

Constructive Adaptation

Enric Plaza and Josep-Lluís Arcos

IIIA-CSIC - Artificial Intelligence Research Institute
Campus UAB, 08193 Bellaterra, Catalonia, Spain.
Vox: +34-93-5809570, Fax: +34-93-5809661
Email: {enric,arcos}@iiia.csic.es

Abstract. Constructive adaptation is a search-based technique for generative reuse in CBR systems for configuration tasks. We discuss the relation of constructive adaptation (CA) with other reuse approaches and we define CA as a search process in the space of solutions where cases are used in two main phases: hypotheses generation and hypotheses ordering. Later, three different CBR systems using CA for reuse are analyzed: configuring gas treatment plants, generating expressive musical phrases, and configuring component-based software applications. After the three analyses, constructive adaptation is discussed in detail and some conclusions are drawn to close the paper.

1 Introduction

Classically, adaptation methods have been classified as generative reuse versus transformational reuse [1]. Derivational (or analogical) replay is the paradigmatic method of generative reuse and has been used in planning tasks. The basic idea in derivational replay is that the trace of the problem solving process in a retrieved case is obtained and replayed (re-instantiated) into the context of the current problem. In transformational reuse there are a number of transformational operators that are applied to (a copy of) the solution of a retrieved case until a solution consistent with the new problem is achieved. Constructive Adaptation (CA) is a form of generative reuse in that the solution of the new problem is constructed (rather than transformed). For this reason it is similar (but not identical) to derivational replay. However, constructive adaptation uses the solution of retrieved cases as such, not the trace of solving the case.

Constructive adaptation, in abstract terms, can be described as a search process in the space of solutions where cases are used in two main phases: hypotheses generation and hypotheses ordering. Before explaining the technique of constructive adaptation we discuss its relation with other generative and transformational techniques for reuse in §2. Then, we present in §3 the main elements of constructive adaptation and the search process over states (representing partial solutions) that is at the core of CA. For better understanding CA three CBR systems using constructive adaptation as reuse method are analyzed. These

CBR systems perform tasks where solutions are complex structures of elements in widely different domains. The three CBR systems have the tasks of configuring gas treatment plants, generating expressive musical phrases, and configuring component-based software applications. After analyzing the CA reuse process in three CBR systems, constructive adaptation is discussed in detail in §4, and finally some conclusions are drawn to close the paper.

2 Reuse in a nutshell

There have been several studies with the goal to systematize the different adaptation techniques used in CBR [13, 14]. In this section we will briefly summarize different approaches to adaptation exclusively with the goal of understanding the relation of constructive adaptation with other existing techniques. We will first discuss the notions of reuse and adaptation, then we will review the major adaptation techniques and compare them with constructive adaptation.

We can distinguish analytical tasks from synthetic tasks; we will see that reuse is quite different for CBR systems performing analytical or synthetic tasks. An analytical task is one in which solutions are expressed as an enumerated collection of elements, typically called classes—and thus the task is usually called classification or identification. A synthetic task is one where the number of solutions is so large that they are not enumerated; instead there are *solution elements* and a solution is a composite structure of these elements. The possible compositions determine the solution search space, and problem solving is a process that is able to find a composition that is a solution. Instead of saying that classification CBR has no adaptation and synthetic CBR has adaptation, the notion of *reuse* was introduced in [1] in order to encompass the processing of knowledge obtained from retrieved cases in synthetic and analytical tasks. Thus, synthetic tasks have adaptation techniques for realizing the reuse process, while analytical tasks have techniques that assess the retrieved cases for deciding which is the solution of a problem. For instance, in *k*-nearest neighbor the *retrieve* process determines the number *k* and those cases that are most similar to the problem, while the *reuse* process uses that information with a specific technique—e.g. using retrieved cases to “vote” on classes and taking as solution the class with most votes.

Concerning adaptation techniques for reuse on synthetic tasks, they fall into two families: transformational adaptation and generative adaptation. Transformational adaptation (see [9, 10]) takes the description of a problem and a retrieved case (including the solution description) and transfers the retrieved solution by modifying it until a new solution structure is achieved that is “consistent” (or “adequate”) for the new problem. Transformational adaptation can be analyzed in more detail, for instance *compositional adaptation* is defined in [13] and [14] as a form of adaptation where solution parts coming from multiple cases are adapted and combined together. For our purposes, we will consider compositional adaptation as a modality of transformational adaptation where the solution structures being transformed originate not from one but several

Table 1. Comparison of derivational, constructive, and transformational methods for reuse considering the *basis* of information used to take decisions and the way a *solution* is build.

