Research Report RR-IIIA-2005-01 (Extended version of a submitted article)

Cooperative Reuse for Compositional Cases in Multi-Agent Systems

Enric Plaza

IIIA - Artificial Intelligence Research Institute
CSIC - Spanish Council for Scientific Research
Campus UAB, 08193 Bellaterra, Catalonia (KoS)
Vox: +34-93-5809570, Fax: +34-93-5809661
Email: enric@iiia.csic.es
WWW: http://www.iiia.csic.es

Abstract. We present a form of case-based reuse conducive to the cooperation of multiple CBR agents in problem solving. First, we present a form of constructive adaptation for configuration tasks with compositional cases. We then introduce CoopCA, a multi-agent constructive adaptation technique for case reuse. The agents suggest possible components to be added to the ongoing configuration problem, allowing an open, distributed process where components used in cases of different agents are pooled together in a principled way. Moreover, the agents can use their case base to inform about a similarity-based likelihood that the suggested component will be adequate for the current problem. We illustrate CoopCA by applying it to the task of agent team formation[1].

1 Introduction

We present a form of case-based reuse conducive to the cooperation of multiple CBR agents in problem solving. First, we present a form of constructive adaptation for configuration tasks with compositional cases. Constructive adaptation is composed of the Hypotheses Generation and the Hypotheses Ordering processes. Then we introduce CoopCA, a multi-agent constructive adaptation technique for case reuse, showing how Hypotheses Generation can be extended to a multi-agent system. The agents suggest possible components to be added to the ongoing configuration problem, allowing an open, distributed process where components used in cases of different agents are pooled together in a principled way. Moreover, the agents can use their case base to inform about a similarity-based likelihood that the suggested component will be adequate for the current

[1] Thanks to David Aha who during ECCBR-2004 in Madrid observed that distributed and multi-agent approached to CBR focused on the Retrieve process and wondered aloud why Reuse process had not been extended to cover distributed and multi-agent scenarios. This work has been partially supported by the CBR-ProMusic project (IC2003-07776-C02-02) and the SAMAP project (TIC2002-04146-C05-01)

problem. This information is used by the Hypotheses Ordering process that thus explores the configuration space guided by the cases of all the involved agents. We illustrate CoopCA proposal by applying this technique to the task of agent team formation.

1.1 Generative Reuse with Constructive Adaptation

The Reuse process in case-based reasoning when the solution is a complex structure, like a plan or a design, can be performed in two ways: techniques for reuse on synthetic tasks, they fall into two families: transformational reuse and generative reuse [1]. *Transformational Reuse* takes the solution structure of one (or several) retrieved case(s) and *transforms* that structure using some specific algorithm or search process until a new solution structure is found that is "consistent" (or "adequate") with the new problem [5, 6]. *Generative Reuse*, on the other hand, generates the new solution *by construction*; the generative process uses past cases (and their similarity to the new case) to construct the solution structure of the new case. The canonical technique in CBR literature for planning tasks is derivational analogy [8].

Constructive adaptation (CA), as presented in [7], is a general technique for generative reuse based on two basic notions: a) that building a solution for a new case is a search process, and b) that the search process is guided by a similarity measure over the precedent cases stored in a case base. The third basic idea of CA is that there are two related but distinct tiers of representation, namely case representation (useful to compute similarities) and state representations (useful to perform search process). Therefore, CA proposed a two-tiered process for constructive adaptation, as shown in Fig. 1. Notice that CA fulfills the requirements for being a form of *compositional adaptation* [10, 9] since CA is reuse technique where solution parts coming from multiple cases are reused and combined together.

This paper presents a more specific version of the Constructive Adaptation approach adequate for design and configuration tasks. Although less general than [7], this specialization allows us to define CA in a more detailed and formal manner. The contribution of this paper is three fold. First, we present an abstract and formal specification of *Compositional Cases*, a case description formalism that is adequate for CBR systems dealing with design and configuration tasks. The second contribution is a detailed description of Constructive Adaptation for design and configuration tasks using Compositional Cases;

2 Constructive Adaptation for Compositional Cases

Configuration tasks are amenable to be supported in CBR systems using compositional cases— i.e. cases that express the relation between a *component* and the *role* it plays in the object being configured. For instance, in the task of configuring a PC, *HardDrive* is a role and *ATA/IBM-DJSA-210* is a component that can fulfill that role.

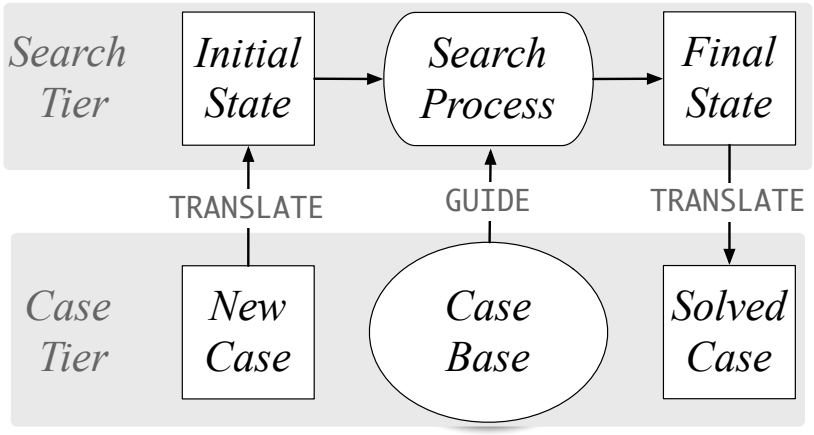

Fig. 1. The two-tiered process of constructive adaptation.

