Chapter 6

Applications

The purpose of this chapter is to provide a set of examples of how diverse applications have been developed using Noos by several persons at the IIIA.

All these applications are described in detail in other publications. The goal of this chapter is to describe their main characteristics. Following this purpose, and after a brief introduction of the task that the application solves, we will focus on three aspects of the applications: how the domain knowledge required for problem solving methods is modeled using Noos representation capabilities, which problem solving methods are developed to perform the task, and which learning methods are incorporated for solving the task.

In this chapter we will present six applications developed using Noos: CHROMA, SPIN, SHAM, GYMEL, *Saxex*, and NoosWeb.

CHROMA is a system for recommending a plan for the purification of proteins from tissues and cultures using chromatographic techniques developed by Eva Armengol [Armengol and Plaza, 1994] [Armengol, 1997].

SPIN is a sponge identification system for a class of marine sponge species (the family of *Geodiidae*) also developed by Eva Armengol [Armengol, 1997].

SHAM is a tool to help a non expert musician to harmonize melodies using background musical knowledge developed by Martí Cabré [Cabré, 1996].

GYMEL is also a system for harmonization of melodies. Nevertheless, GYMEL harmonize melodies using a case-based reasoning approach. GYMEL has been developed by Jordi Sabater [Sabater, 1997].

Saxex is a case-based reasoning system for generating expressive performances of melodies based on examples of human performances that are represented as structured cases. I have developed *Saxex* in the context of my M.Sc.Thesis in Computer Music [Arcos, 1996] [Arcos et al., 1997b].

Finally, NoosWeb is a WWW interface to Noos applications supporting the same interaction capabilities provided in the Noos window-based graphical interface developed for Apple computers. NoosWeb is not exactly an application such as other applications presented in this chapter. We have included NoosWeb for presenting the accessibility facilities of Noos and for describing the set of small improvements performed in Noos to support remote calls. NoosWeb has been developed by Francisco Martín [Martín, 1996].

6.1 CHROMA

Noos has been used to implement CHROMA [Armengol and Plaza, 1994] [Armengol, 1997], a system for recommending a plan for the purification of proteins from tissues and cultures using chromatographic techniques. Purification is an essential process in the analysis of the properties of molecules from biological origin and widely used in industry and research. Proteins are a type of biological macromolecules that are purified by a sequence of laboratory operations. The most used operations for protein purification are chromatographic techniques. Purification plans of CHROMA incorporate different chromatographic techniques such as:

- *Ion-exchange*: process based on the Coulomb's law,
- *Hydrophobic Interaction*: process based on the Van der Vaals law,
- *Gel Filtration*: process that separates the molecules according its size, and
- *Affinity*: process that exploits the existence of specific unions between certain types of molecules.

Moreover, CHROMA incorporates different techniques such as *precipitation* and *clarification*, previous to the chromatographic process.

A plan to purify a molecule can be composed by several steps involving a chromatographic technique, that can be different, in each step. There is no unique way to purify a given protein. To choose an adequate purification plan involves reasoning about different aspects such as, for instance, the protein to purify, the sample origin (culture, tissue, etc), and the future use of the purified molecule. Protein purification requires the experience of an expert. Usually, a human expert first carries out a focused search in the literature in order to obtain a set of purification plans used in "similar" problems. Then, the expert analyzes the set of collected precedents and chooses the most appropriate.

The goal design of CHROMA was to build a system that, using a memory of purificated cases, was capable to find precedent cases useful for solving new experiments in an expert-comparable way. CHROMA learns from experience using two learning methods: CBR learning and induction. Four problem solving methods have been developed for recommending purification plans. One of them is a classification method that uses the induced knowledge. Moreover, a metalevel method is able to decide, for a particular problem, which problem solving method is more likely to succeed. The reflective capabilities of Noos allow CHROMA to analyze and decompose problem solving and learning methods in a uniform way, and also to combine them in a simple and efficient way.

We will give here a brief description of CHROMA's components. The reader may consult [Armengol and Plaza, 1994] and [Armengol, 1997] for a more detailed description of CHROMA.

Figure 6.1. A Noos browser of an experiment from CHROMA's case-base. An experiment is composed of two features: the sample from which the protein has to be extracted and the purification plan. In this experiment purification plan is formed by three steps `precipitation`, `ion-exchange`, and `affinity`.

6.1.1 Modeling domain knowledge

The domain ontology of CHROMA is composed of concepts such as experiments, samples, purification plans, proteins, species, tissues, and chromatographic techniques.

Experiments have two features: the `description` feature, embodying a description of the protein to be purificated and of the sample from which the protein has to be extracted, and the `purification` feature, embodying the purification plan to be performed (see Figure 6.1). A `description` is a feature term with two features: the `protein` to purify and the `sample`. A `sample` is described, in turn, by two features: the `species` where the sample comes from, and the `source` of the sample (such as an animal or vegetal tissue, or a culture).

Purification plans are composed of a variable number of chromatographic steps represented as features of the plan (called `step1`, `step2`, etc) embodying the step. Each step has two main features: the `name` feature, containing a specific chromatographic technique (`Affinity`, `ion-exchange`, etc), and either the `reagent` or the `resin` feature containing the substance used to purify the protein. Other complementary features such as the `PH` feature are also defined.

CHROMA has available an episodic memory of about one hundred solved purifications. Moreover, the system has been tested using twenty-five new problems.

6.1.2 Solving the purification task

The main task of the CHROMA application is the *purification* task. Given a new experiment containing only the sample, the goal of the *purification* task is to determine a purification plan using domain knowledge and the episodic memory of purificated cases. The purification task uses four methods for recommending purification plans: `equal-sample`, `analogy-by-determination`, `classify-by-prototype`, and `default-plan`

Equal-Sample method

The `equal-sample` method detects if there is an experiment in the episodic memory having the same sample as our current experiment.

Specifically, the `equal-sample` method is a CBR method decomposed in two subtasks: `retrieve` and `reuse` subtasks.

The `retrieve` subtask is achieved by a method defined by refinement of the `retrieve-by-pattern` built-in method. The retrieval method takes the sample of the current experiment problem as a pattern to perform a search for precedents into the episodic memory. The `reuse` task is achieved by reinstantiating the purification plan given in the precedent found to the current problem.

The `equal-sample` method is useful to solve routine purifications with commonly occurring samples and proteins assuring a correct solution for these cases.

Analogy-by-determination method

The `analogy-by-determination` method is a case-based method based on analogical *determinations* [Russell, 1990]. Determinations are functional dependencies used as justifications in analogical reasoning (see an example in Section 4.2).

