning violets // will pass, but you and all of it as well // to earth, shelter, dirt, to gloom, to nothingness.

where all the words from the query (italicized text) have been arranged except *una* because there was no determiner in the original poem.

In order to make this process possible, the designer needs to choose and configure the generic adaptation methods. The method of adaptation by substitution is chosen, which, as described in Section 2.2, leads to the subtasks: copy solution, find adaptable parts, and apply substitution.

There is only one method available for copying the solution, and it does not need any customization. The method for finding adaptable parts in the solution needs to know where in the solution can be accessed the candidates for substitution which, in this case, are the poem words. This information is provided to a new instance of the generic method through the input requirement *toadapt*. *toadapt* is parameterized with the chain of attributes which has to be composed in order to access to the poem words from the entity representing the whole case:

```
(put-input-requirements 'ifind_adaptable_parts_method
  '((toadapt '((solution)(stanza)(has-line)(has-word)(of-word)))))
```

Notice how we can profit from the *is-a* hierarchy of attributes, by using *has-line* which subsumes *first-line*, *second-line* and *third-line*. More sophisticated configurations could be provided for this method to indicate, for example, that only non final words are to be considered for substitution

```
(put-input-requirements 'ifind_adaptable_parts_method
  '((toadapt '((solution)(stanza)(has-line)
                (has-word Non-final-word-occurrence)(of-word)))))
```

which could be useful if we would like to maintain the rhyme of the final words.

The output of the previous method is the input to the method responsible for applying the substitutions. This method, as described in Section 2.2, is a cycle which iterates through the list of candidates for substitution: finding substitutes, selecting one of them, making the substitution, and validating it. Notice that this local validation –word-based– is different to the global validation described in the next section –poem line or poem-based–. The process terminates when the list of candidates gets exhausted, or when an iteration ends without substitutions.

In order to solve the task of finding substitutes, we need to choose and configure one of the available retrieval methods. In the example we are interested on finding words in the query with the same POS tag. We may choose the method to obtain items by classification in representational retrieval, where we look for words which POS tag attribute is classified under the same concept as the candidate for substitution. And, then, further parameterize this method to consider only those words included in the query, instead of the whole vocabulary:

```
(put-input-requirements
  'ifind_substitutes_items_by_classification_method
  '((unique_source '(query solution stanza has-line has-word of-word))))
```

If more than one substitute have been retrieved we need to apply a selection method to choose one of them. Applicable methods are: user selection, random selection, and similarity based selection. Since there is a similarity function associated with the concept *Word*, we could choose the similarity based selection method, so that the substitute would be selected taking into consideration all of the word attributes, apart from the POS tag which serves as the filter in the task of finding substitutes. Nevertheless, since substitutes are restricted to those appearing in the query, it is unlikely to find more than one, and, for the example to work, we just need random selection.

The last two tasks of making the substitution and validating it, are trivial in the example, since the old value is directly substituted by the new one, and no local validation is needed when we only want to maintain syntactic correctness which is guaranteed by employing words with the same POS tag.

The problem with this adaptation process is that, although it preserves the syntactic structure of the retrieved poem, its metric characteristics will be probably lost. As discussed above, this includes number of syllables per line, and rhyme of the final words of lines 1 and 3. The revision process repairs, whenever possible these characteristics.

3.6 Revision

After adaptation, the self revision method is used to solve the revision task, and is in charge of evaluating and, if needed, repairing the proposed solution. As it was described in Section 2.2, the self evaluation method compares the classification between the adapted case and the retrieved case. The concepts under which the retrieved case is classified in the domain model are used as declarative descriptions of the properties that should be maintained by the adapted case after transformations. In the example the retrieved case is recognized as *PoetryType*, the solution is recognized as *Poem*, the stanza is recognized as a *Terceto-uno-tres*, and each one of its poem lines are recognized as *Endecasilabo*s. Besides, the first and third poem lines are recognized as *Rhymed-poem-line.*

Before substitutions, the copy of the retrieved case that will be adapted has the same classification. If substitutions provokes a change in the concepts the system recognizes for a certain individual, the evaluation task classifies this individual below a concept representing a type of problem. The type of problems (subconcepts of the *FailureType* CBROnto concept) have associated a repair strategy that tries to put the individual back as an instance of the goal concept.