Constructive adaptation is composed of the *Hypotheses Generation* (HG) and the *Hypotheses Ordering* (HO) processes. Both HG and HO work upon states (see Fig. 1) representing a partial configuration being considered by the system. Concerning compositional cases, HG takes a state and generates new states with refined partial configurations; specifically it takes a open role R_i and generates a new state for each particular component C_j that can fulfill role R_i. Components C_j are obtained by retrieval of the configurations in case-base with a role R_i.

Concerning Hypotheses Ordering, HO orders open states assessing the similarity of the state's partial configuration with respect to the case base of configurations. Specifically, let C_j in role R_i be the last component added to a state; HO will give the state a rank value that is the highest similarity of a case with C_j in role R_i with respect to the current problem. Notice that we are assessing similarity comparing the problem specification and not the solutions (the configuration of the cases and the partial configuration of the state).

The only requirement to use CoopCA in a configuration task is that the CBR system has to be able to describe the characteristics that specify the possible components that may fill a role. We will call this description as a *task specification* (or simply a *task*) of a role[2]. As we will see, task specification is the means used to inform other agents about the current focus of interest in the reuse process. Then, the agents receiving a task specification can use it as a query to their case base —and from which the components satisfying it are retrieved.

[2] The simplest task specification is just a role name R_i; however, often some constraints on the type and/or properties of components that can legally fill the role are expressed in the task specification

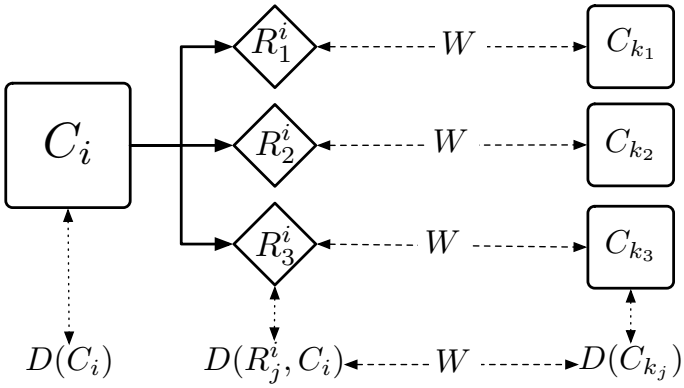

Fig. 2. Compositional cases consist of a complex component C_i that specifies the roles for the required subcomponents (R_j^i) and the bindings (W) of those roles with further components (C_{k_j}). Compositional cases and roles have descriptions (D) used to establish valid matchings in the application domain.

2.1 A framework for compositional cases

This section develops a domain-independent description framework for case-based compositional design (or configuration, in the following we will use both terms synonymously).

First, we will define a language $\mathbb{L} = \langle \mathcal{R}, \mathcal{T}, \mathcal{C}, \mathbb{O} \rangle$ for compositional design, where $\mathcal{R}$ is the set of roles, $\mathcal{T}$ is the set of *tasks*, $\mathcal{C}$ is the set of components, and $\mathbb{O}$ is an object language with a subsumption ($\sqsubseteq$) relation[3] used to describe tasks and components. Moreover, a task $T \in \mathcal{T}$ is a triple $T = \langle R, C, D(R, C) \rangle$ where R is a role in component C and $D(R, C)$ is a description of the characteristics that specify the possible components that may fill role R. We will use the dot notation to refer to an element of a tuple, e.g. $T_i.R$ denotes the role of task T_i.

We will distinguish two types of components ($\mathcal{C} = \mathcal{C}_E \cup \mathcal{C}_X$), namely elementary and complex components. A complex component $C \in \mathcal{C}_X$ is a pair $C = \langle D(C), \{(R_i, D(R_i, C))\}_{i=1...n} \rangle$ where the head $D(C)$ is a description of the component in the object language $\mathbb{O}$ and the *tail* is the collection of roles (and their descriptions) required by C; an elementary component is simply one that requires no further roles. A complex component defines a collection of new (sub)tasks that we note $Tasks(C) = \{T_i | (R_i, D(R_i, C)) \in Tail(C)\}$.

Component matching ($T_i \preceq C_j$) is a relation that establishes whether a component C_j can fulfill a role R_i by checking that if satisfies the associated task description— i.e. $T_i \preceq C_j = D(R_i, C') \preceq_{\mathbb{O}} D(C_j)$. Since both descriptions,

[3] Subsumption is the inverse relation to satisfaction: given two formulae $\psi, \psi' \in \mathbb{O}$ that ψ *subsumes* ψ' ($\psi \sqsubseteq \psi'$) if all that is true for ψ is also true for ψ' (or that ψ' satisfies ψ).

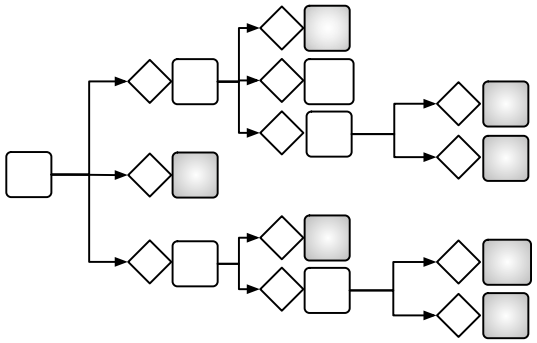

Fig. 3. Compositional cases represent a configuration that is complete and valid; this figure shows a configuration with 12 roles that is complete (since all roles are bound). Elementary components are shown as gray boxes.

$D(R_i, C')$ and $D(C_j)$, are expressed in the object language $\mathbb{O}$ the relation $\preceq_{\mathbb{O}}$ also depends on the object language.

