Towards a Software Architecture for Case-based Reasoning Systems

Enric Plaza and Josep-Lluís Arcos

IIIA - Artificial Intelligence Research Institute
CSIC - Spanish Council for Scientific Research
Campus UAB, 08193 Bellaterra, Catalonia, Spain.
Vox: +34-93-5809570, Fax: +34-93-5809661
{enric,arcos}@iiia.csic.es
http://www.iiia.csic.es

Abstract. We present a software architecture model of adaptation in CBR. A software architecture is defined by its components and their connectors. We present a software architecture for CBR systems based on three components (a task description, a domain model, and adaptors) connected by a type of connectors called bridges. Adaptors are basic inference components that perform specific transformations to cases. Two kinds of adaptors are introduced: domain adaptors (*d-adaptors*) and case-based adaptors (*c-adaptors*). Adaptors are applied to a given problem, performing search until a sequence of adaptor instantiations is found such that a solution is achieved. Thus, in the ABC architecture adaptation is viewed as a search process on the space of adaptors. We describe how the ABC components have been used in the SaxEx application, a CBR system for generating expressive musical phrases.

1 Introduction

The goal of software architectures is learning from system developing experience in order to provide the abstract recurring patterns for improving further system development. As such, software architectures contribution is mainly methodological in providing a way to specify systems. In this paper we present a software architecture for adaptation in CBR—called Adaptors-Bridges-Connectors software architecture (ABC)—based on the notion of connectors and inspired on object-oriented and component-based methodologies.

The three main elements of the ABC software architecture are (i) a task description—characterizing the goal that a CBR system pursues; (ii) a domain model—characterizing the ontology and properties of the knowledge content; and (iii) a library of adaptors—performing transformations to case-specific models. These three elements are connected with a special kind of connectors called bridges.

Problem solving in ABC is considered as the construction of a *case-specific model*. ABC follows the "problem solving as modeling" view, that is to say, solving a problem consists of building a model specific to the problem that satisfies the task requirements. In this view, a knowledge system uses a domain model to enlarge the input model until a complete and correct case-specific model is built—where "complete and correct" are with respect to the requirements of the task.

We have considered two kinds of adaptors: domain adaptors (*d-adaptors*) and case-based adaptors (*c-adaptors*). *D-adaptors* use some domain-specific knowledge to transform the case-specific model (in a way specified by the adaptor's competence). *C-adaptors* also transform the case-specific model but use domain knowledge that includes precedent cases retrieved from case memory.

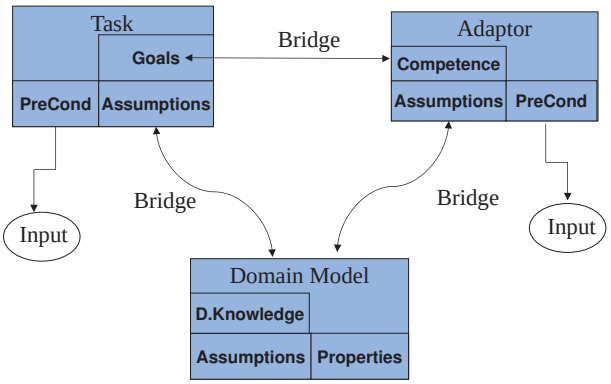

Fig. 1. The ABC software architecture consists of three elements: a task description, a domain model, and a library of adaptors. These three elements are connected by connectors called bridges. The problem to be solved is called input in this picture.

Adaptors are applied to the case-specific model, performing search until a sequence of adaptor instantiations is found such that transforms the initial case-specific model into a correct case-specific model that satisfies the task goals. Thus, adaptation is viewed as a search process on the space of adaptors. Since new adaptors can be applied to the first adapted object, several search strategies (such as depth-first, breadth-first, and beam search) are possible.

We will show how the ABC theory has been applied in the SaxEx application, a complex real-world case-based reasoning system for generating expressive performances of melodies based on examples of human performances that are represented as structured cases.

In general terms, a *software architecture* describes the (i) components, (ii) connectors, and (iii) a configuration of how the components should be connected [9]. We can consider CBR systems as a specific variant of knowledge systems that furthermore use experiential knowledge [5]. Because of this we have taken UPML, a software architecture being developed for reuse of knowledge systems, and we are developing a variant adequate for CBR systems. The Unified Problem-solving Method Development Language UPML is currently in development by the IBROW consortium, and the first version is currently released [8]. Although UPML can still have future modifications we expect them to be minor and maintain stable the core ideas.