<i>Method</i>		<i>Basis</i>	<i>Solution</i>
Generative	Derivational	trace	construct
	Constructive	case	construct
Transformational		case	transform

cases. This view is consistent with the fact that both derivational adaptation (e.g. derivational analogy in [11]) and constructive adaptation (as shown later in the paper) may use information from one or several cases¹.

Derivational (or generative, as sometimes is called) adaptation is based on augmenting the case representation to detailed knowledge of the decisions taken while solving the problem, and this recorded information (e.g. decisions, options, justifications) is used to “replay” them in the context of the new problem. As originally defined in *derivational analogy* [6] for planning systems the cases contain *traces* from planning process performed to solve them; also it is stated that in Prodigy/Analogy stored plans are annotated with plan *rationale* and reuse involves adaptation driven by this rationale [12].

Let us define *generative adaptation* as a process that uses information from retrieved cases to *construct* a (new) solution for the current problem. From this point of view, we have two large families of adaptation techniques: transformational adaptation (including compositional, structural, etc varieties) and generative adaptation — that includes both derivational adaptation and constructive adaptation. Table 1 shows the main features of these techniques:

Generation vs. Transformation. The solution is generated (using case information) in generative adaptation while it is derived by transforming old solution(s) in transformational adaptation;

Cases vs. Traces. Derivational adaptation use annotations of the problem solving process (the “trace”) in the retrieved cases while constructive adaptation uses just the cases (i.e. a description of the problem and the solution without intermediate information about how the system went from problem to solution during problem solving).

In the following sections we will present constructive adaptation as a reuse method for configuration tasks, where we use *configuration* as a general term for tasks where the solution is a structure of relations among elements. Specifically, we will present applications of constructive adaptation to the design of plants for gas treatment (§ 3.1), generation of expressive musical phrases (§ 3.2) and configuration of CBR systems based on software components (§ 3.3). We will not consider planning tasks in our framework; although it could be included in this

¹ Another distinction is made in [14] between transformational and structural adaptation; since both are based on reorganizing solution elements provided by retrieved cases it will suffice for our purpose here to subsume structural adaptation as a modality that fits in our definition of transformational adaptation

generic definition planning involves a very specialized collection of approaches (both in the case-based planning approaches and in the planning community at large) that focus on the *sequential* structure of plans.

3 Constructive adaptation

Succinctly, constructive adaptation (CA) is a form of best-first heuristic search in the space of solutions that uses information from cases (solved problems) to guide that search. Different modalities of CA can be developed: e.g. the search process can be exhaustive or not, the representation of cases and states can be identical or not. We are now going to characterize CA in more detail by defining its two constitutive processes or functions: *Hypotheses Generation* and *Hypotheses Ordering*. Subsequently, we will show three specific realizations of CA in three different CBR systems.

Let us start defining a case $C_i = (P_i, K_i)$ as a pair of problem description P_i and a solution K_i . The problem description P_i include the problem requirements $Req(P_i)$, usually the input provided by the user. Since the solution is a *configuration*, we can consider that —given a CBR system using a representation language with concepts and relations— a configuration is a *structure* of those concepts and relations. Let us denote \mathcal{K} the set of possible configurations expressible in a language. Notice that the set \mathcal{K} contains both partial and complete configurations; thus we will note the set of complete configurations $\mathcal{K}_C \subset \mathcal{K}$. The solution of a case is a configuration that is both complete and valid. A configuration K_i is complete when $K_i \in \mathcal{K}_C$. Moreover, we say that a solution is *valid* if $Sat(Req(P_i), K_i)$, i.e. if the solution satisfies the input requirements.

Constructive adaptation works upon *states*. A state is a domain-specific representation of the information needed to represent a partially specified solution, in our case a *partial configuration*. We will assume that there is a function $SAC : \mathcal{S} \rightarrow \mathcal{K}$ (where \mathcal{S} is the set of states expressible in a CBR system) such that given a state $s \in \mathcal{S}$ as input SAC yields a corresponding (partial) configuration $K \in \mathcal{K}$ in the language used in the case base. Notice that a state contains more information than the partial configuration: it may contain the user input requirements, intermediate values used for problem solving, etc. There are CBR systems that may use the case representation itself as a way to represent a state — this option is often taken in transformational adaptation, since transformation rules and operators are defined upon the structure of cases.