In the example, we have substituted a word by other of the same POS tag but possibly different rhyme, or number of syllables. Two problem types has been identified during the design of the application. The first one is called *Rhyme-failure* meaning that a poem line should rhyme and it does not. The domain concepts involved in this failure are the concepts that represent the rhyming strophic forms *Terceto-uno-tres* and *Cuarteto*, and the *Rhymed-poem-line* concept. When an individual leaves these concepts it is recognized as an instance of *Rhyme-failure.* The second one is called *Syllables-count-failure* and the domain concepts involved are *Endecasilabo* –11 syllables– and *Octosilabo* –8 syllables.

Next step is to define repair strategies associated to problem types. Each strategy is represented as an instance of the concept *Repair-strategy*, that are linked to the *FailureType* concept through the relation *has-repair-strategy.* The self repair method divides the repair task into two subtasks: find strategy and apply strategy. The two subtasks are solved in a loop that finishes when no failures are found and no strategies can be applied. To find the next strategy that will be applied, the find strategy method searches in the hierarchy rooted by *FailureType* and finds the strategy that is associated to the most specific concept in the hierarchy of problem types. The next step is applying the strategy to the individual that is classified below the problem type concept.

The revision is implemented as the process of substituting words in the adapted poem so that the detected problems can be repaired. For the example, we are defining a repair strategy to the problem type *Rhyme-failure*. The instances of this problem type are the stanzas where we want to modify certain words. We can take advantage of the CBROnto PSMs whose competence is the adaptation task. The method splits the task into subtasks whose methods has to be configured (as we did to configure them for the adaptation task). In order to repair the rhyme of the final words we have to select a word to substitute and then find its replacement:

– The items to be substituted in the stanza are the final word occurrences of the poem lines that belong to the *Rhyme-failure* concept. Besides, in order not to loose the effects of the adaptation, in the revision we are constrained to substitute only those words which do not appear in the query. This is configured as:

```
(put-input-requirements 'ifind_adaptable_parts_method
  '((toadapt '((has-line rhyme_failure)
             (has-word (:and Final-word-occurrence
                             Not-query-component)) (of-word)))))
```

– The find substitutes method is configured to use a relevance criteria method that finds substitutes with the same POS tag and the proper rhyme depending on the strophic form. The algorithm is to select the first word of the broken rhyme which does not come from the query, and if both words were in the query then to ask the user.

In the example we have lost the rhyme of the final words of first and third lines, and the first line has 9 syllables instead of 11. The word *nada* is the one to be substituted as it does not belong to the query. In order to find a replacement for this word, we search for a word with the following requirements: has the same POS tag as *nada*, and rhymes with *ardiente*. If more than one word were retrieved then the selection process would come into play, and the most similar to *nada*, according to the rest of word attributes, would be selected. If there is no word under the given requirements, then the process fails and the user could be asked for help. In the example, the only word retrieved is *serpiente*, which substitutes *nada* and repairs the rhyme of the final words of first and third lines.

The next loop tries to repair the number of syllables of the poem lines. The order between the repairing processes depend on the classification of the problem types and is important because the solution to a problem may cause a new problem of different type. In the example, since *serpiente* has 1 more syllable than *nada* the third poem line is not an *endecasilabo* (11 syllables) any more.

When fixing the number of syllables of a poem line we take into account that this is the second process of revision, and therefore an additional constraint is not to substitute the last word of a rhymed line.

In order to repair the first line, we select as candidate for substitution the shortest word which was not in the query and is not a final word: *no* and *en*. Then we search for words with one more syllable than the candidate one, and the same POS tag. For *no* we find no candidate, but we find *para* as a substitute for *en* (both are prepositions). With this substitution, the number of syllables is automatically recomputed, and although *para* has only one more syllable than *en*, the new line is an *endecasilabo* because by substituting *en* we have also break the synaloepha between *sólo* and *en*.

In order to repair the third line, we select as candidate for substitution the longest word which was not in the query and is not a final word: *tierra* and *sombra*. We choose the first one and then search for a word with one syllable (one less than *tierra*), with the same POS tag, and ending with a vowel in order not to break the synaloepha. The retrieval word is *tía* which repairs the poem line into 11 syllables, leading to poem (words marked with * were obtained in the revision process):

no sólo para* *boca y* viola *ardiente*
se *pase* mas tú y ello juntamente
en tía* en *techo* en *suelo* en sombra en serpiente* [3]

4 Conclusions and Future Work

We have described CBROnto's task and method ontology and its application to a CBR system to generate Spanish poetry versions of texts provided by the user. The problem chosen as an example had already been tackled elsewhere using CBR. The approach described here presents several advantages with respect to the original one. First, the use of the frame of tasks and methods of CBROnto allows a very clear explicit representation of all the decisions that need to be taken. The domain allows many possible ways of solving the problem at each of the stages, and ad hoc development without a systematic approach run a risk of losing sight of where a design decision has been taken; thereby closing off a possible avenue of exploration for a solution. Second, this very set of tasks and methods provides a set of useful tools that may help to solve particular problems, or provide ideas for developing new solutions.