The organization of this paper is as follows. In Section 2 we present the ABC architecture. Section 3 describes how two different families of adaptors (c-adaptors and d-adaptors) are incorporated in the Noos language. Section 4 shows the use of c-adaptors and d-adaptors in the SaxEx application. Finally, in Section 5 we present the conclusions and discuss related work.

2 The ABC architecture

The three main elements of the ABC software architecture are (i) a task description, (ii) a domain model, and (iii) a library of adaptors. Figure 1 shows these three elements connected with a special kind of connector called bridge. In addition, the problem to be solved is called input in the figure and for simplicity we will include the case base into the domain model element. More specifically, we will consider that each solved problem is a model per se, and we will call it *case-specific model*—in other words, it is the model of an episode of solving that problem [5].

These three elements are taken from UPML where the main goal is the reuse of Problem Solving Methods (PSMs); since our goal is the *reuse of cases* we propose the specific architectural variation where adaptors play the role of PSMs. This transformation makes sense

since PSMs are the components that perform the inferences for a knowledge system to build the *case-specific model* of the problem (i.e. the "solution" to the problem). In our approach the final case-specific model is build by the adaptors that transform the *case-specific model* imported from the case(s) retrieved from the case-base.

Tasks, domain models, and adaptors are conceptually distinct entities, although in practice CBR systems use an implicit description of the task and domain knowledge is tightly integrated with the CBR engine. From a methodological stance, however, it is better to consider they separate and possibly coming from distinct sources.

In a similar manner, specification of tasks is also being studied [14] to provide a vocabulary capable of describing tasks across a range of different domains of application, i.e. independent of the domain-specific vocabulary. For instance, a task description of diagnosis [8] is specified in terms of findings and hypothesis. A *bridge* is then needed to connect in a meaningful way task descriptions and domain models. For instance, in a medical diagnosis domain the bridge from the task description to the domain model maps findings and hypothesis to the terms manifestation and cause respectively.

Therefore the methodological approach we are endorsing takes two main aspects of knowledge modeling techniques: explicit representation and conceptual separation of tasks, domain knowledge, and adaptors. From UPML software architecture [8] we adopt the bridge connectors among ABC architecture components but we change the main elements of the architecture for CBR systems. In the rest of this section we make explicit those components, while in later sections we show a particular adaptation engine developed following this methodology.

Before considering in detail the components of the ABC software architecture a point needs to be clarified. It may seem ABC deals only with the adaptation phase of CBR and not with the retrieval phase. In a certain way this is true, but we are not dismissing or ignoring case retrieval. The ABC software architecture focuses on *case reuse* and the rest of the CBR phases (retrieve, reuse, repair, and retain) is abstracted away, simply being considered part of the knowledge contained in the domain model. For instance, the SaxEx system shown in section § 4 has two main processes before adaptation: given an input case-specific model SaxEx uses domain knowledge (two musical theories) to analyze it and build a more complex case-specific model—however in the ABC architecture this process is just considered part of the domain model. Also, since ABC does not specify control then it is not reflected that the musical analysis being performed previously to retrieval and adaptation. Concerning retrieval of cases (and subparts of cases, also considered cases in SaxEx) we consider that the set of case-specific models (the case base) is part of the domain model and we also consider that the criteria for assessing similitude and relevance of these precedent cases for the current problem is knowledge contained in the domain model.

2.1 The ABC components

Tasks A task provides a way to characterize what a CBR system is intended to achieve.

Task Description consist of a *task name* and
1. pragmatics (author, explanation, URL, last change date)
2. ontology (the vocabulary)
3. specification
   - goals (expressions characterizing the output case-models)
   - preconditions (expressions characterizing valid input case-models)
   - assumptions (expressions characterizing requirements on domain knowledge)

The main elements for characterizing a task are goals, preconditions and assumptions. These elements are described in some logical language, the option of which is open to the designer. Preconditions state constraints to be satisfied by the problems to be solved (input case models). Goals specify properties to be satisfied by the solved problem, i.e. by the output

case-specific model. Finally assumptions determine assumptions made by the task description upon the content of the domain model.

Domain Models Domain models are specified using a specific vocabulary (domain ontology) and is characterized by properties, assumptions, and domain knowledge.