In general, however, representation of state and case need not be identical. We introduce this distinction not only for theoretical clarification, but also for practical purposes: it has been convenient to implement state representation as distinct from case representation. The rationale can be summarized as follows: *cases* have representation biased towards storing and retrieving “experience episodes” of problem solving, while *states* have representation biased towards the search-based problem solving.

After introducing the basic elements of CA (cases, configurations, and states) we will describe the *process* of constructive adaptation shown in Figure 1. The

```

Initialize OS = (list (Initial-State Pi))
Function CA(OS)
  Case (null OS) then No-Solution
  Case (Goal-Test (first OS)) then (SAC (first OS))
  Case else
    Let SS = (HG (first OS))
    Let OS = (HO (append SS (rest OS)))
    (CA OS)

```

Fig. 1. The search process of constructive adaptation expressed in pseudo code. Functions HG and HO are *Hypotheses Generation* and *Hypotheses Ordering*. Variables OS and SS are the lists of *Open States* and *Successor States*. The function SAC maps the solution state into the *configuration* of the solution. Function Initial-State maps the input requirement P_i into a state.

CA process is a best-first search process with two basic functions (*Hypotheses Generation* and *Hypotheses Ordering*) and two auxiliary functions (*Goal Test* and *Initial State*). *Goal Test* is simply a function that given a state s checks whether or not it is a solution—i.e. if the corresponding configuration $SAC(s)$ is complete and valid. *Initial State* is a function that maps from the input requirements of the system $Req(P_i)$ to the *initial* state received by CA.

The process of *Hypotheses Generation* is domain specific and can use knowledge acquired from cases (CK) and from domain models containing general knowledge (GK), or both. Given s , an *open state*, *Hypotheses Generation* “expands” this state, i.e. generates the *successor* states of s . The essential idea of *Hypotheses Generation* is that when several options exist about elements or relations that can be added to the configuration of the state $SAC(s)$ then each option is considered a possible hypothesis, and a new (successor) state is generated incorporating one hypothesis.

In order to decide which open state s is selected to be expanded the *Hypotheses Ordering* function is used to rank the open states; CA (see Fig. 1) then selects the best one according to this ordering. The process of *Hypotheses Ordering* is domain specific and can use knowledge provided by cases (CK) and from domain models containing general knowledge (GK), or both.

The CA search process is summarized in Fig. 1. CA starts receiving (a list with) an initial state generated from the problem description P_i by auxiliary function *Initial State*. The CA algorithm works with an ordered set of hypotheses, the list of open states OS. Being recursive, CA checks first the termination conditions: a) if OS is empty all possible states have been explored, and CA terminates because there is no solution, and b) if the best state in OS (the first in the ranking of open states) passes the *Goal Test* this state is a solution for P_i . The recursive step in Fig. 1 generates the successor states SS of the best state in OS using the *Hypotheses Generation* function and they are added to the open states OS; then the list OS is re-ordered using the *Hypotheses Ordering* function and this is passed to the recursive call of CA.

Since *Hypotheses Generation* and *Hypotheses Ordering* are both domain-specific (as well as the representation of state) the explanation so far of CA has been very abstract. The best way to understand more concretely CA is through several case studies where specific CBR systems use specific *Hypotheses Generation* and *Hypotheses Ordering* functions. The following sections describe constructive adaptation in several CBR systems focusing on how states are represented, and how *Hypotheses Generation* and *Hypotheses Ordering* use case knowledge (CK) and domain knowledge (GK).

3.1 Design of gas treatment plants

T-Air is a case-based reasoning application developed for aiding engineers in the design of gas treatment plants [2] developed at the IIIA for the Spanish company TECNIO. The gas treatment is required in many and diverse industrial processes such as the control of the atmospheric pollution due to corrosive residual gases which contain vapors, mists, and dusts of industrial origin. Examples of gas treatments are the absorption of gases and vapors such as SO_2 , CLH , or CL_2 ; the absorption of NO_x with recovering of HNO_3 ; the absorption of drops and fogs such as PO_4H_3 or $CLNH_4$; dust removal in metallic oxides; and elimination of odors from organic origin.

The main problem in designing gas treatment plants is that the diversity of possible problems is as high as the diversity of industrial processes while the experimental models about them is small. The knowledge acquired by engineers with their practical experience is the main tool used for solving new problems. For the design of gas treatment plants about forty different types of main equipment have been covered. We also started with a case-base comprising one thousand solved problems (cases) involving, each of them, from two to twenty different types of equipment.