The `analogy-by-determination` method is a method decomposed in three subtasks: `retrieve`, `select`, and `reuse` subtasks.

The `retrieve` task is achieved by a retrieval method that searches for experiments from the episodic memory purifying the same protein than the current experiment problem—that is to say, stating that the purification plan depends on the protein.

When the retrieval method finds several precedents, the method defined for the `select` task ranks the precedents according to domain specific criteria such as similarity of species or source. When domain criteria are not sufficient to determine a most preferred precedent, the precedents are presented to the user who must choose one of them.

Finally, the `reuse` subtask is achieved by reinstantiating the purification plan given in the most preferred precedent, according to the inference performed by the `select` subtask, to the current problem.

Classify-by-prototype method

The `classify-by-prototype` method is a classification method. This method is decomposed in four subtasks: `obtain-classes`, `plausible-classes`, `select`, and `reuse`.

The `obtain-classes` task is achieved by a method that obtains the set of solution classes in which a new experiment can be classified. This set of solution classes is generated using inductive learning by the `induce-prototype` method (see next method). Solution classes are defined as feature terms embodying a purification plan and a prototype that is a partial description of an experiment description generalizing the set of experiments that has been solved using this purification plan.

The `plausible-classes` task selects, using a method based on subsumption, the subset of solution classes which description subsumes the new experiment.

When more than one solution class is chosen, the `select` task has defined an user-interface method that presents their associated purification plans to the user and asks to the user to select the best of them (the user required to be himself the responsible instead of automating further the process).

As in all CHROMA methods, the `reuse` subtask is achieved by reinstantiating the purification plan given in the most preferred precedent, according to the inference performed by the `select` subtask, to the current problem.

Induce-prototype method

The `induce-prototype` method is an inductive method based on antiunification (see Section 4.6). The goal of the `induce-prototype` method is to construct a set of purification prototypes of experiments that share the same purification plan. These set of prototypes are used by the `classify-by-prototype` method.

The `induce-prototype` method is decomposed in three subtasks: `build-partitions`, `select-representative-sets`, and `generate-prototype`.

The first subtask divides the base of cases into sets containing experiments purified following the same purification plan.

Because of some of the formed sets may have few elements and it is not desirable to make induction with these small sets, the `select-representative-sets` task, defined by a filter method, rejects those sets having a number of elements lower than a threshold.

Finally, the `generate-prototype` subtask constructs, using methods based on antiunification such as INDIE (a bottom-up induction method) or DISC (a top-down induction method) [Armengol, 1997], a purification prototype for each representative set.

Default-plan method

The `default-plan` method is a domain method based on a statistical analysis of purification experiments. This method is used when there is no experiment in the episodic memory purifying the protein we are interested in and other methods are failed.

The `default-plan` method is based on the observation that when an expert doesn't find similar precedents in the literature for solving a given problem, the expert uses generally valid default plans. These default plans are less efficient but nonetheless appropriate for purification. The `default-plan` method captures this observation recommending a plan obtained from a statistical analysis of purification experiments.

Configuration of methods

CHROMA is provided by a metalevel reasoning method to decide, on a case-by-case basis, the applicability of the CHROMA methods and the order in which they are attempted. The metalevel method analyzes the current problem in order to determine the subset of applicable methods and an order for attempting methods. An empirical assessment has shown that this approach is better for this domain than the usual fixed sequence of methods attempted until one succeeds.

The implementation of the CHROMA metalevel reasoning method illustrates the powerful capabilities of Noos to implement, combine and experimentally evaluate the quality and efficiency of methods and method-combinations, and tailor the system to the task requirements in an application domain.

6.2 SPIN

SPIN [Armengol, 1997] is another system developed using Noos at the IIIA. SPIN is a sponge identification system for a class of marine sponge species (the family of *Geodiidae*). SPIN currently integrates a bottom-up induction method, a top-down induction method, a CBR method based on an entropy measure, and a method that combines lazy induction and CBR.

The identification of marine sponge specimens is a specially complex task that requires an expert due to the genetic diversity and the morphological plasticity of marine sponges. Moreover, in some sponge phylums, such as the Porifera phylum, is not clear how the different taxa are characterized.

SPIN knowledge base has been constructed from a subset of the specimens used to test the SPONGIA system [Domingo, 1995], a knowledge based system implemented at our Institute using MILORD-II [Puyol-Gruart, 1995]. An important remark is that a specific sponge specimen is described in SPIN using only the set of features used by SPONGIA in its identification.

A specimen can be identified as belonging to five different taxonomic levels (*class*, *order*, *family*, *genus*, and *species*). Currently, SPIN knowledge base contains only specimens from the *Geodiidae* family. This implies that SPIN can only identify specimens at *genus* and *species* levels. Nevertheless, incorporating specimens from other taxa, SPIN could perform identification at the five taxonomic levels.

Analogously to the description of CHROMA, we will give here a brief description of SPIN components. The reader can consult [Armengol, 1997] for a more detailed description.

Figure 6.2. A Noos browser of a sponge problem from SPIN's case-base. A sponge problem is composed of two features: the `description` of a sponge specimen and its classification into the five taxonomic levels.

6.2.1 Modeling domain knowledge

The domain ontology of SPIN is composed of concepts such as specimen descriptions, skeleton characteristics, and taxa.

Cases are described by refinement of a `sponge-problem`. A `sponge-problem` is a feature term holding two features: `description` and `solution` features (see Figure 6.2). The feature value of a `description` feature is a feature term describing the specific information of a sponge specimen. The feature value of a `solution` feature embodies a feature term describing the classification of the sponge specimen into the five taxonomic levels.

6.2.2 Solving the identification task

The main task of the SPIN application is the *identification* task. Given a new sponge specimen, the goal of the *identification* task is to determine the taxa (genus and species) to which this new sponge specimen belongs[1]. There are two alternative ways to proceed in order to identify specimens: (1) analyzing whether a new specimen has the features characteristic of some taxa, or (2) looking for similar specimens into the episodic memory and classifying the new specimen according to the taxa of the most similar precedents.

[1]They are all of the Geodiidae family, so there is no need of identification at family or higher levels

The identification task can be achieved using three different methods: `identify-by-subsumption`, `CRASS`, and `LID`. Moreover, the `identify-by-subsumption` method may use domain knowledge built by two inductive learning methods: either by INDIE or DISC.