Definition 1. ***(Binding)*** *A* binding $W_k = (T_i \doteq C_j)$ *is the assignment of a specific component* C_j *to a particular role* $T_i.R$ *of a component* $T_i.C$.

We note $\mathcal{W}$ as the set of all possible bindings in a language $\mathbb{L}$ and $W_k.T$ (resp. $W_k.C$) the task (resp. component) of a binding W_k. A binding $W_k = (T_i \doteq C_j)$ is *legal* when their elements satisfy the component matching relation $T_i \preceq C_j$.

Definition 2. ***(Configuration)*** *A* configuration $K \in \mathcal{K}$ *is a collection of bindings* $K = \{W_k = (T_i \doteq C_j)\}_{k=1...m}$. *If all bindings are* legal *we say the configuration* K *is* valid.

A configuration may be partial or complete: intuitively a configuration is complete when every task has an assigned component; otherwise it's partial. Let us note the set of components in a configuration $K.C = \{C_j | \exists B_k \in K : B_k.C = C_j\}$ and the set of tasks $K.T = \{T_i | \exists C_j \in K.C : T_i \in Tasks(C_j)\}$.

Definition 3. ***(Complete Configuration)*** *A configuration* K *is complete iff* $Complete(K) = \forall T_i \in K.T : \exists B_k \in K : B_k.T = T_i$; *otherwise* K *is partial.*

We can now define a *composite case*, and for that purpose we will assume that a problem specification (or *query*) Q is a special type of task $Q = \langle -, -, D(Q) \rangle$, namely one that has no role or component but only a description $D(Q)$ in the object language $\mathbb{O}$ specifying the requirements that a solution to the problem has to satisfy.

Definition 4. ***(Composite Case)*** *A composite case is a pair* (Q, K) *where* $(Q \doteq C_j) \in K$ *and* K *is both valid and complete.*

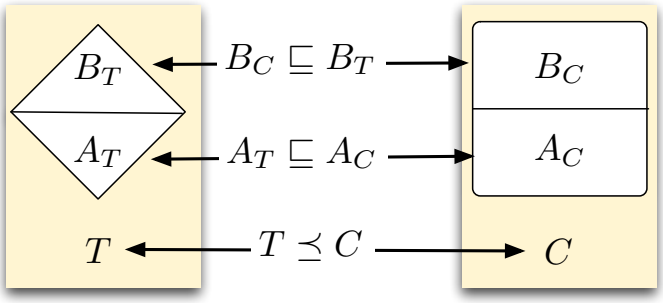

Fig. 4. Matching $T \preceq C$ between a task (role or user query) T and a component C is valid when their *Before-formulae* and *After-formulae* satisfy the plug-in matching criteria.

Notice that the above definitions of a complete and valid configuration K does not imply that is satisfies the requirements put forward by Q. In fact, this is the information provided by a (correct) case base; that is to say, a case (Q_r, K_t) states that it is known that K_t is an adequate solution for Q_r. As we will see in the following sections, this is the information source that will be used by the process of constructive adaptation.

2.2 Compositional design specialized descriptions

In order to specialize the general description of constructive adaptation (CA) to the task of compositional design we will make a further assumption concerning the descriptions of tasks and components. We will assume form now on that tasks and components are described as pairs (B, A) — where B are the *Before-formulae* (or *preconditions*) and A are the *After-formulae* (or *goals*) that characterize what they assume to be true in the world *before* and *after* they are used inside a configuration.

Concerning a component C with $D(C) = (B_C, A_C)$, A_C is a collection of formulae in language $\mathbb{O}$ that express what is true after a component is used for some role with a legal binding, while B_C express what C assumes to be true in the designed configuration (and should be provided by some other component in the configuration in order to insure that C fulfills its role).

Concerning a task $T = \langle R, C, D(R, C)\rangle$ with $D(R, C) = (B_T, A_T)$, A_T is a collection of formulae in language $\mathbb{O}$ that express what needs to be true after whatever component has been chosen to fulfill role R in C, while B_T express that which any component fulfilling role R of C can assume to be true.

Now, since a problem query Q is also a task, it will be a tuple ($Q = \langle -, -, (B_Q, A_Q)\rangle$), where $Q.D$ is a pair of B- and A-formulae. Notice that the interpretation of the task induced by a query is the following: A_Q are the goals that the configured design has to satisfy and B_Q specifies the statements that can be assumed to be true by the configured design.

Component matching may now also be specialized to this description framework. Since we had matching as $T_i \preceq C_j = D(R_i, C') \preceq_{\mathbb{O}} D(C_j)$, now we have that $T_i \preceq C_j = (B_T, A_T) \preceq_{\mathbb{O}} (B_C, A_C)$ can be defined over B-formulae and A-formulae. Adopting the usual notion of matching from software components literature (often called *plug-in matching*) we have that

$$T_i \preceq C_j = (B_T \sqsupseteq B_C) \wedge (A_T \sqsubseteq A_C) \tag{1}$$

that is to say, $(A_T \sqsubseteq A_C)$ the component's A-formulae satisfy all task's A-formulae (all of task's goals are achieved by the component) and $(B_T \sqsupseteq B_C)$ the task's preconditions satisfy all component's preconditions (i.e. a component can achieve the same goals with less stringent preconditions).

2.3 Compositional design constructive adaptation

We will present now the search process of constructive adaptation for compositional design. For this purpose, we will define what a state is, how states are generated (Hypotheses Generation) and how to select the state to be expanded (Hypotheses Ordering).