A *solution* for a *T-Air* problem (Fig. 2) is a configuration holding the required equipments (mainly scrubbers, pumps, tanks, and fans), the design and working parameters for each equipment, and the topology of the installation (the gas circuit and the liquid circuits). The CBR process is organized on three *task levels*: a) selecting the class of chemical process to be realized, b) selecting the major equipments to be used (and their inter-connections), and c) adding auxiliary equipment and assessing the values for the parameters of each equipment.

A *state* in *T-Air* contains information about a) the input requirements, b) the corresponding partial configuration, c) a collection of *open issues*, and d) hypotheses supporting cases. The open issues represent the collection of decisions pending to be resolved in the state's partial configuration. For every hypothesis incorporated into a state a set of *supporting cases* is recorded; they are the cases from which this particular hypothesis was derived. The initial state is formed by the input requirements plus the initial open issue— namely, selecting the class of the chemical process to be realized.

Hypotheses Generation. The successor function takes the best state (see below how “best” is assessed) and selects as current issue one of the open issues— this selection is based on a domain model (GK). If the issue belongs to the *task*

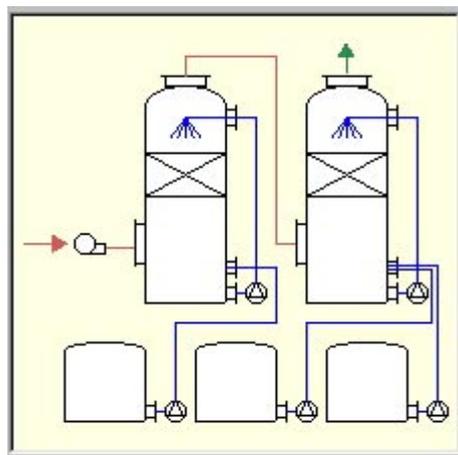


Fig. 2. A solution in *T-Air* is the structure of equipments of a gas treatment plant.

level of chemical process class the new hypotheses are generated using a case-based process (CK). *T-Air* retrieves cases based on the input requirements (and some derived features inferred from the requirements). Then *T-Air* generates one hypothesis (a successor state) for each chemical process class found in the retrieved cases. For instance, odor elimination can be solved by *adsorption*² or *absorption*. The successor states for *adsorption* and *absorption* will also record the *supporting cases* corresponding to each hypothesis.

If the open issue belongs to the *task level* of equipment selection the new hypotheses are generated using a case-based process (CK). *T-Air* retrieves cases based on the input requirements and the chemical process class; a new hypothesis is generated by each “core equipment” found in the retrieved cases. A new hypothesis is a new state with a partial configuration that incorporates one of the “core equipments”, the new *open issues* corresponding to this equipment, and the supporting cases for the hypothesis. For instance, odor elimination with absorption can be realized with the core equipment “two scrubbers, a pump, and a fan” or “one multiventuri, a pump, and a fan”.

If the open issue belongs to the third *task level* the hypotheses to be generated concern auxiliary equipment and parameter assessment. The major parameters are assessed using a case-based process (CK) because there are no analytical models capable of estimating them, and the TECNIUM company experience (represented as cases) is the only available knowledge³. However, when a parameter is assessed to have a certain value, this may provoke the rising of new open issues that need to be solved. These open issues can be solved adding auxil-

² Adsorption is the use of solids for removing substances from either gaseous or liquid solutions.

³ There are less critical parameters that can be computed using analytical methods, and *T-Air* uses them when available.

iliary equipment to the configuration. For instance, when the chemical reaction is exothermic and the gas flow exceeds a certain threshold a new open issue for refrigerating the involved equipment is generated. This issue can be solved by the addition of a refrigerator as auxiliary equipment.

Hypotheses Ordering. The ordering of hypotheses is performed by an assessment heuristic based on both domain knowledge (GK) and cases (CK). The assessment heuristic takes into account several dimensions: i) estimated overall cost (GK), ii) plant overall reliability (GK and CK), iii) critical parameter values (GK), and iv) supporting cases (CK). The assessment heuristic receives a state as input and estimates these dimensions on the partial configuration that corresponds to that state; the pending states are ordered by this heuristic. This means that, all other things equal, the *T-Air* system will first explore the states corresponding to configurations that have greater number of related cases in the case base.

Finally, the *Goal-Test* function checks whether a state is a solution —i.e. whether the configuration of the state is complete and valid (using domain knowledge). The search process is not *exhaustive* because there is no guarantee that the Hypotheses Generation function will generate all *possible* hypotheses. Hypotheses Generation uses cases retrieved from the case base to pinpoint the hypothesis to be considered.