Identify-by-subsumption method

The `identify-by-subsumption` method is a classification method based on domain knowledge acquired by means of an inductive learning method. The goal of `identify-by-subsumption` method is to classify a new sponge specimen from a set of concept descriptions of the taxa in which the new specimen can be classified. Concept descriptions are acquired using either INDIE or DISC. The decision of which inductive method to use is made dynamically by the user.

The `identify-by-subsumption` method is a method decomposed in four subtasks in a similar way of `classify-by-prototype` method developed in CHROMA.

CRASS

CRASS is a domain independent case-based method that uses an entropy-based assessment of similitude importance [Plaza et al., 1996b]. The goal of CRASS is to classify a new case, from a set of solution classes and a set of precedent cases, estimating its similarity to precedent cases.

The CRASS method is decomposed in two subtasks: `build-similarity--terms` and `select` tasks.

The goal of the `build-similarity-terms` terms subtask is to build a set of *similarity terms* from the new case N and each precedent case P_i. A similarity term is a partial description, built using antiunification, containing the commonalities between N and P_i.

Once the set of similitude terms are generated, the goal of the `select` case is to choose the precedent most similar to the new case using similitude terms. Specifically, the `select` subtask is performed by a method decomposed, in turn, in two subtasks: `entropy-assessment` and `weight` subtasks. The first `entropy-assessment` subtask estimates the importance of a similitude term S by measuring the entropy of the set of precedents subsuming S with respect to solution classes. This estimation is performed by a method based on Shannon entropy. The second `weight` subtask uses an aggregation-based method for weighting the entropy measure of each solution class with respect to the cardinality of the set of precedents belonging to this class. The solution class selected is that of with higher weight.

LID

LID is a domain independent method that combines top-down lazy induction and case-based reasoning. The goal of LID is also to classify a new case, from a set of solution classes and a set of precedent cases.

The LID method is decomposed in two subtasks: `build-description` and `identify` tasks.

The goal of the first `build-description` task is to construct a partial description of the new problem. The partial description is constructed incrementally, from an empty description, by adding one feature (the most discriminating one) to the description. The task is performed by a method that combines antiunification and a discriminant measure of features' relevance based on López de Mántaras distance [López de Mántaras, 1991].

The goal of the `identify` task is to discriminate the new problem, using the partial description constructed by the `build-description` task, with respect to precedent cases. The method developed to perform such task is based on subsumption and on Shannon entropy.

`Build-description` and `identify` subtasks are reiterated until the partial description constructed by the `build-description` task discriminates the new problem with respect to all precedent cases and it classifies it in a solution class, or there is no more discriminant features. In the first option, LID yields that solution class. In the second option, LID yields a set of ranked possible solution classes.

6.3 SHAM

SHAM [Cabré, 1996] is also a system developed using Noos. The main goal of SHAM was to develop a tool to help a non expert musician to harmonize melodies using background musical knowledge.

Musical knowledge is described by means of (1) PSMs that characterize local situations (for instance notes with specific weight in a metre), (2) PSMs that propose chords for these local situations, and (3) PSMs that combine the alternatives constructing a complete chord sequence.

Currently, SHAM takes a MIDI file[2] containing a melody line, translates the information to Noos descriptions, and starts an inference cycle of Noos obtaining a chord sequence and a bass line described in Noos. Finally, SHAM generates a new MIDI file containing the melody line plus the chord sequence and the bass line[3].

The user is not required to know neither Noos nor the internal musical representation of the system. SHAM is provided with a window-based interface that allows to choose an input MIDI file, specify musical information not provided in the MIDI file such as the key and the metre, and choose a set of alternative parameters stating the harmonization style. These set of harmonization parameters determine the set of used chords (e.g. not taking into account sevenths in chords and the maximum number of chord inversions) and the degree of complexity of the final chord sequence.

[2] MIDI files are the standard format for representing musical scores and can be generated by all computer music editors.

[3] The reader can visit our web site at <http://www.iiia.csic.es/Projects/music/> for sound examples.

Figure 6.3. A `Noos` browser of the song 'El noi de la mare' from SHAM. A problem to be solved in SHAM is described as a work with two features: `parts` describing the hierarchical decomposition of a musical piece in terms of bars and notes, and `harm-type` describing the user preferences for harmonization.

An interesting feature of SHAM is that different runs may result in different harmonizations depending on that several parameters and on a random component giving, therefore, the possibility to explore and combine different results. SHAM is being applied to harmonize catalan folk songs and sometimes shows a good degree of creativity.

6.3.1 Modeling musical knowledge

The domain ontology of SHAM is composed of concepts analyzing musical pieces in a hierarchical way (see Figure 6.3): a piece is represented as a `work` decomposed in several `parts`. Each `part` is able to have a different key. Moreover, a `part` is decomposed in several `bars`. Finally, every `bar` holds a set of `notes` and a set of `chords`.

Specifically, a `work` is a feature term with five features: `parts`, holding the set of parts of the work; `harm-type`, holding the harmonization parameters chosen by the user; `selected-chords`, holding the subset of applicable chords following the hamonization style specified in `harm-type`; `chord-sequence`, holding the final chord sequence inferred by SHAM; and `bass-line`, holding the final base line inferred by SHAM.

Feature values of features `parts` and `harm-type` are determined by using an input graphical interface. Feature `selected-chords` is defined using a filtering method that infers a subset of chords from the set of all defined chords and the set of user preferences defined in feature `harm-type`. Features `chord-sequence` and `bass-line` specify a possible harmonization of the piece and are inferred using problem solving methods defined for the harmonization task (see next subsection).

A `part` is a feature term with five features: `from-work`, that is a reference to the work it belongs; `order`, holding a reference number identifying the part; `key`, holding the key of the part; `mode`, holding the key mode (`major` or `minor`[4]); and `bars`, holding the set of bars of the part.

A `bar` is a feature term with two kinds of features: features given as problem data (such as `from-part`, `order`, `metre`, and `notes`) and features inferred, using background musical knowledge, while selecting the chord sequence (such as `vertical-choose`, `horizontal-choose`, `final-chords`, and `final-bass`). Feature `from-part` is a reference to the part it belongs. Feature `order` holds a reference number identifying the bar. Feature `metre` holds the metre of the bar. Feature `notes` holds the set of notes contained in the bar. Inferred features will be described in next section.

A `note` is a feature term with six features: `from-bar`, that is a reference to the bar it belongs; `pitch`, holding the pitch relative to the current key (e.g. pitch `C` in key `C` is represented as `p1`); `onset`, holding the delay from the start of the bar to the start of the note; `duration`, holding the note's duration; `octave`, holding the note's octave (only used for MIDI interface); and `chord-note?`, stating if the note is important enough to have a chord. Feature `chord-note?` is defined by means of a method that estimates the importance of the note from its `duration` and its `onset`.