Definition 5. ***(State)*** *A state Z given a query Q is a tuple $Z(Q) = \langle B^{\perp}, A^{\perp}, B^{\top}, A^{\top}, W^{\perp}, W^{\top}, W^{H} \rangle$, where*

1. *$B^{\perp}$ and $A^{\perp}$ are open B- and A-formulae, i.e. those not satisfied in Z*
2. *$B^{\top}$ and $A^{\top}$ are closed B- and A-formulae, i.e. those satisfied in Z*
3. *$W^{\perp}$ is the set of open bindings (those tasks that are not bound to any component in Z), $W^{\top}$ is the set of closed bindings (those tasks already bound to a component in Z), and W^{H} is the last binding (that introduced in the predecessor state of Z)*

A state Z is valid *when all bindings in $Z.W^{\top}$ are valid.*

As we have seen, constructive adaptation is a two-tiered process where case-based problem solving works both at the case representation tier and the state representation tier. Therefore, we will need some mapping functions that both tiers. The first function *initial state* ($\texttt{IS} : \mathcal{Q} \rightarrow \mathcal{Z}$) that transforms a query Q into an (initial) state $Z_0(Q)$, as follows

$$\texttt{IS}(Q) = \langle \emptyset, Q.D.A, Q.D.B, \emptyset, T_Q, \emptyset, \emptyset \rangle$$

that is to say, a state where the Q's B-formulae become *closed preconditions*, A-formulae become *open goals*, there are no closed bindings except for the open binding representing the query task itself.

Hypothesis Generation The *Hypothesis Generation* function ($\texttt{HG} : \mathcal{Z} \rightarrow 2^{\mathcal{Z}}$) generates the successor states of a state Z_i in three steps: 1) an open task is selected, 2) the components that match that task are gathered, and 3) a successor state is generated for each of the components that can be bound to the task. Specifically:

1. (**Open Task Selection**) `HG` takes a task from the state's open bindings $T^j_{Z_i} \in Z_i.W^\perp$. This selection is random since is no reason to order the open tasks: once an open task is introduced in a configuration it has to be solved by finding an adequate component; thus, there is no speed up to be gained in ordering them since solving one task before another makes no difference.
2. (**Component Gathering**) `HG` gathers the set of components that match this task: $C(T^j_{Z_i}) = \{C | T^j_{Z_i} \preceq C\}$. This gathering can performed in two ways:
   (a) (*Catalog Component Gathering*) If all components descriptions are placed in repository then we only have to check for those components in the catalog that satisfactorily match the task description $C(T^j_{Z_i})$. This approach is adequate when all information on components (a Catalog) is directly available.
   (b) (*Case-based Component Gathering*) If the component descriptions available are those used in previous configurations stored as cases then a CBR system withe a retrieval technique supporting subsumption ($\sqsubseteq$) can infer which components will match the selected task (since we have already defined $\preceq$ in terms of $\sqsubseteq$ in (1).
3. (**Successor States**) `HG` generates a new *successor state* for each component $C^k \in C(T^j_{Z_i})$ as follows

$$succ(Z_i, T^j_{Z_i}, C^k) = \langle B^\perp, A^\perp, B^\top, A^\top, W^\perp, W^\top, W^H \rangle$$

   where $T^j_{Z_i}$ is no longer an open task in $W^\perp$ and a new binding $T^j_{Z_i} \doteq C^k$ has been added to $W^\top$. The Appendix A formally describes how the new state is generated. Essentially the new component C^k achieves some new goals not yet achieved in Z_i and therefore $A^\top$ and $A^\perp$ are updated accordingly. Moreover, if C^k has subtasks they are added to $W^\perp$ and since each subtask introduces A-formulae and B-formulae again $A^\top$ and $A^\perp$ need to be updated accordingly.

Hypothesis Ordering The essential notion of constructive adaptation is to use cases similar to the current problem to guide the search process. Since in compositional design a problem query $Q = (B_Q, A_Q)$ is a specification of the properties desired for the solution (A_Q) plus the assumptions of what can be assumed to be true (B_Q), we need a similarity relation S between Q and the problem specification part of composite cases (Q_i, K_i). The relation S provides thus a ranking $\mathbf{S}$ of the cases in the case base $\Sigma = \{(Q_i, K_i)\}_{i=1...N}$

$$\mathbf{S}(Q, \Sigma) = \{\langle (Q_i, K_i), S(Q, Q_i) \rangle\}_{i=1...N}$$

Next, we have to transform this case ranking into a ranking of the open states $\mathcal{Z}^{open}_t(Q) \subset \mathcal{Z}$ at a step t in the CA process.

For this purpose, consider the latest hypothesis to which the CA process is committed to in an open state, namely $Z.W^H$ the last component added to

the configuration. Since CA will pick the highest ranking state in $\mathcal{Z}_t^{open}(Q)$ to expand (generating successor states) we are interested into assessing how likely that the partial configuration of an open state Z is to lead to a correct solution. Since comparing the whole structure of the of the partial configuration with the case base would be excessively time consuming, CA will assess this likelihood by considering only the latest hypothesis $Z.W^H$ of each open state.