3.2 Expressive music generation

SaxEx [4] is a system for generating expressive performances of melodies based on examples of human performances (currently SaxEx is focused in tenor saxophone interpretations of standard jazz ballads). The input of SaxEx is a musical phrase with a sound track and a score in Midi format plus some affective labels characterizing the intended mood for the expressive performance. In the notation introduced in §3 this input is the $Req(P)$ part of the problem description. Two musical theories are used by SaxEx: Narmour’s implication/realization (IR) model and Lerdahl and Jackendoff’s generative theory of tonal music (GTTM). SaxEx 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 concentrates on the hierarchical structures associated with a piece. The problem description P is formed both by the input $Req(P)$ and the musical structures inferred using these domain models.

A solution for a problem in SaxEx is a sequential structure of notes belonging to a phrase, where each note has an associated *expressive model*. The expressive model holds the following parameters: 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). Summarizing, a solution is achieved when each note has an assigned value for each expressive parameter.

A *state* in SaxEx contains information about i) the input requirements, ii) the expressive models generated up to this point, and iii) the set of *open notes* (the notes in the phrase without an expressive model). The *initial state* is formed by the input requirements and all the notes of the musical phrase as open notes.

Hypotheses Generation. Given a state s Hypotheses Generation selects the next note from open notes and uses cases (CK) for generating several expressive models for that note —each one embodied in a new successor state. Notice that the hypotheses generated are expressive models and not individual expressive parameters. For each note, a set of similar notes is retrieved using the mechanism of perspectives [3]. Analyzing the expressive models of the retrieved notes, with the help of musical knowledge that constrain over the possible combinations of values, several (alternative) expressive models are generated, each with an assessment of *note similarity* —comparing the problem note and the retrieved notes involved in each expressive model.

Hypotheses Ordering. The ordering of hypotheses uses domain knowledge (GK) and case knowledge (CK). Domain knowledge assesses the coherence of the different expressive models in a state. This assessment takes into account that the expressive models represent the way the melody (a sequence) will be performed. SaxEx establishes two kinds of main coherence criteria: smoothness and variation. Moreover, these criteria are established both over single expressive parameters (e.g. pitch, attack) and over the relations among expressive parameters (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. Case knowledge is used to estimate an overall similarity value of a state with respect to the cases used in generating that state. This is done by aggregating the *note similarity* values of the notes with expressive model belonging to the state. The Hypotheses Ordering function combines these two assessment values into a state “goodness” value and ranks the *open states* according to these values.

Finally, the *Goal-Test* function just checks that the solution is complete and valid. Validity in SaxEx is defined by two threshold values of smoothness (min) and variation (max) that have to be satisfied by a final state. Smoothness and variation thresholds can be set by the user according to her particular musical interests.

3.3 Component-based software configuration

Finally, we will present a CBR system using CA for configuring software applications from a library of software components called *CBR broker*. The *CBR broker* only assumes that the software components are expressed in the UPML (Universal Problem-solving Methods Language) formalism [7]. In a nutshell, UPML can describe *tasks* (what is to be achieved), *problem-solving methods* (how can a task be achieved), and *domain models* (knowledge needed to achieve tasks). Task and

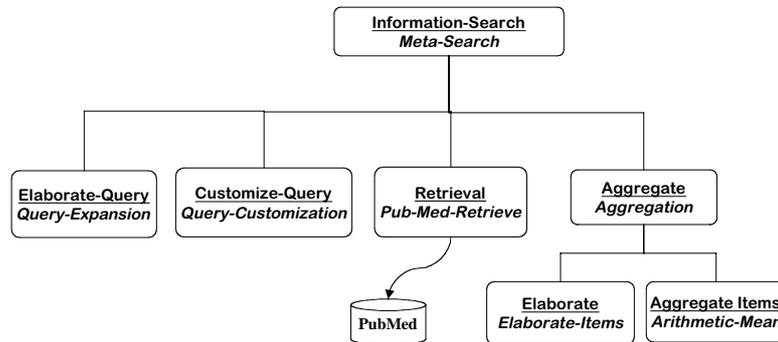


Fig. 3. A configuration in the WIM multiagent system. A box contains the binding of a task and a *PSM*, while a cylinder is a domain model.

PSMs (problem-solving methods) are characterized by their input/output signature and their competence; the competence of a task or PSM is described by *preconditions* (statements of what is supposed to hold for the component to be applicable) and *postconditions* (statements about what holds after applying the component). A domain model is characterized by properties of the knowledge it contains.