A `chord` is a feature term with five features: `from-bar`, that is a reference to the bar it belongs; `name`, holding the kind of chord (e.g. `Imaj7`); `onset`, holding the delay from the start of the bar to the start of the chord; `duration`, holding the chord's duration; and `weight`, stating the appropriateness of the chord in the bar. Feature `weight` is defined by means of a method that estimates the suitability of the chord taking into account the notes of the bar it belongs. SHAM deals with more than one hundred different chords.

6.3.2 Solving the harmonization task

The main task of SHAM is the *harmonization* task. Given the melody of a musical piece, the goal of the *harmonization* task is to find a sequence of chords and a bass line for this piece. The chord sequence is built taking into account user

[4]currently SHAM supports only major modes since there are no harmonization methods dealing with minor modes. Nevertheless, adding new harmonization methods minor modes could be harmonizated.

preferences and background musical knowledge expressed as methods. The bass line is built using background musical knowledge and according to the chord sequence.

The selection of a chord sequence is more complex than the selection of an appropriate bass line. Here we only sketch the problem solving method developed by selecting chord sequences. The reader can consult [Cabré, 1996] for a detailed description of the harmonization process in SHAM.

The main reasoning in selecting chords is performed in the context of a bar. SHAM can choose one chord covering a whole bar, two chords covering each of them half a bar, a chord covering half a bar (the beginning or the end), or a rest (a bar without chords). The inference process engaged for each bar is decomposed in four subtasks: (1) first the set of chords available according to the user preference is taken from the work feature `selected-chords`, (2) then task `vertical-choose` is engaged in order to filter the subset of feasible chords according to the notes of the bar, (3) next task `horizontal-choose` estimates the appropriateness of chords according to neighbor bars, (4) finally task `final-chords` selects a chord (or a set of chords) for the bar taking into account the weight of chords and a random factor.

6.4 GYMEL

GYMEL [Sabater, 1997] is also a system developed in Noos for harmonization of melodies. The main goal of GYMEL was to develop a tool to help a non expert musician to harmonize melodies using a case-based reasoning approach. Moreover, GYMEL incorporates background musical knowledge for solving isolated situations not covered by existing cases.

The approach taken in GYMEL is quite different to SHAM's approach. SHAM is based on a hierarchical decomposition of a piece, given as input, from which the selection of possible chords is performed. On the other side, GYMEL starts from a melody line (a sequence of notes) constructing an analysis structure over that melody line that is the basis for retrieval of similar precedents.

GYMEL, as SHAM does, takes a MIDI file containing a melody line, translates the information to Noos descriptions, starts an inference cycle of Noos obtaining a chord sequence described in Noos, and finally generates a new MIDI file containing the melody line plus the chord sequence.

GYMEL suggests different harmonizations for a given musical phrase. The user can select some of them to be incorporated into the episodic memory. These harmonizations will then be used in solving new phrases.

6.4.1 Modeling musical knowledge

The domain ontology of GYMEL is composed of concepts such as phrases, keys, notes, chords, and nodes.

A musical piece is described as a `phrase`. A `phrase` is a feature term with six features: `key`, holding the key of the piece; `mode`, holding the key mode

Figure 6.4. A Noos browser of a musical phrase from GYMEL's case-base. A Case is represented as a feature term with six features: `key`, `mode`, and `metre` describe general knowledge of the phrase; `melody` holds a sequence of notes; `structure` holds a sequence of nodes; and `harmony` holds a sequence of chords.

(`major` or `minor`); `metre`, holding the metre of the piece; `melody`, holding the melody line as a sequence of notes; `structure`, holding the analysis structure of the piece as a sequence of nodes; and `harmony`, holding a sequence of chords. Figure 6.4 shows a browser of a phrase structure.

A `note` is a feature term with six features: `value`, holding the corresponding MIDI number of the note (used by the MIDI interface); `name`, holding the name of the note (e.g. `C4`); `pitch`, holding the pitch relative to the current key (as same as SHAM); `dur-rel`, holding the time distance from the beginning of the phrase to the note; `duration`, holding the note's duration; and `next`, holding a reference to the following note in the phrase.

A `chord` is a feature term with four features: `kind`, holding the kind of the chord (e.g. `IIminor7`); `dur-rel`, holding the time distance from the beginning of the phrase to the chord; `duration`, holding the chord's duration; and `next`, holding a reference to the following chord in the phrase.

The analysis structure is composed of a sequence of `nodes` grouping sets of consecutive notes in the phrase. Each note will belong to a unique node, and for each node one chord will be selected. A `node` is a feature term with six features: `notes`, holding a group of consecutive notes; `main-note`, holding the most important note of the group; `metre`, holding a reference to the metre of the phrase; `chord`, holding the chord to be played at same time than the group of notes; `next`, holding a reference to the following node; and `prev`, holding a reference to the previous node.

6.4.2 Solving the harmonization task

The main task of GYMEL is the *harmonization* task. Given the melody of a musical phrase, the goal of the *harmonization* task is to find a sequence of chords for this phrase. The *harmonization* task is performed by a method decomposed in two subtasks: `build-analysis-structure` and `propose-harmony`.

The goal of task `build-analysis-structure` is to analyze the phrase and to construct a structure grouping subphrases candidates to share the same chord. This grouping structure is built using background musical knowledge. A structure is composed by a sequence of `nodes`. For each `node` a collection of notes forming a subphrase and a main note is selected.

After the construction of the analysis structure, the harmonization process starts properly. The `propose-harmony` task is performed by a case-based method that proposes a chord for each node of the analysis structure. Moreover, when the case-based method is not able to find precedents for a given node, GYMEL uses a method for proposing a chord based on background musical knowledge.

The case-based method is decomposed in two subtasks: `retrieve` and `reuse`.

The `retrieve` task is performed, in turn, by a method decomposed following the usual retrieval subtasks: `identify`, `search`, and `select` subtasks.