Let us call note $Z.W^H.C$ the component C_j bound by the latest hypothesis $Z.W^H = (T_i, C_j)$ in state Z and let be $\Sigma|_{(T_i,C_j)} \subset \Sigma$ the subset of cases in the case base where the component C_j fills role T_i. Since the similarity relation S induces also a ranking of the cases in this subset $\mathbf{S}(Q, \Sigma|_{(T_i,C_j)})$ we can now define the function M that yields the similarity degree of highest ranking case in $\Sigma|_{(T_i,C_j)}$, namely

$$M(\Sigma, T_i, C_j) = max(\{S(Q, Q_k)|(Q_k, K_k) \in \Sigma|_{(T_i,C_j)}\}) \tag{2}$$

The ranking relation $\mathbf{R}$ induces a ranking over the open states by computing a heuristic value r_i for each open state Z_i:

$$\mathbf{R}(\mathcal{Z}_t^{open}(Q)) = \{\langle Z_i, r_i\rangle\} = \{\langle Z_i, M(\Sigma, Z_i.W^H.T, Z_i.W^H.C)\}_{Z_i \in \mathcal{Z}_t^{open}(Q)\rangle}$$

that is to say, each open node Z_i is ranked according to the degree of similarity of the highest ranking case that has current hypothesis component $Z_i.W^H.C$ fulfilling $Z_i.W^H.T$, the current hypothesis role.

Goal Test The last element of constructive adaptation is the *Goal Test* function $\texttt{GT}$: $\mathcal{Z} \rightarrow \{True, False\}$. Goal Test checks whether a state Z is solution, i.e. whether the state corresponds to a valid solution that satisfies the problem query Q:

$$\texttt{GT}(Z, Q) = Valid(Z) \wedge Satisfies(Z, Q) = (Z.W^{\perp} = \emptyset) \wedge (Z.A^{\perp} = \emptyset) \wedge (Z.B^{\perp} = \emptyset)$$

namely, there is no task not bound to a component ($Z.W^{\perp} = \emptyset$), all $Q.A$ goals are satisfied ($Z.A^{\perp} = \emptyset$), and there is no B-formula required by a configures component that is not satisfied ($Z.B^{\perp} = \emptyset$).

Notice that in CA the similarity relation S is left open and may vary across different application domains and representation languages used to specify the problem query Q. In our experiments we have used SHAUD, a similarity relation for feature terms [2], but other similarity relations can be used — e.g. RIBL-2 for Horn clause representation[3].

3 Multi-Agent Cooperative Constructive Adaptation

In this section we will present a distributed framework for case reuse using constructive adaptation. For this purpose we will focus on a particular application that is essentially distributed: team formation of cooperative agents. Team formation is the process by which, given a task to be achieved specified by a user

or an agent, such that no single agent is capable of achieving it, a selection of agents with the required capabilities is made and then organized as a multi-agent system with the required interactions protocols to coordinate those agents.

The reason we focus of team formation is that a distributed form of the case reuse process only makes sense if the *knowledge* used for reuse is itself *distributed*. Multi-agent systems can be characterized precisely by this fact: there is no central repository of containing the information and knowledge. In other words, each agent has a local view of the problem solving episodes in which it is involved, and each agent has its specific capabilities (and its particular knowledge). Clearly, if a task can be performed using the capabilities of a single agent there is no need to form a term.

We will now present the cooperative constructive adaptation (CoopCA) technique in the framework of agent team formation. CoopCA assumes that the agents are willing to cooperate in forming a team and sharing the necessary information. The next subsections will first express the concept of team as a compositional case and later will present the distributed reuse technique of CoopCA.

3.1 Teams as compositional cases

Agent teams can be modeled as compositional cases; in fact, the ORCAS framework [4] represents agent teams as compositional cases and uses case-base reasoning to form teams adequate for specific tasks. Recall the general schema of compositional cases in Fig. 2: an agent team fills this schema if we interpret components C_i as agents, roles R^i_j as subtasks, component descriptions $D(C_i)$ as descriptions of *agent capabilities*, and role descriptions $D(R^i_j, C_i)$ as descriptions of agents subtasks as shown in Fig. 5 . A team is formed when the bindings W are established associating to each role/subtask an agent with a capability suitable to achieving that task. We say an agent Ci plays role R^i_j in a team when such a binding W exists, and agent C_i acts as *coordinator* of the agents that play the roles defined by C_i. Moreover, agent C_i can either solve role/subtask R^i_j either alone or defines further subtasks (that will be achieved by a subteam of which C_i is the coordinator).

We will view the process of team formation as a *compositional design* task. Let us assume there is a user that poses a query Q to a *broker*, i.e. an agent that will be in charge of designing the team structure, then negotiate the specific agents that will fulfill the team roles, and finally setting up the interaction protocols for the agent team. In this paper we will deal mainly with the first stage, namely designing the team structure, although some issues on selecting agents will also be discussed. Concerning the second and third stages, no CBR is used there, but see [4] for details. Now, the current approach in multi-agent systems (MAS) is to assume there is one or several "yellow pages" services where agents register their capabilities.

Our model, however, will be to use an experience-based approach. Specifically, we assume that the broker agent that uses CBR on a case base composed of previous teams; the broker will try to use CoopCA to form new teams by reusing old teams in its case base. In fact, this two approaches to find adequate agents as

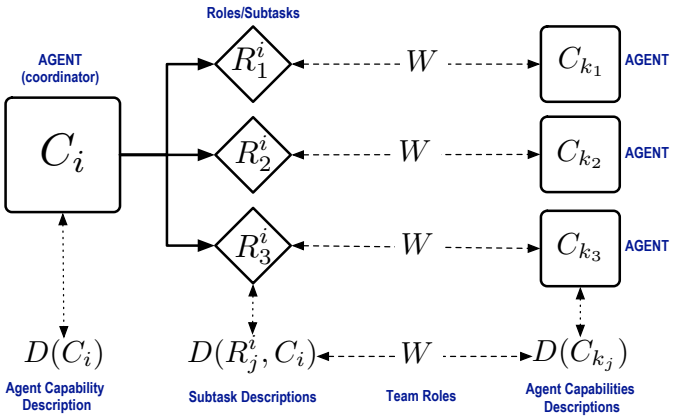

Fig. 5. Team as a compositional case.

team components are called in Section 2.3 *Catalog Component Gathering* (since yellow pages is a catalog of agent capabilities) and *Case-based Component Gathering* (since the new team will have capabilities used in past teams stored in the case base). Moreover, these two approaches are not incompatible: the broker may resort to use the yellow pages catalog if need be.