Two lines of research are now open for further work. First, more than one case may be used for adaptation. For instance, one case may be used to provide the structure of the result, whereas other cases are used to provide the required vocabulary. The variety of methods of CBROnto allow different criteria to be applied when retrieving cases for each of the possible purposes.

An important improvement that is envisaged is the incorporation of an ontology for the terms of the language being employed. As the reader may have

[3] not just to mouth or burning violets // will pass, but you and all of it as well // to girl, shelter, dirt, to gloom, to snake.

noticed, the example presented in the paper has been carefully chosen to exemplify the available mechanisms, and, of course, not every query would result in a meaningful poem with the right metric. Having a representation of the meanings of words as well as the other information already in use in the system would present several important advantages: 1) The ontology may provide the information needed to modify the structure selectively, for instance by replacing a masculine singular noun with a feminine singular noun if they refer to the same concept; 2) During retrieval a semantic description for words introduces the possibility of recovering cases where a similar meaning is conveyed with completely different syntactic constructions; 3) Additionally, it could act as a mechanism for extracting from particular cases the relevant relations between case description and case solution, to be used in selecting an adequate vocabulary.

References

1. A. Aamodt and E. Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications*, 7(i), 1994.
2. B. Díaz-Agudo and P. A. González-Calero. An architecture for knowledge intensive CBR systems. In E. Blanzieri and L. Portinale, editors, *Advances in Case-Based Reasoning – (EWCBR'00)*. Springer-Verlag, Berlin Heidelberg New York, 2000.
3. B. Díaz-Agudo and P. A. González-Calero. Classification based retrieval using formal concept analysis. In *Procs. of the (ICCBR 2001)*. Springer-Verlag, 2001.
4. B. Díaz-Agudo and P. A. González-Calero. A declarative similarity framework for knowledge intensive CBR. In *Procs. of the (ICCBR 2001)*. Springer-Verlag, 2001.
5. B. Díaz-Agudo and P. A. González-Calero. CBROnto: a task/method ontology for CBR. In *CBR Track (FLAIRS) accepted to be published*. 2002.
6. P. Gervás. Wasp: Evaluation of different strategies for the automatic generation of spanish verse. In *Proceedings of the AISB-00 Symposium on Creative & Cultural Aspects of AI*, pages 93–100, 2000.
7. P. Gervás. An expert system for the composition of formal Spanish poetry. *Journal of Knowledge-Based Systems*, 14(3–4):181–188, 2001.
8. P. A. González-Calero, M. Gómez-Albarrán, and B. Díaz-Agudo. A substitution-based adaptation model. In *Challenges for Case-Based Reasoning - Proc. of the ICCBR'99 Workshops*. University of Kaiserslautern, 1999.
9. D. B. Leake, A. Kinley, and D. C. Wilson. Acquiring case adaptation knowledge: A hybrid approach. In *Proceedings of the thirteenth National Conference on Artificial Intelligence*, pages 684–689, Menlo Park, CA, 1996. AAAI Press.
10. R. Mac Gregor and R. Bates. The loom knowledge representation language. ISI Reprint Series ISI/RS-87-188, University of Southern California, 1987.
11. J. McDermott. Preliminary steps towards a taxonomy of problem-solving methods. In S. Marcus, editor, *Automating Knowledge Acquisition for Knowledge-Based Systems*. Kluwer Academic Publishers, Boston, 1988.
12. T. Schreiber, B. J. Wielinga, J. M. Akkermans, W. V. de Velde, and R. de Hoog. CommonKADS: A comprehensive methodology for KBS development. *IEEE Expert*, 9(6), 1994.
13. L. Steels. Components of expertise. *AI Magazine*, 11(2):29–49, 1990.
14. G. Van Heijst, A. Schreiber, and B. Wielinga. Using explicit ontologies in knowledge based systems development. *International Journal of Human and Computer Studies*, 46(2/3), 1997.