The `identify` task is performed by a method, called `build-node-perspective`, based on perspectives (see Section 4.3). The `build-node-perspective` method builds a retrieval pattern based on the analysis structure of the phrase. Given a node, the method builds a perspective of that node taking into account the main note of the node and the selected chords of its two previous nodes. Specifically, the `build-node-perspective` method is defined as follows:

```
(define (perspective build-node-perspective)
  (node )
  (pattern (define (node)
             (main-note (define (note)
                          (pitch (define (relative-pitch)))))
             (prev (define (node)
                     (chord (define (chord)
                              (kind (define (chord-kind)))))
                     (prev (define (node)
                             (chord (define (chord)
                                      (kind (define (chord-kind)))))))))))
```

For instance, the application of that method to a node with a main note with pitch P4 and with two previous nodes with selected chords `IImin7` and `Imaj`, yields the following pattern:

```
(define (node)
  (main-note (define (note)
               (pitch P4)))
  (prev (define (node)
          (chord (define (chord)
                   (kind IImin7)))
          (prev (define (node)
                  (chord (define (chord)
                           (kind Imaj))))))))
```

The `search` task is performed by a method based on the `retrieve-by-pattern` built-in method. This method retrieves, using the pattern constructed by perspectives, nodes sharing the same main note and the same chord progression of the two previous nodes than the node problem.

The `select` task performs a random selection. The Noos backtracking mechanism assures that all precedents will be attempted.

The goal of the `reuse` task is to access to the chord of the selected precedent node and propose a new chord, with same `kind` of the precedent chord, for the current node. The other features are determined from the features of the current node and background musical knowledge.

The GYMEL's problem solving method is exhaustive: all alternative solutions that it can be built are shown to the user. Then, the user chooses some of them to be stored into the episodic memory, and thus to be used in solving future problems.

6.5 Saxex

Saxex [Arcos, 1996] [Arcos et al., 1997b] is a case-based reasoning system for generating expressive performances of melodies based on examples of human performances that are represented as structured cases. *Saxex* has been developed using sound analysis and synthesis techniques of the SMS environment (Spectral Modeling Synthesis) [Serra, 1997] and the Noos language.

An input for *Saxex* is a musical phrase described by means of a musical score (a MIDI file) and a sound file. The score contains the melodic and the harmonic information of the musical phrase. The sound file contains the recording of an inexpressive interpretation of the musical phrase played by a musician. The output of the system is a new sound file, obtained by transformations of the original sound file, containing an expressive performance of the same phrase.

Solving a problem in *Saxex* involves three phases: the *analysis* phase, the *reasoning* phase, and the *synthesis* phase (see Figure 6.5).

Analysis and synthesis phases are implemented using SMS sound analysis and synthesis techniques, based on spectrum models, that are useful for the extraction of high level parameters from real sounds, their transformation and the synthesis of a modified version of the original sound. *Saxex* uses SMS in order

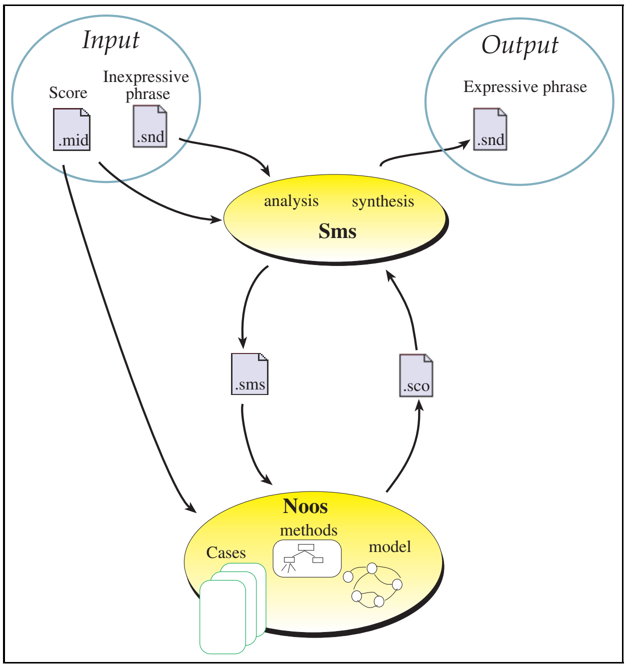


Figure 6.5. General view of Saxex components. Analysis and synthesis phases are performed in SMS. Reasoning phase is performed in Noos.

to extract basic information related to several expressiveness parameters such as dynamics, rubato, vibrato, and articulation. The SMS synthesis procedure allows *Saxex* the generation of new expressive interpretations (new sound files).

The reasoning phase is performed using case-based techniques and is implemented in Noos. This phase of *Saxex* incorporates background musical knowledge based on Narmour's Implication/Realization model [Narmour, 1990] and Lerdahl and Jackendoff's Generative Theory of Tonal Music (GTTM) [Lerdahl and Jackendoff, 1993]. These theories of musical perception and musical understanding are the basis of the computational model of musical knowledge of the system: using Noos perspectives methods with background musical knowledge, *Saxex* takes dynamical decisions about the relevant aspects of a given problem. That is to say, the background musical knowledge is used by *Saxex* as a set of declarative biases for retrieval.

Problems to be solved by *Saxex* are represented as complex structured cases embodying knowledge about the score of the phrase, knowledge about musical understanding of the phrase, and knowledge about the expressive performance of the phrase.

Figure 6.6. A Noos browser of the score for the 'All of me' ballad. Features are represented as thin boxes, dots indicate not expanded terms, and gray boxes express references to existing terms.

6.5.1 Modeling musical knowledge

The domain ontology of *Saxex* is composed of concepts representing three different types of musical knowledge: (1) concepts related to the score of the phrase such as notes and chords, (2) concepts related to background musical theories such as implication/realization structures and GTTM's time-span reduction nodes, and (3) concepts related to the performance of musical phrases. A *Saxex* case is represented as a feature term with three features: the `score`, the `structure`, and the `performance`.

The score

A score (see Figure 6.6) is represented embodying a musical `phrase`. A `phrase` is a feature term with two features: the `melody` feature, embodying a sequence of notes, and the `harmony` feature, embodying a sequence of chords (see Figure 6.6). Each `note` holds in turn a set of features such as the `pitch` of the note (`C5`, `G4`, etc), its `position` with respect to the beginning of the phrase, its `duration` (using CommonMusic notation [Taube, 1991][Taube, 1996]), a reference to its `underlying-harmony`, and a reference to the `next` note of the phrase. Moreover, a note holds the `metrical-strength` feature, inferred using GTTM theory, expressing the note's relative metrical importance into the phrase. Chords have also a set of features such as the `name` of the chord (`Cmaj7`, `E7`, etc), their `position`, their `duration`, and a reference to the `next` chord.