Finally, notice that what we call *broker agent* is in fact a role; that is to say, there is no such a thing as *the* broker but a number of agents that have played the *broker role* in forming new teams. Therefore, there is no unique and centralized repository of cases describing teams; instead, we have that agents playing the role of brokers have individual case bases storing their experience in team formation. We can now see that the knowledge for team formation is essentially distributed and thus the *reuse process* that CoopCA embodies should be such that makes use to this distributed knowledge as far as possible. In what follows, we will call a description of a team stored as a case in the case base of an individual agent a *team-case*.

3.2 Cooperative Constructive Adaptation

The assumptions made by CoopCA are the following: 1) there is a collection of agents $\mathcal{B}^l(a_b)$ that played the role broker and store their team designs on an individual case base; and 2) there is an *acquaintance relation* $\mathbb{A}(a_b, a_k)$ among the agents in $\mathcal{B}^l(a_b)$ such that for an agent a_b then $\forall a_k \in \mathcal{B}^l(a_b) : \mathbb{A}^l(a_b, a_k)$; i.e. either $\mathbb{A}(a_b, a_k)$ (a_k is an acquaintance of a_b or there is a chain of acquaintances of length not larger that l that links a_b and a_k). When an agent in $\mathcal{B}^l(a_b)$ receives a query Q to form a team it will use CoopCA to design such a team using the collective experience in team formation of the agents in $\mathcal{B}^l(a_b)$. Moreover, notice that in a MAS framework there are several agents that can possess the same capability and, thus, the same *team design* can be realized by different

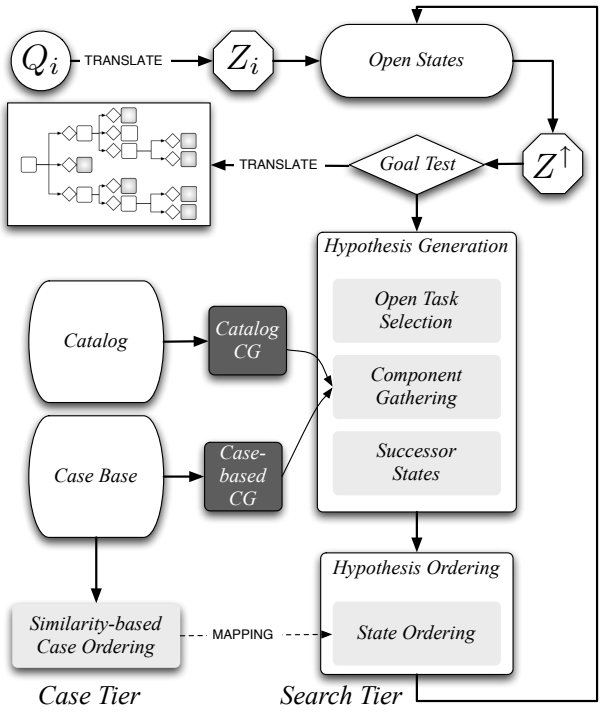

Fig. 6. Constructive adaptation two-tiered process.

collections of agents. We assume in the following that the capabilities are the *components* of CoopCA for team design.

CoopCA follows basically the CA search process described in section 2.3 and summarized in Fig. 6 with a few modifications. The broker agent that receives the query Q will perform the CA search process (i.e. it will generate new states and maintain the open and closed states) but it will need the help of other agents to generate the hypothesis and to rank them. In other words, *Hypothesis Generation* and *Hypothesis Ordering* will require the broker to communicate with other agents and use the acquired information to generate and order the hypothesis during search. Let us first consider Hypothesis Generation and later we will turn to Hypothesis Ordering.

Cooperative Hypothesis Generation. The cooperative hypothesis generation function will generate the successor states with the help of other agents in $\mathcal{B}^l(a_b)$. For this purpose we need to modify the second step (*Component Gathering*); steps *Open Task Selection* and *Successor States* are not modified. We will use a distributed form of *Case-based Component Gathering* such that an broker agent a_b will send a message (containing a task description) to his acquaintance agents $\mathbb{A}(a_b, a_k)$; in turn they will send this message to their acquaintance agents up to l times. Those agents in $\mathcal{B}^l(a_b)$ that receive the message and find some

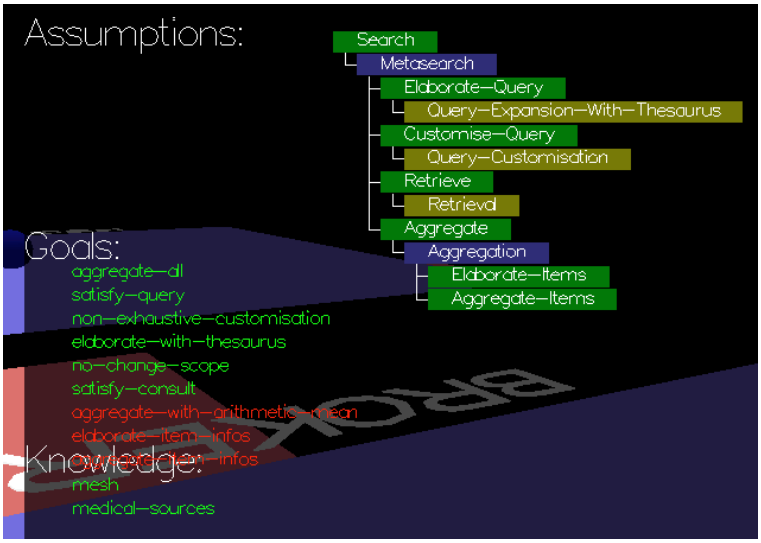

Fig. 7. Image capture of the CBR broker in ORCAS.

component in their case bases that satisfy that task description will answer to the broker a_b with a message containing the component(s) and a degree of similarity.