Domain Model Description consist of a *domain model name* and
1. pragmatics (author, explanation, URL, last change date)
2. ontology (the vocabulary)
3. specification
   - properties (meta-expressions characterizing domain knowledge)
   - domain knowledge (expressions describing knowledge)
   - assumptions (expressions characterizing assumptions on domain model)

Properties and assumptions both are used to characterize the knowledge content of a KB. These characteristics can be directly inferred from the domain knowledge or can be derived from requirements introduced by other components of the specification. While properties deal with characteristics of the knowledge content assumptions deal with external requirements like the environment of the system. As before, properties, assumptions, and domain knowledge are expressed in a specific formal language of choice.

Task-domain bridge The *td-bridge* is a connector that translates (refines) the task specification to a particular domain specified in domain-model. This bridge may add assumptions (on domain knowledge) to ensure that the translation result is valid. The only formal requirement is the union of both task and domain specifications is logically consistent.

Adaptors An adaptor is a special kind of connector between case-specific models—i.e. between "models of cases".

Adaptor Description consist of an *adaptor name* and
1. pragmatics (author, explanation, URL, last change date,
2. ontology (the vocabulary)
3. specification
   - preconditions (expressions characterizing valid input case-models)
   - assumptions (expressions characterizing domain knowledge needed by the adaptor to succeed)
   - competence (expressions characterizing the output case-models)

The preconditions of an adaptor specify the requirements to be satisfied by the input case-specific model for the adaptor's result be a valid one. The competence is a description of the transformation resulting from the application of the adaptor. Finally, the assumptions express the kind of domain knowledge the adaptor requires in order to be able to function. These assumptions may enlarge the requirements on domain knowledge already specified by the task. Since the case base is considered as a specific type of domain knowledge, case-based adaptation is considered to be realized by adaptors that use the experiential knowledge of the case base. Case-based adaptors are later discussed on § 3.

Task-Adaptor bridge The *ta-bridge* works like the tb-bridge above but now is connecting the task goals with the adaptors competence. Since the task goals specify the conditions for a problem to be correctly and completely solved the problem solving process is finished when an adaptor with a corresponding competence is available.

There are different ways to realize the ta-bridge depending on the strategy used to implement adaptors. A common strategy is designing a component library of adaptors. Moreover,

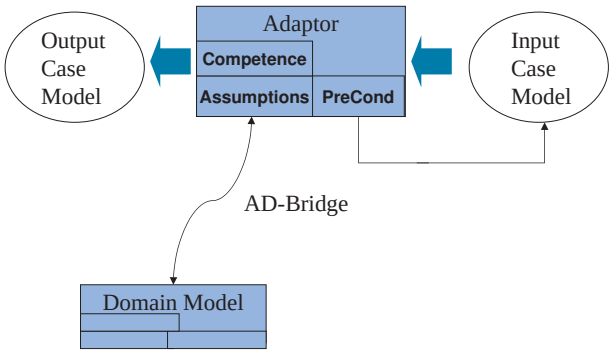

Fig. 2. The adaptor is a connector between "models of cases", here called case-specific models.

depending on the complexity of the application domain the designers may implement one-shot adaptors—i.e. adaptors with a competence that directly fulfills the task goals. In more complex situations, the "total" adaptor need to be constructed from the elementary components in the adaptor library to fit the needs of each particular problem. Section 4 below show that this is the implementation we have chosen for the SaxEx system. In this setting, adaptation is then a search problem over the space of adaptors whose goal is finding a combination of adaptor instantiations such that the final competence satisfies, via the bridge, the task goals.

From the software architecture stance what is formally required to establish a ta-bridge is only that the adaptor competence logically implies the task goals. The ABC architecture does not establish control constraints on the implementation nor distinguishes the situations where the adaptors already exist or have to be constructed from elementary adaptor components (and whether this construction is automated or performed by hand).

Adaptor-Domain bridge The *da-bridge* connects the assumptions upon domain knowledge specified by the adaptor with the domain model, in a similar way to how td-bridge maps task assumptions to the domain model. Some requirements of PSM upon domain models have already been established by connecting method with task and task with a domain. Now we only need to map the knowledge requirements that are exclusively for the method.