A PSM elementary or be can decomposed into subtasks; when a PSM is a decomposition the *CBR broker* has to find which PSMs can achieve those (sub) tasks and when the PSM is elementary the *CBR broker* has to find which domain models are required. The *CBR broker* receives as input a *competence requirement* specifying the preconditions, postconditions and available domain models for building a target application. The output of the *CBR broker* is a *configuration*: a structure specifying a) for a task, a specific PSM able to achieve that task, and b) for each elementary PSM, which domain models are used — we call those association *bindings* (see Fig. 3). We say a configuration is *complete* when all bindings have been resolved, and *valid* when the input competence requirement is satisfied; a configuration is a solution when it is complete and valid. The *CBR broker* stores in the case base those configurations obtained in the past and *CA* uses them to guide the search process over the solution space.

The *CBR broker* has been used in a multi-agent system for retrieval and integration of medical information in databases called WIM (Web Information Mediator) [8]. The Problem Solving Agents register their competence as PSMs into a Library using the UPML language. When a User Agent has some particular task to achieve for its user it sends a competence requirement to the broker, that interacts with the Librarian Agent to obtain UPML specifications and finally reaches a complete and valid configuration. Since a configuration is just an abstract specification there is now the need to *operationalize* it into a working system—in this case, a team of agents with the adequate competences.

This process of *team formation* is achieved by a negotiation process between the CBR broker and the registered Problem Solving Agents [8].

A *state* in *CBR broker* corresponds to a partial configuration. A state holds information about *tp-bindings* (task/PSM bindings), preconditions and postconditions. The *closed bindings* is the collection of task/PSM bindings in the partial configuration of that state. The *open bindings* are those bindings not yet resolved (tasks without a PSM bound to it). Moreover, a state has open (resp. closed) preconditions and postconditions: they are those pre- and postconditions not yet satisfied (resp. already satisfied) by the current partial configuration. The *initial state* is created from the input competence requirement: the input preconditions are closed preconditions (we assume they are satisfied) and the input postconditions are open postconditions.

Hypotheses Generation. The *CBR broker* takes one task T from the open bindings; several hypotheses can be generated for T , each one a PSM capable of achieving T . The *CBR broker* uses the notion of *component matching* to select PSMs that can meaningfully achieve a task. Component matching is defined as follows: a PSM M matches a task T when a) their input/output signatures are consistent, b) all the postconditions of T are satisfied by M , and c) all the preconditions of M are satisfied by T ⁴.

For each PSM that matches T a new state is generated where T is bound to one of these PSMs. The new *tp-binding* is called **current tp-binding** and is added to the closed bindings. Moreover, the rest of the information of the state is updated as follows. If the new PSM has subtasks they are added to open bindings; if the new PSM is elementary the required domain models are associated with it (if some are not available this new state is not valid and is not generated). The open postconditions satisfied by the postconditions of the new PSM are deleted and become closed postconditions, while its new preconditions become open preconditions.

Notice that only general knowledge (GK) is used in Hypotheses Generation, since the options are retrieved from the UPML library using the notion of *component matching*. Because of this, *CBR broker* performs an exhaustive search with respect to the Library of components being used.

Hypotheses Ordering. The *CBR broker* uses only case knowledge (CK) to rank the open states (the partial configurations); the cases are pairs $C_i = (P_i, K_i)$ where P_i is the input competence requirement and K is the configuration found for that input. First, the *CBR broker* computes the similarity between the problem P and each case input description P_i using the LAUD structural similarity distance[5]⁵. Thus, LAUD provides a ranking over the set of cases in

⁴ Notice that this condition is the converse of the previous one. The reason is that preconditions are assumptions about what is true in the world so that a component is applicable. If a PSM can achieve the same postcondition requiring less assumptions than the task specified, the PSM still satisfies that task.

⁵ A structural similarity is needed because the problem descriptions are represented as feature terms; in particular, pre- and postconditions are sets of feature terms that require a distance measure capable to deal with relational cases.

Table 2. Dimensions that characterize Constructive Adaptation (CA) in the three CBR systems reviewed. Keywords used in the table: GK = General (Domain) Knowledge, CK = Case (based) Knowledge.