More formally, the broker agent interacts with its acquaintance agents to obtain the information concerning a specific task in the following steps:

(1) Start Cooperation: First the broker agent a_b informs its acquaintance agents of the task to be performed and sends a message m_1 containing the query $Q = (B_Q, A_Q)$. This information is forwarded by a_b's acquaintances to their respective acquaintances until all agents in $\mathcal{B}^l(a_b)$ has this information. Notice that Q will be used by the agents to compute the similarity using definition (2). The agents are of course free to decline to cooperate, so only those that send back an acceptance message before a time τ_{m_1} will be considered in the following steps as members of the multi-agent system $\mathcal{A}$.

(2) Current Task: Let T^j be the current task given by *Open Task Selection*. The broker agent broadcasts a message m_2 containing the task description T^j to the agents in $\mathcal{A}$ and waits for their answers before time τ_{m_2}. Figure 7 shows a snapshot of the visualization tool of the ORCAS platform with the current state of a CBR broker agent using constructive adaptation; notice the goals on the left (the first 6 achieved and the last 3 still open) and the task/capability bindings of the current sate on the right.

(3) Available Capabilities: Every agent $a_k \in \mathcal{A}$ will receive message m_2 and execute the *Case-based Component Gathering* process over their case base retrieving a set of components $C^k(T^j)$ that match the task description T^j. Notice that a retrieved "component" is in fact a specific agent that uses a capability in some role in some team-case, and what interests the broker a_b are the available

capabilities matching the current task T^j. For each component $C_i \in C^k(T^j)$ agent a_k will send a message m_3 with a tuple $\langle T, C, A, M\rangle$ containing:

$$\langle T^j, C_i, \mathcal{A}_k(C_i), M_k(C_i)\rangle$$

where C_i is a capability adequate for task T^j, $\mathcal{A}_k(C_i)$ is the set of agents known to a_k that possess capability C_i, and $M_k(C_i)$ is the maximum degree of similarity —defined in *Hypothesis Ordering* in equation (2).

(4) Hypothesis Generation: Now the broker a_b has received a set of messages $\mathcal{M}_3$ of type m_3, from which it knows the set of available capabilities $C^k(T^j) = \bigcup_{m\in\mathcal{M}_3} m.C$. Then a new successor state is generated for each capability $C_i \in C^k(T^j)$, and this ends the Hypothesis Generation precess. Moreover a_b also builds a list of available agents $\mathcal{A}(C_i) = \bigcup_{m\in\mathcal{M}_3(C_i)} m.A$ for each capability $c \in C^k(T^j)$ from the relevant messages $\mathcal{M}_3(C_i) = \{m \in \mathcal{M}_3 | C_i = m.C\}$.

(5) Hypothesis Ordering: Since $\mathcal{M}_3$ also contains the similarity information needed for Hypothesis Ordering, the broker simply has to aggregate the values coming from different agents for each capability $c \in C^k(T^j)$. For our purposes, the maximum similarity value is a good option so a capability c will have as similarity value $M(\mathcal{A}, T^j, C_i) = max_{m\in\mathcal{M}_3(C_i)} m.M$. This value allows the broker to use the ranking relation $\mathbf{R}$ to order the open states in the constructive adaptation process. Once *Hypothesis Ordering* finishes, then either the search is terminated by the *Goal Test* and the next step is (6), or it goes to step (2).

(6) Agent Selection and Instruction: When the team design is finished the broker a_b has a complete specification of the hierarchical team structure. This last step consists of selecting for each particular task/capability ($T^j \doteq C_i$) binding an agent with capability C_i, i.e. one of the set $\mathcal{A}(C_i)$ computed in step (3). We will not go into the details of this process (explained in [4]), suffice to say the broker has to negotiate with the candidate agents and select a crew to fully staff the team and then provide the selected agents with the instructions on how to coordinate to achieve the global task.

We have seen that CoopCA is a straightforward extension of constructive adaptation for compositional cases for multi-agent scenarios. CoopCA is in fact applicable to scenarios where the knowledge exploited in the Reuse process is in some way distributed over a collection of entities, e.g. web services.

4 Conclusion

This paper discusses three different but related issues relevant to case-based reasoning. First, the definition of compositional cases as a useful abstraction for a wide variety of CBR applications in design and configuration tasks; compositional cases are however limited to tasks where the designed structure is hierarchical. Second, constructive adaptation (CA) for compositional cases specializes the basic ideas of CA [7] in a generic reuse algorithm that is valid for any CBR system that espouses compositional cases for a task; CA for compositional cases leaves open which representation language $\mathbb{O}$ is used for B-formulae and A-formulae. The language $\mathbb{O}$ can be anyone (from Horn clauses to description logics to simple concept taxonomies) adequate for a specific domain as long as

the operation of subsumption (or satisfaction) is supported and some relational case similarity (e.g. SHAUD [2] in ORCAS or RIBL-2 [3] for Horn clauses) is defined.