2.2 ABC and CBR systems design

The very idea of software architectures is learning from system developing experience in order to provide the abstract recurring patterns for improving further system development. As such, software architectures contribution in mainly methodological in providing a way to specify systems. If we consider existing CBR systems and the ABC architecture we can observe that ABC is making explicit issues that CBR system developers already know but treat implicitly when developing new systems and that they are not explicit either on the actual CBR system. Let's take the task of a CBR system, for instance. The specific task a CBR system has always to be specified, albeit informally, in the system design phase. The ABC approach considers this a specification of the task but also provides a specific way to relate that specification to each other component of the architecture: preconditions relate to the input problem, assumptions relates with the availability of knowledge, and goals relate to the search process performed by the CBR system.

Furthermore, let us consider domain knowledge. Some CBR systems use cases (and similarity) as the unique source of knowledge available to solve problems—e.g. instance-based

learning approaches. However, a great number of CBR systems use domain knowledge, for different purposes and in different ways, in addition to cases. Commonly, this domain knowledge is not described as such, but it is described by explaining the implementation of the CBR system. In other words, what is described is the representation used to encode it (rules, constraints) and the role it plays in the system implementation (mainly concerning control issues). It is our personal opinion that a clarification of the role of domain knowledge in CBR systems is needed to improve the understanding of CBR and the development of CBR systems.

As a result of focusing on the adaptation process, ABC suggests that retrieval (and similarity assessment) is also a type of domain knowledge. In our approach, solving a problem is constructing a case-specific model of the "input problem"—as was established in the knowledge-level description of CBR [5]. A software architecture is a much refined level of description, so solving a problem in ABC involves building a case-specific model that satisfies the task description goals. Domain knowledge is used to perform the inference necessary to build this model[1]. The ABC architecture does not deal with control aspects of the implementation, thus the order in which domain-specific inference and case retrieval are performed is unspecified.

3 Implementing Adaptors

We will consider two kinds of adaptors: domain adaptors (*d-adaptors*) and case-based adaptors (*c-adaptors*). "Transformational adaptation" is realized by d-adaptors, i.e. by adaptors that use some domain-specific knowledge to transform the case-specific model (in a way specified by the adaptor's competence). Moreover, that domain knowledge is the one explicitly required by the adaptor's assumptions.

"Derivational replay" is realized by c-adaptors. Case-based adaptors also transform the case-specific model but use some domain knowledge that includes a precedent case retrieved from case memory[2]. In the simplest scenario there is only on retrieved case, but in CBR systems where parts of cases are also cases each part can be adapted in a case-based way by c-adaptors. Derivational replay in planning is one example and the SaxEx system below is another example. As shown below, adaptation in the SaxEx system combines d-adaptors and c-adaptors.

The main issue to go from a specification like ABC to an actual implementation is deciding how is 1) the representation of components and bridges, and 2) the control scheme. We are implementing adaptors in Noos, a representation language designed for supporting knowledge modeling approaches to problem solving and learning [4] in which different CBR systems have been built, including SaxEx. In Noos cases are represented as feature terms [12], a formalism for representing structured cases in which any subpart of a case (feature term) is also a term—and thus is also a case. Inference is provided by problem solving methods (PSMs) that use domain knowledge to build models (or parts of models). A problem is solved when a case-specific model is completed, and then it is retained in the case base. Retrieval is performed by specialized PSMs, retrieval methods, that use domain knowledge or heuristic principles to search the case base. Concerning the control scheme, Noos inference is on demand, i.e. follows a lazy evaluation strategy. The chain of control is thus backwards: retrieval methods determine the features of a case that they need, thus forcing the evaluation of the PSMs that infer those features needed that were not part of the input problem model. Moreover, c-adaptors use retrieval methods so the retrieval process is in fact directed by the adaptation strategy.

[1] Some CBR papers distinguish between primary and derived feature cases. Primary features are those appearing on the "input case" and derived features are inferred by the system from primary features. In our approach, inference uses domain knowledge (including cases) to build a model of the problem.

[2] Recall that, for the ABC architecture, the base of cases is also part and parcel of the domain model.

The main ABC elements incorporated in Noos are i) an explicit description of a task, ii) adaptors, iii) and ta-bridges. Since the rest of the ABC elements is obviated, some parts of this elements need not be represented explicitly: the reason being that Noos will not be reasoning about them. Thus, a task holds only goals and preconditions, while adaptors holds only competence and preconditions. Assumptions are not present since we are not representing td-bridges nor da-bridges. The contents of these slots (goals, competence, preconditions) are expressed by feature terms. Satisfaction is represented as feature term subsumption ($\sqsubseteq$), thus a case-specific model C satisfies an adaptor preconditions A^i_P when $A^i_P \sqsubseteq C$ (A^i_P subsumes C).