CA Dimensions	<i>T-Air</i>	SaxEx	CBR Broker
Hypothesis Generation	GK & CK	CK	GK
Hypothesis Ordering	GK & CK	GK & CK	CK
Exhaustive Search	No	No	Yes

the case base. The next step is to transfer this knowledge to the set of open states and rank them accordingly. Since each open state incorporates a new hypothesis in the form of the *current tp-binding*, the *CA-broker* searches in the ranked cases which C_i has the same task/PSM binding in the solution. The state is given as *endorsement value* the similarity of the highest ranking case containing the state's current *tp-binding* and all the open states are ranked according to their endorsement values.

Finally, *Goal-Test* checks whether a state is a solution, i.e. there are no open bindings and all input competence requirements pre- and postconditions are satisfied by the closed pre- and postconditions.

4 Constructive adaptation revisited

After the description of three CBR systems using constructive adaptation (CA) we can review the basic notions of this reuse technique. Basically, CA is a generative technique for reuse (since CA constructs a solution using case information). We have seen in the exemplified CBR systems that both general domain knowledge (GK) and case knowledge (CK) can be used inside CA and that they can be used for generating hypotheses and/or ordering hypotheses. Table 2 characterizes several dimensions of constructive adaptation, with the columns characterizing the three CBR systems reviewed in §3.

The first dimension, Hypothesis Generation, specifies whether general domain knowledge (GK) and/or case knowledge (CK) is used in generating new hypotheses while generating successor states. We can see that *T-Air* uses both GK and CK while SaxEx and the *CBR Broker* use only CK and GK respectively. The second dimension, Hypothesis Ordering, specifies whether general domain knowledge (GK) and/or case knowledge (CK) is used in ordering the hypotheses to be considered (i.e. ranking the open states). We can see that the *CBR Broker* uses only CK while the other two combine general and case knowledge.

Finally, the Exhaustive Search dimension specifies whether the search process is able to consider all possible solutions. Only the *CBR Broker* performs an exhaustive search (with respect to the totality of the components in the Library). Since the three systems provide just one solution (and not all valid solutions) this means that exhaustivity only assures that when no solution is found it's because it does not exist. The reason for being exhaustive is that Hypothesis Generation uses GK to retrieve *all* components in the Library that match a

task specification; retrieved cases are used only to order the hypotheses. *T-Air* and *SaxEx* are not exhaustive because CK and GK is used to focus the search process only on those hypotheses that have some endorsement from general or case knowledge. Exhaustivity is not a good thing per se, it is a property that a system may or may not need. *T-Air* has as a user an engineer of TECNIUM company that uses past cases (plants engineered by TECNIUM) to achieve rapid prototyping of new plants. There are always several solutions and the system helps to find the good ones in terms of economy and ease of construction. In *SaxEx* the number of possible solutions is always so large that exhaustivity is not an issue.

5 Conclusions

The overall view of constructive adaptation, at this point, is that it's a flexible technique for using cases and general knowledge in CBR systems for synthetic tasks. The three reviewed CBR systems have quite different application domains, and the only common aspect is that the solution to be built by CBR is a complex structure of elements. Constructive adaptation offers a way to organize reuse processes into the well founded paradigm of state-based search, and clarifies the phases where cases (and general knowledge) can be used, namely generation and ordering of hypotheses. The exact way in which cases are used to make the decisions involved in generation and ordering is open and may vary from one application domain to the other. However, applying CA to these CBR systems, understanding how to use cases was easy once we had a clear idea of what a *state* was in the system. As a lesson learned from these experiments we can say that the main issue to apply CA was to clarify what information was to be present in a state; once this was clear the rest of the elements in CA were easy to design: hypotheses generation and ordering, and goal test.

Let us consider now the relation between constructive adaptation and derivational reply: in §2 we considered both as techniques for generative reuse, the difference being that derivational reply uses cases augmented with problem solving data (the *trace* that is later *replayed* while CA just uses the cases —conceived as (P, S) problem solution pairs. This difference comes from the distinction of *states* and *cases* in CA. When the solution state is found, a solution configuration is built, but no “trace” information is stored— trace information is contained in the search branches that generate, evaluate and discard states, and it is discarded when only the final configuration *S* is stored in a case. The reason why it is so in CA was simply because this information was not needed. In retrospect, we think this difference is due to the nature of planning tasks and the type of search process required for planning. A plan is a sequential structure with a total or a partial order. Planning can be seen as a search problem but its nature has required that AI researchers developed specific methods, heuristics, and representations for planning tasks. Thus, we think that recording trace information on plans stored as cases comes naturally because it fits this specialized approaches of search processes for planning.