Third, we have shown that CA can be extended in a natural way to a distributed design task, and we have focused on applying CoopCA to the task of agent team formation. CoopCA shows the power of applying CBR to multi-agent systems tasks such as team formation. Often in MAS agents are supposed to be capable of reasoning and learning but they rarely are on practice. Let's think about the *yellow pages* approach to team formation: agents are assumed to go to the yellow pages every time a new team is to be formed: this is just because agents are assumed not to learn. We have shown that learning team-cases a broker agent will not need to repeat needles work for every team it forms. Once a broker a_b has formed a team for task T^i and a new task T^j similar to T^i arrives, a_b already knows most of the components (agents and their capabilities) that most likely will be in the new team. Thus learning cases decreases not only search costs but also communication costs among agents. In fact, since in a given environment most tasks tend to be repetitive, CoopCA shows that the CBR approach offers a straightforward way to form teams efficiently.

The CoopCA is now simply using a best-first search regime with similarity-based heuristic; as future work we want to use more powerful (satisficing) search regimes that would allow also to minimize solution costs.

A Successor

The *successor* function generates a new state Z_j given a current state

$$Z_i = \langle B_i^{\perp}, A_i^{\perp}, B_i^{\top}, A_i^{\top}, W_i^{\perp}, W_i^{\top}, W_i^{H} \rangle$$

for a new component C^k filling a role $T_{Z_i} \in W_i^{\perp}$ as follows

$$Z_j = succ(Z_i, T_{Z_i}^j, C^k) = \langle B_j^{\perp}, A_j^{\perp}, B_j^{\top}, A_j^{\top}, W_j^{\perp}, W_j^{\top}, W_j^{H} \rangle$$

where

(Current Hypothesis) $W^H = (T_{Z_i}^j \doteq C^k)$
(Closed Bindings) $W_j^{\top} = W_i^{\top} \cup (T_{Z_i}^j \doteq C^k)$
(Open Taks) $W_j^{\perp} = W_i^{\perp} \cup Tasks(C^k)$
(Open A-formulae) $A_j^{\perp} = \{x \in A_i^{\perp} | \nexists y \in C^k.A : y \sqsubseteq x\}$
The open A-formulae $A_j^{\perp}$ are those in the previous state $A_i^{\perp}$ such that are not satisfied ($\sqsubseteq$) by a formula in the postconditions of the new component $C^k.A$
(Closed A-formulae) $A_j^{\top} = A_i^{\top} \cup \{x \in C^k.A | \nexists y \in A_i^{\top} : y \sqsubseteq x\}$
The closed A-formulae are those in the previous state $A_i^{\top}$ or in the post-conditions of the new component ($C^k.A$) that are not satisfied ($\sqsubseteq$) by any formula in $A_i^{\top}$.

(Closed B-formulae) $B_j^{\top} = B_i^{\top} \cup \{x \in B_i^{\perp} | \exists y \in C^k.A : x \sqsubseteq y\}$
A new formula is added to closed B-formulae when a formula in open B-formulae $B_i^{\perp}$ is satisfied ($\sqsubseteq$) by a formula in the goals of the new component $C^k.A$.

(Open B-formulae) $B_j^{\perp} = \{x \in C^k.B | \nexists y \in B^{\top} : y \sqsubseteq x\} \cup \{x \in B_i^{\perp} | \nexists y \in C^k.A : x \sqsubseteq y\}$
A new formula is added to open A-formulae when the new component C^k has some formula as a precondition $C^k.B$ that is not satisfied ($\sqsubseteq$) by any formula in the closed B-formulae $B^{\top}$; moreover when an open B-formula becomes closed it is removed.

Essentially the process is the following. Since a component satisfies a task if and only they satisfy the component matching, there is no need for adding the goals of a component subtasks to the open A-formulas. Whichever component will be bound to it will satisfy them and then we will have to remove them to closed A-formulas. We only have to check the goals of the query problem: how introducing a new component satisfies one or more general goal. Dually, when a component is introduced we have to check if some of its preconditions are not satisfied by the problem query B-formulas (or by some component's goals).

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. Eva Armengol and Enric Plaza. Relational case-based reasoning for carcinogenic activity prediction. *Artificial Intelligence Review*, 20:121–141, 2003.
3. W. Emde and D. Wettschereck. Relational instance-based learning. In Lorenza Saitta, editor, *Machine Learning - Proc. 13th Int. Conf. Machine Learning*, pages 122 – 130. Morgan Kaufmann, 1996.
4. Mario Gómez and Enric Plaza. ORCAS: Open, reusable and configurable multi-agent systems. In *Proc. Third International Joint Conference in Autonomous Agents and Multiagent Systems*, pages 144–152. ACM Press, 2004.
5. K. J. Hammond. *Case-based Planning*. Academic Press, 1989.
6. T. Heinrich and J. L. Kolodner. The roles of adaptation in case-based design. In *Proc. AAAI Worksop on Case-based Reasoning*. AAAI, 1991.
7. Enric Plaza and Josep-Lluís Arcos. Constructive adaptation. In *Advances in Case-Based Reasoning*, volume 2416 of *Lecture Notes in Artificial Intelligence*, pages 306–320. Springer Verlag, 2002.
8. Manuela M. Veloso and Jaime G. Carbonell. Derivational analogy in PRODIGY. *Machine Learning*, 10(3):249–278, 1993.
9. 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.
10. W. Wilke, B. Smyth, and P. Cunningham. Using configuration techniques for adaptation. In *Case-based Reasoning Technology*, volume 1400 of *Lecture Notes in Artificial Intelligence*, pages 139–168. Springer Verlag, 1998.