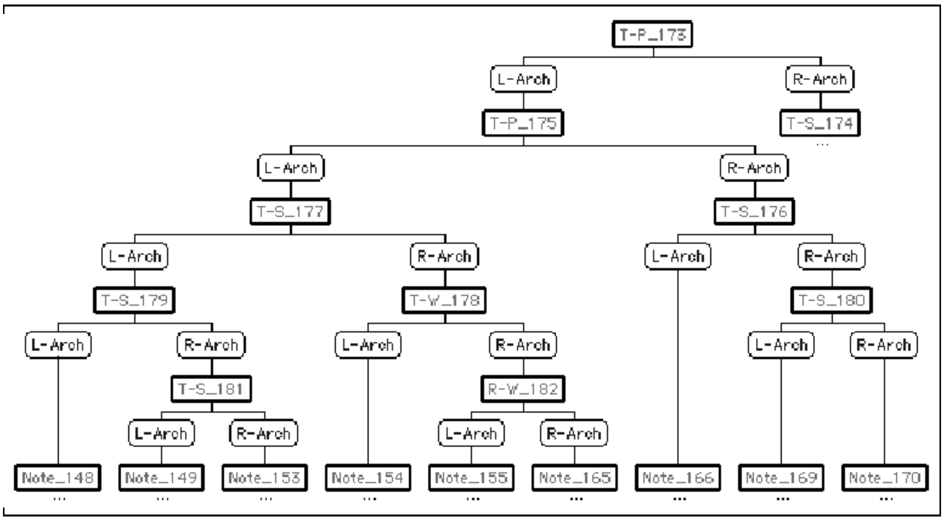


Figure 6.7. A Noos browser of the prolongational reduction structure for the 'All of me' ballad.

The musical structure

The musical structure embodies the musical analysis of the phrase built using the background musical knowledge. Narmour's implication/realization model (IR) proposes a theory of cognition of melodies based on eight basic structures. These structures characterize patterns of melodic implications that constitute the basic units of the listener's perception. Other parameters such as metric, duration, and rhythmic patterns emphasize or inhibit the perception of these melodic implications. The use of the IR model provides a musical analysis based on the structure of the melodic surface.

On the other hand, Lerdahl and Jackendoff's generative theory of tonal music (GTTM) offers an alternative approach to understanding melodies based on a hierarchical structure of musical cognition. GTTM proposes four types of hierarchical structures associated with a piece. This structural approach provides the system with a complementary view for determining relevant aspects of melodies.

The musical analysis builds a set of structures over the musical phrase. It is represented by the `analysis` feature term with three features: `prolongational-reduction`, `time-span-reduction`, and `process-structure`. The `prolongational-reduction` feature embodies a hierarchical structure describing tension-relaxation relationships among groups of notes. Tension-relaxation relationships are represented in Noos as trees (see Figure 6.7). The `time-span-reduction` feature embodies another hierarchical structure that describes the relative structural importance of notes within the heard rhythmic units of a phrase. These structural relationships are also represented in Noos

Figure 6.8. A Noos browser of the musical performance structure for the 'All of me' ballad.

as trees. Finally, feature `process-structure` embodies a sequence of implication/reduction (IR) Narmour's structures. There are eight types of IR structures. Each IR structure has a set of features representing the different roles that can play the notes in the structure (such as `first-note`, `med-notes`, and `last-note`) and characteristics specific of each IR structure such as the melodic direction.

The musical performance

A musical performance is represented as a sequence of events (see Figure 6.8). There is an `event` for each note within the phrase embodying knowledge about expressive parameters applied to that note. Specifically, an `event` has features representing expressive parameters of notes such as `dynamics`, `rubato`, `vibrato` level, `articulation`, and `attack`. Expressive parameters are described using qualitative labels as follows:

Changes on dynamics are described relative to the average loudness of the phrase by means of a set of five ordered labels. The middle label represents average loudness and lower and upper labels represent respectively, increasing or decreasing degrees of loudness.

Changes on rubato are described relative to the average tempo, also by means of a set of five ordered labels. Analogously to dynamics, qualitative labels about

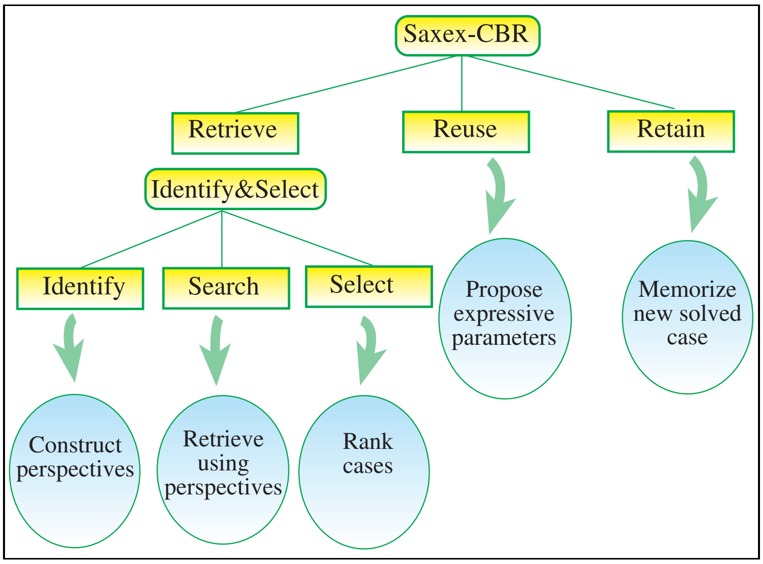


Figure 6.9. Task decomposition of the *Saxex* CBR method.

rubato cover the range from a strong accelerando to a strong ritardando.

The vibrato level is described using two parameters: the frequency vibrato level and the amplitude vibrato level. Both parameters are described using five qualitative labels from `no-vibrato` to `highest-vibrato`.

The articulation between notes is described using again a set of five ordered labels covering the range from `legato` to `staccato`.

Finally, *Saxex* distinguishes two transformations over a note attack: (1) reaching the pitch of a note starting from a lower pitch, and (2) increasing the noise component of the sound. These two transformations were chosen because they are characteristic of saxophone playing but other transformations can be introduced without altering the system.

6.5.2 The Saxex task

Given a musical phrase, *Saxex* infers a specific set of expressive transformations to be applied to every note in the phrase. These sets of transformations are inferred note by note. For each note in the phrase the same problem solving method is performed.