The overall adaptation process is realized following an "Adaptation as Search" strategy. The initial state is the case-specific model of the problem; this begins with the information given as input, but the domain PSMs can enlarge this model performing inference as needed. The goal state is a complete and correct case-specific model C_F that satisfies the task goals T_G. The ta-bridge provides a translation from the task description vocabulary to the domain vocabulary used in adaptors and case specific models. Thus, the task goals expressed in domain vocabulary are obtained applying the bridge B_{TA} to the task goals $B_{TA}(T_G)$ and therefore a solution is defined as a case-specific model C_F such that $B_{TA}(T_G) \sqsubseteq C_F$.

Adaptors are applied to the case-specific model, performing search until a sequence of adaptor instantiations is found such that transforms the initial case-specific model into C_F. A classical means ends analysis technique is used with the adaptors, where *preconditions* establish if the adaptor is applicable to a particular case-specific model, and *competence* establishes the goals or subgoals achievable by instantiating the adaptor. Since Noos provides automatic backtracking, selection of adaptors and adaptor instantiation following several search strategies —such as depth-first, breadth-first, and beam search— can be easily implemented for a particular CBR system. An interesting issue left for future work is performing a case-based search of adaptor selection and instantiation: since adaptors are feature terms, they are stored in memory by Noos and they are thus amenable to be retrieved. This case-based adaptation process would be able to use both c-adaptors and d-adaptors, unifying "transformational" and "generative" adaptation in a case-based reuse of cases.

4 Adaptors in SaxEx

SaxEx reuse process uses both c-adaptors and d-adaptors, thus unifying "transformational" and "generative" adaptation. Currently, the reuse process has a first phase using c-adaptors and a second one using d-adaptors. We need now to summarize the SaxEx system in order to later focus on the use of the adaptors.

SaxEx [3] is a system for generating expressive performances of melodies based on examples of human performances (for the moment SaxEx is focused in tenor saxophone interpretations of standard jazz ballads). SaxEx consists of two modules: a) a SMS module of sound analysis and synthesis, and b) a CBR module implemented on Noos. The input of SaxEx is musical phrase with a sound track and a score in Midi format. Thus the input case-specific model is $C(sound, score)$, where the *sound* track is analyzed by SMS while the *score* is translated by Noos to feature terms. Domain knowledge consists of two musical theories: Narmour's implication/realization (IR) model [11] and Lerdahl and Jackendoff's generative theory of tonal music (GTTM) [10]. Noos employs IR and GTTM to construct two complementary models of the musical structure of the phrase. While the IR model holds an analysis of melodic surface, the GTTM model is concentrated on the hierarchical structures associated with a piece.

Thus, an enlarged case-specific model is constructed: $C(sound, score, ir, gttm)$. Models IR and GTTM are highly relational models that infer the most relevant relations among notes and groups of notes (see Fig. 3). These relational structure is then used by retrieval methods to

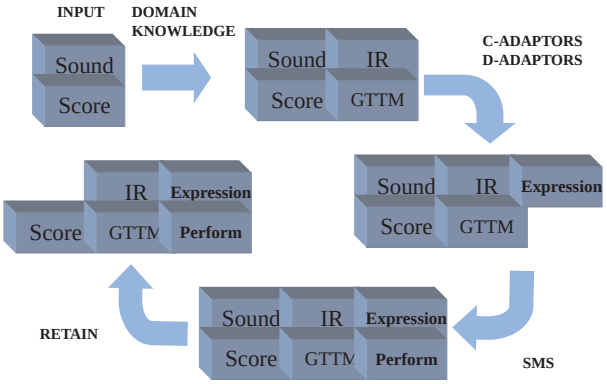

Fig. 3. The transformations of the case-specific model of the current problem in SaxEx. First, domain knowledge adds musical analysis models; then adaptors generate an expressive score; next SMS manipulates the sound track to add the expressivity features; and finally the reusable parts of the case-specific model are stores in the case base.

find in the case base other (parts of) phrases that share some musical structure. The musical structure to be shared is declared in the form of *patterns* that are used by perspectives [2] to extract and retrieve the parts of relevant musical phrases. A retrieved case has a complete case-specific model $C(score, ir, gttm, expression, performance)$ where i) *expression* is a symbolic description of the expressive parameters—such as dynamics, rubato, vibrato, and articulation—applied to each note; and ii) *performance* is a sound track with expressive performance. The goal of SaxEx is now to infer by CBR the *expression* model, and later pass it to SMS that will perform the specified changes in the sound track outputting a new expressive *performance*.