Our approach, however, has been focused on configuration tasks where a solution is a structure of elements and their relations. Constructive adaptation proceeds by adding elements to the solution and relating it to the other elements; if more than one element can be added, these alternative options are interpreted as alternative hypotheses embodied in several states. Cases can be used for generating and ordering hypotheses, and for this purpose there was no need to store in cases information about the trace of the search process. The information needed from the cases can be directly obtained from their solution structures, in a specific way adequate for each application domain.

Finally, there are different types of search processes that can be implemented in the framework of constructive adaptation. The search process in adapting cases can be exhaustive when all options (hypotheses) are always generated and cases are used to rank the order in which each combination of hypotheses (state) is explored— e.g. the *CBR broker* of §3.3. However, in applications where there is no general knowledge available cases can be used to generate alternative hypotheses; this process is performed in the *SaxEx* system although general (musical) knowledge is also used.

Generating hypothesis from a case base does not imply that the search process of CA is exhaustive or not: it depends on the case base and on the kind of memory search that the retrieval method performs. We have seen that *T-Air* and *SaxEx* are not exhaustive, but the reason for this concerns the type of CBR system being developed. It is possible to have an exhaustive search process in CA when hypotheses are generated from CK and not GK, as in the following example. Consider a variant of the *CBR broker* where there is no Library of UPML components and just a case base of configurations of components. Let us suppose that the case base has a number of cases that assures that every component originally in the Library is used in at least one configuration case. In this scenario, the *CBR broker* can use the definition of *component matching* in §3.3 to search the memory of configurations and find all components that satisfy some requirements. Therefore, with certain conditions on the case base (when the number and variety of cases are a good sample the possible solutions) and on the retrieval method over cases (finding all items relevant for a specific context) the CA search process can be exhaustive.

Acknowledgments

This research has been supported by the Esprit Long Term Research Project 27169: *IBROW An Intelligent Brokering Service for Knowledge-Component Reuse on the World-Wide Web*, and the TIC Project 2000-1094-C02 *Tabasco: Content-based Audio Transformation using CBR*.

References

1. Agnar Aamodt and Enric Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *Artificial Intelligence Communications*, 7(1):39–59, 1994.

2. Josep Lluís Arcos. T-Air: A case-based reasoning system for designing chemical absorption plants. In David W. Aha and Ian Watson, editors, *Case-Based Reasoning Research and Development*, number 2080 in Lecture Notes in Artificial Intelligence, pages 576–588. Springer-Verlag, 2001.
3. 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.
4. J. L. Arcos and R. López de Mántaras. An interactive case-based reasoning approach for generating expressive music. *Applied Intelligence*, 14(1):115–129, 2001.
5. Eva Armengol and Enric Plaza. Similarity assessment for relational CBR. In David W. Aha and Ian Watson, editors, *Case-Based Reasoning Research and Development, 4th International Conference on Case-Based Reasoning, ICCBR 2001*, volume 2080 of *Lecture Notes in Computer Science*, pages 44–58, 2001.
6. Jaime Carbonell. Derivational analogy: A theory of reconstructive problem solving and expertise acquisition. In *Machine Learning: An Artificial Intelligence Approach*, volume 2, pages 371–392. Morgan Kaufmann, 1986.
7. 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.
8. Mario Gómez, Chema Abásolo, and Enric Plaza. Domain-independent ontologies for cooperative information agents. In *Proceedings Workshop on Cooperative Information Agents*, volume 2128 of *LNAI*, pages 118–129, 2001.
9. K. J. Hammond. *Case-based planning: Viewing planning as a memory task*. Academic Press, 1989.
10. T. Heinrich and J. L. Kolodner. The roles of adaptation in case-based design. In *Proceedings of the AAAI Workshop on Case-based Reasoning*, 1991.
11. Manuela Veloso and Jaime Carbonell. Toward scaling up machine learning: A case study with derivational analogy in prodigy. In Steven Minton, editor, *Machine Learning Methods for Planning*, pages 233–272. Morgan Kaufmann, 1993.
12. Manuela M. Veloso, Alice M. Mulvehill, and Michael T. Cox. Rationale-supported mixed-initiative case-based planning. In *AAAI/IAAI*, pages 1072–1077, 1997.
13. W. Wilke, B. Smyth, and P. Cunningham. *Using configuration techniques for adaptation*, pages 139–168. Number 1400 in *LNAI*. Springer Verlag, 1998.
14. W. Wilke and R. Bergmann. Techniques and knowledge used for adaptation during case-based problem solving. In *IEA/AIE (Vol. 2)*, pages 497–506, 1998.