The problem solving method developed in *Saxex* for this purpose follows the usual subtask decomposition of CBR methods described in Section 4.5: *retrieve, reuse, and retain* (see Figure 6.9). Given a current note problem of a problem phrase, the overall picture of the subtask decomposition of *Saxex* method is the following:

- *Retrieve*: The goal of the retrieve task is to choose the set of notes (cases) most similar to the current note problem. This task is decomposed in three subtasks:

- *Identify*: The goal of this task is to build retrieval perspectives using two alternative biases. A first bias based on Narmour's implication/realization model, and a second bias based on Lerdahl and Jackendoff's generative theory.
- *Search*: The goal of this second task is to search cases in the case memory using Noos retrieval methods and previously constructed perspectives.
- *Select*: The goal of the select task is to rank the retrieved cases using Noos preference methods. The preference methods use criteria such as similarity in duration of notes, harmonic stability, or melodic directions.

• *Reuse*: the goal of the reuse task is to choose a set of expressive transformations to be applied to the current problem from the set of more similar cases. The first criterion used is to adapt the transformations of the most similar case. When several cases are considered equally similar, transformations are selected according to the majority rule. Finally, when previous criteria are not sufficient, all the cases are considered equally possible alternatives and one of them is selected randomly.

• *Retain*: the incorporation of the new solved problem to the memory of cases is performed automatically in Noos. All solved problems will be available for the reasoning process in future problems.

After describing the task decomposition of *Saxex* problem solving method, we will introduce a simplified example to help its understanding. Let us suppose that *Saxex* has to infer a set of expressive transformations for the following note within a phrase[5]:

```
(define (note Note1)
  (pitch A4)
  (position 17)
  (duration Q.)
  (metrical-strength extremely-high)
  (belongs-to (define (P)
                (first-note (>>))
                (med-notes Note2)
                (last-note Note3)
                (direction down)))
  (next Note2))
```

The first task engaged is the `retrieve` task. The `retrieve` task engages in turn the `identify` subtask. Taking as example the following bias based on Narmour's model:

[5]The right side presents a picture of the note in musical notation.

```
(define (note P1)
  (pitch C5)
  (position 0)
  (duration Q)
  (metrical-strength extremely-high)
  (belongs-to (define (Process)
                (first-note (>>))
                (med-notes Note102)
                (last-note Note103)
                (direction down)))
  (next Note102))
```


```
(define (note P2)
  (pitch E4)
  (position 25)
  (duration Q)
  (metrical-strength extremely-high)
  (belongs-to (define (Process)
                (first-note (>>))
                (med-notes Note21)
                (last-note Note22)
                (direction up)))
  (next Note21))
```

Figure 6.10. Two precedent cases retrieved by *Saxex* Problem solving method.

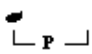
Determine as relevant the role that a given note plays in a implication/realization structure.

described as a Noos description as follows:

```
(define (note)
  (belongs-to (define (N-structure)
                ($f (>>)))))
```

We obtain the following constructed perspective of `Note1`:

```
(define (note)
  (belongs-to (define (P)
                (first-note (>>)))))
```



that is, the first note of a `P` process.

Then, the *search* task is engaged in order to find similar situations among the precedent cases. Let us assume that the *search* task finds the following two notes (called `P1` and `P2`) as precedent cases (see Figure 6.10).

Next, the `select` task is engaged for ranking the precedents. Taking as preference criterion the melodic direction, precedent `P1` is considered as the most relevant precedent (since it belongs to a process with descending `direction` like `Note1`).

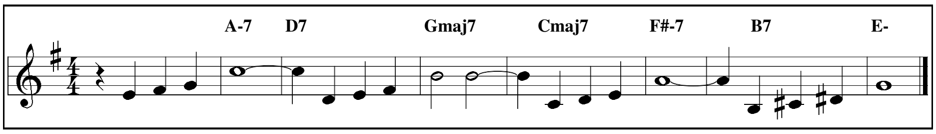


Figure 6.11. First phrase from the 'Autumn Leaves' theme.

After choosing precedent `P1` as the most relevant precedent, the `reuse` task is engaged. For this simplified example, since we only have selected one precedent, the set of expressive transformations to be applied to `Note1` are the same were applied to precedent `P1` and that are stored as part of precedent case `P1`.

6.5.3 Experiments

We have studied the issue of musical expression in the context of tenor saxophone interpretations. We have done several recordings of a tenor sax performer playing several Jazz standard ballads ("All of me", "Autumn leaves", "Misty", and "My one and only love") with different degrees of expressiveness, including an (almost) inexpressive interpretation of each piece. These recordings are analyzed, using the SMS spectral modeling techniques, in order to extract basic information determining the expressive parameters. The set of extracted parameters together with the scores of the pieces constitute the set of structured cases of the case-based system. From this set of cases and using similarity criteria based on background musical knowledge, the system infers a set of possible expressive transformations for a given piece. Finally, using the set of inferred transformations and the SMS synthesis procedure, *Saxex* generates a new sound file containing expressive performances of the jazz ballads.

We have performed two sets of experiments combining the different Jazz ballads recordered. The first set of experiments consisted in using examples of expressive performances of some phrases of a piece in order to generate new expressive performances of another phrase of the same piece. More concretely, we have worked with three different expressive performances of two phrases of a piece, having about twenty notes, in order to generate new expressive performances of another phrase of the same piece. This group of experiments has revealed that *Saxex* identifies clearly the relevant cases even though the new phrase introduces small variations with respect to the phrases existing in the memory of cases.

The second set of experiments consisted in using examples of expressive performances of some pieces in order to generate expressive performances of other pieces. More concretely, we have worked with three different expressive performances of pieces having about fifty notes in order to generate expressive performances of new twenty note phrases. This second group of experiments has revealed that the use of perspectives allows to identify situations such as long notes, ascending or descending melodic lines, etc. Such situations are also usually identified by a human performer.

Let as describe briefly some of the expressive transformations applied to the first phrase of the 'Autumn Leaves' theme (see the score in Figure 6.11) based on precedent cases of similar phrases. Concerning to changes of dynamics, the ascending melodic progressions are performed using crescendo. For instance, the first note of the theme (E) starts piano and the dynamics is successively increased yielding a forte in the fourth note (C). Concerning rubato, after the fourth note (C) the attack of the fifth note (D) is delayed and brought closer to the next note, then the duration of sixth note (E) is expanded, and finally the duration of the next note (F) is reduced. Vibrato is applied over notes with long duration combined with a dynamics decay (for instance, over fourth note). In ascending melodic progressions, articulation is also transformed by decreasing the interruption between notes (i.e. playing closer to legato than to staccato). Finally, the transformation of the attack consisted in reaching the eighth and ninth notes (B and B) starting from a lower pitch [6].

6.6 NoosWeb

The goal of the development of NoosWeb [Martín, 1996] was to provide a WWW user interface to Noos applications supporting the same interaction capabilities provided in the Noos window-based graphical user interface developed for MacOS computers (see Appendix A).