The expression model holds knowledge such as: sound amplitude (dynamics); note anticipations/delays (rubato); note durations (rubato); attack and release times (rubato and articulation); vibrato frequency and vibrato amplitude of notes; articulation mode of each note (from legato to staccato); and note attacks (allowing effects such as reaching the pitch of a note starting from a lower pitch or increasing the noise component of the sound).

Retrieval and the first phase of adaptation (using c-adaptors) are closely coupled. In this phase, SaxEx uses c-adaptors considering one note, and finishes after treating all notes in the phrase. C-adaptors use perspectives to extract a particular context around the note consisting of the closer notes, where "closer" means those notes related to the current note either on the score or on the IR and GTTM models. Retrieval methods instantiate their patterns on these contexts and use perspectives to find cases that satisfy them. The c-adaptor uses the retrieved (portions of) cases to determine the expressivity features of the current note based on the expressivity patterns of retrieved cases. There is one c-adaptor for each expressivity parameter. Examples of c-adaptors are majority, minority, strict majority and strict minority, continuity, non-continuity, and random. For instance, the majority c-adaptor chooses the values that were applied in the majority of precedents, while the continuity c-adaptor gives priority to precedent notes belonging to the same musical subphrase in the case base. The reuse strategy determines which adapter is used for each situation: musical expression being more an art than a science there is clearly no unique correct solution for the adaptation process. Reuse strategy can also be interactively set by SaxEx's user, as shown in [1].

Although the first phase of the reuse process focuses on individual notes it takes into account its immediate context as given by musical knowledge. However, there are other considerations that can be only observed taking a broader view and taking into account groups of notes. Thus, the task description goals of SaxEx specify conditions that have to be satisfied by the

expressive features of note groups. These goals establish two kinds of main criteria: smoothness and variation. Moreover this criteria are established both over single expressive features (e.g. pitch, attack) and over the relationships among expressive features (e.g. the relation between pitch and attack). Smoothness and variation are basically contradictory: the first tends to iron out strong variations, while the second, variation, is against repetition of structures and thus strengthens variations. The resulting expressive performance deals with the trade-offs among them with the aim of striking an overall balance pleasant to the ear. Moreover, the criterion used in each part of the musical piece depends on the musical model. For instance, according to the melodic structure of a given melody, rough changes can be enforced in some notes and prevented in other.

The ta-bridge instantiates these goals for the current problem in new feature terms that describe the states to be achieved; these descriptions are in terms of the vocabulary used in adaptors and the domain model. Those descriptions that are not satisfied by the current case-specific model are matched with d-adapter's competence to select those applicable. Each adaptor selected is then instantiated on the current case-specific model and produces a new model where the incorrections have been straighten out. Notice that adaptors work on note groups, so the same adaptor may be instantiated on a number of occasions over different sequences of the musical phrase.

An example of d-adaptor (based on "smoothness") is RAA that works upon a repetitive sequence of notes where attack time has been advanced–the anticipation of the note attack produces an expressive effect that is destroyed by the iteration of the same effect. The result produced by the RAA d-adaptor is a new sequence where the attack is maintained in the first note and less advanced in the rest of notes. There is a family of similar adaptors whose effect is, for specific situations, increasing or decreasing the value of an expressive feature upon a sequence of notes. A second example of d-adaptor (based on "variation") is ND that works upon a sequence of descending notes where dynamics and articulation is the same—because of this, the passage will be perceived as mechanical. The result produced by the ND d-adaptor is a new sequence where dynamics is successively decreased and the first note is emphasized (changing articulation and attack mode).

Since Noos has a lazy and on demand evaluation mode, the actual control flux follows an order opposite to the order in which we have explained the different phases and steps. At the start we have the input and the task specification that is not satisfied—causing the activation of adaptors. Since d-adaptors use the expressive features of notes, and they are not in the input, the c-adaptors are activated. Finally, since c-adaptors use musical concepts, the corresponding domain knowledge (embodied in PSMs) will be activated. This activation chain has nodes where more than one option is available, so in fact Noos spawns an activation tree performing backtracking on choice nodes. Choice and backtracking is not chronological, since a language of *preferences* is used to specify a partial order upon alternatives at choice points. Preferences are also part of the domain knowledge and they are the basic mechanism used to control adaptor selection at all points.

When the adaptation process finishes, the expressive model is passed to the SMS module starting the synthesis procedure and generating an expressive interpretation (a sound file) that can be listened and judged by the user.

5 Discussion and Related work

A conceptual framework for describing CBR systems is Richter's knowledge containers [13]. An approach towards a formal model of transformational adaptation based on the knowledge containers framework is presented in [6]. The purpose of Bergmann and Wilke's paper is to characterize when properties such as soundness and completeness can be formally proven to

hold in transformational adaptation. Interestingly, their approach centered on adaptation also seems to downplay the importance of retrieval (and similarity) in CBR systems, in a similar way as the ABC architecture conceives of retrieval as a part of domain knowledge. In our approach it is up to the designer of a CBR system to decide whether completeness is required or possible. Moreover, the designer may decide to use a logical language for specifying a ABC architecture and then formally prove that certain formal properties hold. For an approach of using UPML with automated reuse see [7]. It is an interesting question whether the knowledge containers framework could be refined to provide a software architecture with the containers as components—in which case appropriate connectors should be defined.

Acknowledgments

This research has been supported by the Project IST-1999-19005 IBROW *An Intelligent Brokering Service for Knowledge-Component Reuse on the World-Wide Web*, and the CICYT Project SMASH: *Systems of Multiagents for Medical Services in Hospitals.*

References

1. Josep Lluís Arcos and Ramon López de Mántaras. An interactive CBR approach to generate expressive music. *Journal of Applied Intelligence*. In Press.
2. Josep Lluís Arcos and Ramon López de Mántaras. Perspectives: a declarative bias mechanism for case retrieval. In David Leake and Enric Plaza, editors, *Case-Based Reasoning. Research and Development*, number 1266 in Lecture Notes in Artificial Intelligence, pages 279–290. Springer-Verlag, 1997.
3. Josep Lluís Arcos, Ramon López de Mántaras, and Xavier Serra. Saxex : a case-based reasoning system for generating expressive musical performances. *Journal of New Music Research*, 27 (3):194–210, 1998.
4. Josep Lluís Arcos and Enric Plaza. Inference and reflection in the object-centered representation language Noos. *Journal of Future Generation Computer Systems*, 12:173–188, 1996.
5. E. Armengol and E. Plaza. A knowledge level model of case-based learning. In S. Wess, K.D. Althoff, and M. Richter, editors, *Topics in Case-Based Reasoning*, number 837 in Lecture Notes in Artificial Intelligence, pages 53–64. Springer-Verlag, 1993.
6. R. Bergmann and W. Wilke. Towards a new formal model of transformational adaptation in case-based reasoning. In *European Conference on Artificial Intelligence (ECAI'98)*, 1998.
7. D. Fensel and V. R. Benjamins. Key issues for automated problem-solving methods reuse. In *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI-98)*, pages 63–67, 1998.
8. D. Fensel, V. R. Benjamins, M. Gaspari S. Decker, R. Groenboom, W. Grosso, M. Musen, E. Motta, E. Plaza, G. Schreiber, R. Studer, and B. Wielinga. The component model of upml in a nutshell. In *Proceedings of the International Workshop on Knowledge Acquisition KAW'98*, 1998.
9. D. Garland and D. Perry (Eds.). Special issue on software architectures. *IEEE Transactions on Software Engineering*, 1995.
10. Fred Lerdahl and Ray Jackendoff. An overview of hierarchical structure in music. In S.M. Schwanaver and D.A. Levitt, editors, *Machine Models of Music*, 1993.
11. Eugene Narmour. *The Analysis and cognition of basic melodic structures : the implication-realization model.* University of Chicago Press, 1990.
12. Enric Plaza. Cases as terms: A feature term approach to the structured representation of cases. In M. Veloso and A. Aamodt, editors, *Case-Based Reasoning, ICCBR-95*, number 1010 in Lecture Notes in Artificial Intelligence, pages 265–276. Springer-Verlag, 1995.
13. M. M. Richter. The knowledge contained in similarity measures, 1995. Invited talk to ICCBR-95. Available at http://wwwagr.informatik.uni-kl.de/ lsa/CBR/.
14. K. Seta, M. Ikeda, T. Shima, O. Kakusho, and R. Mizoguchi. Clepe: a task ontology based conceptual level programming environment. *Trans. of IEICE*, (9), 1999.