In order to provide full access to the Noos facilities, NoosWeb was designed to support, at least, (i) a Noos listener and (ii) a graph browsing facility. The Noos listener permits the evaluation of any valid Noos expression. The graph browsing facility emulates Noos browsers generating HTML documents.

The WWW interface to Noos is a sequence of dynamically-generated HTML documents that include forms (for engaging actions) and tables (for graph browsing). The user sees a standard NoosWeb interface document displaying a set of forms implementing valid actions (such as browsing and evaluating an expression from the listener) and displaying the answers of the last action performed. Although the user can view past actions using the NoosWeb client-cached documents, in order to avoid time-traveling problems, the user can only perform a new action from the current (the last) document. Actions requested from cached documents are detected and dismissed. Nevertheless, NoosWeb incorporates in documents the list of browsers displayed in the session allowing to redisplay any of them at any time.

NoosWeb is accessible at <http://www.iiia.csic.es/Interficies/NoosWeb>.

6.6.1 The NoosWeb architecture

The NoosWeb architecture is composed of two elements: a collection of clients connected through the network to a server (see Figure 6.13). The server side

[6]The reader can visit our web site at <http://www.iiia.csic.es/Projects/music/Saxex> for sound examples.

Figure 6.12. A NoosWeb browser of a sponge-problem from the SPIN system.

Figure 6.13. The NoosWeb architecture (from [Martín, 1996]).

maintains the state for each user (client) involved in a session. The server checks each client request against the current state. The server connects one or several Noos applications. Each Noos application only supports one user session over the WWW. A standard HTTP daemon uses CGI to process requests by means of a script called *NoosWebCGI*. *NoosWebCGI* dispatches requests to several Noos applications by TCP/IP. Each Noos application keeps contact with a Common-Lisp program called *NoosWebCL* that receives the requests, asks Noos when necessary, and returns an answer generating an adequate HTML document.

NoosWebCGI

NoosWebCGI is a CGI script implemented in the C language. *NoosWebCGI* is a dispatcher that distributes incoming requests to the appropriate Noos application by means of BSD sockets. *NoosWebCGI* is generic since it forwards all the request contents to the Noos application without any pre-processing and it also returns answers without any post-processing. *NoosWebCGI* maintains information about active Noos applications and current client sessions. When a user initiates a session, *NoosWebCGI* allocates a session number for an user and a Noos application. For each client request the script checks that the session identifier is valid. A request is valid if no time limit has been exceeded. There are time limits for session (1 hour) and for under-using (5 minutes without any request from a client). User sessions declared invalid are deallocated.

NoosWebCL

NoosWebCL is implemented in Common Lisp and is composed of four modules: *PassiveTCP*, *Parser-URL-encode*, *NoosWeb-obj*, and *Noos-to-HTML* modules. The module *PassiveTCP* waits to receive requests by means of TCP from *NoosWebCGI*. Received requests are parsed by the *Parser-URL-encode* module that determines which functionality of Noos is invoked (browsing a feature term, asking a specific query expression, evaluating a Noos description, etc). The *Parser-URL-encode* module engages Noos using the set of remote Noos functions. The result of the requested action is translated and stored into the *NoosWeb-obj* module structures. Finally, the *Noos-to-HTML* module generates an adequate HTML document that is returned to *NoosWebCGI*.

Changes to Noos

Noos has not been changed for allowing WWW interface. Only two new functions has been developed in order to provide an interface to remote calls. The first function is the *remote-browse* function that returns a Noos browsing structure in a list syntax. The second function is the *remote-eval* function that checks it is a legal Noos expression (a description of a query expression) and evaluates the expression returning the result of the evaluation.

6.7 Summary

This chapter presented a set of diverse applications developed using Noos by several persons at the IIIA. The purpose was to provide examples of how applications can be developed using Noos. In this chapter our description has focused on three aspects of the applications: how the domain knowledge required in each application is modeled using Noos representation capabilities, which problem solving methods are developed, and which learning methods are incorporated. Specifically we presented,

- how diverse domain specific case-based reasoning techniques are developed in Noos. Case-based reasoning is modeled by domain specific methods from a knowledge modeling analysis of an application. These methods incorporate domain-knowledge into Noos retrieval built-in methods. Examples of applications incorporating case-based reasoning methods are CHROMA, SPIN, GYMEL, and *Saxex*.

- how preferences are used for ranking precedents in case-based reasoning modeling different domain specific criteria in applications such as CHROMA and *Saxex*. The development and combination of diverse domain specific preferences provide a mechanism for the assessment of complex structured cases.

- how diverse inductive learning methods such as INDIE, DISC, and LID are developed in Noos. These methods are based on a search in the space of feature terms. Different methods perform several search strategies using the subsumption ordering in the feature terms space.

- how several alternative problem solving methods can be defined for solving the same task. We described how CHROMA uses four different methods for recommending purification plans in the *purification* task, and how SPIN uses three different methods for achieving the *identification* task.

- how CHROMA is provided by a metalevel reasoning method to decide, on a case-by-case basis, the applicability of the problem solving methods defined for solving the purification task and the order in which they are attempted.

- how case-based reasoning and inductive learning have been integrated in CHROMA and SPIN applications. The knowledge modeling analysis of a specific application determine how different learning methods can be integrated in different subtasks. Then, metalevel reasoning capabilities and the episodic memory of Noos allows to effectively implement this integration.

- how to use domain knowledge intensively. For instance, SHAM makes an intensive use of background musical knowledge for harmonizing melodies. SHAM models musical knowledge by means of (1) PSMs that identify which notes are important in the melody, (2) PSMs that propose alternative sets of feasible chords according to these notes, and (3) PSMs that combine the alternatives constructing a complete chord sequence.

- how the structured representation of cases offers the capability of treating subparts of cases as full-fledged cases. For instance, in GYMEL and *Saxex* applications the solution for a new problem is built by combining and adapting subparts of solutions from several precedent cases.

- how two complex musical theories for musical perception and musical understanding are modeled in *Saxex* and are then used for analyzing musical phrases and assessing their similitude with respect to other phrases.

- how perspectives are used as a mechanism to describe declarative biases for case retrieval in structured representations of cases. For instance, perspectives provide to GYMEL and *Saxex* applications a flexible way to dynamically construct partial descriptions for retrieval.

Finally, we presented NoosWeb, a WWW interface to Noos applications supporting the same interaction capabilities provided in the Noos window-based graphical interface developed for MacOs computers. A set of small improvements has been performed in Noos to support remote calls.