Activating Case-Based Reasoning with Active Databases

Sheng Li and Qiang Yang

School of Computing Science, Simon Fraser University
Burnaby, BC Canada, V5A 1S6

Abstract. Many of today's CBR systems are passive in nature: they require human users to activate them manually and to provide information about the incoming problem explicitly. In this paper, we present an integrated system that combines CBR system with an active database system. Active databases, with the support of active rules, can perform event detecting, condition monitoring, and event handling (action execution) in an automatic manner. The combined `ActiveCBR` system consists of two layers. In the lower layer, the active database is rule-driven; in the higher layer, the result of action execution of active rules is transformed into feature-value pairs required by the CBR subsystem. The layered architecture separates case-based reasoning from complicated rule-based reasoning, and improves the traditional *passive* CBR system with the *active* property. This paper shows how to construct `ActiveCBR` system and provides an analysis of the resulting system architecture.

1 Introduction

As an interactive process, case retrieval in most CBR systems has been based mainly on a user-interaction model. In this model, a user provides all the information necessary for the CBR system to draw a conclusion. This *'passive'* nature of interactive CBR requires the direct involvement of human users in order to provide information about the incoming problem explicitly. This *passive mode* has some obvious drawbacks. The manual operation on information collection cannot handle massive amount of user data and real time events properly. The manual operation on system activation limits the ability to perform reasoning in timely fashion, especially in some emergent applications such as forest fire protection and coastal salvage in which the group profile of events and data trigger the operation of a CBR system.

To solve these problems, we integrate an active database system with a CBR system. An active database system is a database system that monitors situations of interest, and when they occur, triggers an appropriate response in a timely manner. An active rule in active databases extends the expert system rules with the ability of autonomously responding to external events, e.g., the modifications of data table such as *INSERT, DELETE,* and *UPDATE*. It generally follows the *Event-Condition-Action* paradigm.

E. Blanzieri and L. Portinale (Eds.): EWCBR 2000, LNAI 1898, pp. 3–14, 2000.

on *event* # event detecting
if *condition* # condition monitoring
then *action* # action executing

An active database system can complement passive CBR systems in the following aspects:

- The underlying database system is capable of handling massive amount of user data, so after integrating with an active database system, CBR systems need not handle raw data directly.
- The rule mechanism is capable of detecting external events, and transferring the responses to CBR systems.
- Active database systems support complicated database query, so the statistical result of the collection of user data can be obtained.

To combine the two technologies, we propose an `ActiveCBR` architecture that builds a case-based reasoning subsystem on top of an active database, and realizes problem solving based on the data and events in a relational database. The `ActiveCBR` system developed under this architecture consists of two layers. In the lower layer, the ADB subsystem is rule-driven; in the higher-layer CB subsystem, the result of action execution of active rules is transformed into feature-value pairs required for the reasoning procedure. Connected by the rule engine in an active database, we can perform problem solving on large data sources – relational databases. Furthermore, the reasoning procedure is performed reactively in a real-time manner by responding to external events.

We can find many practical applications where the `ActiveCBR` system will provide real-time support for many industrial applications. This is because in many applications, the input data tend to repeat similar patterns from time to time. For instance, in an active database system that monitors travel information, when the season, economy and social factors recur, the interest in travel plan is likely to display regularly repetitive patterns. This observation ensures that it is reasonable to apply case-based reasoning in the problem solving in the real-time active database environment. In this paper, we motivate our research using a realistic Cable-TV diagnosis problem which we have applied our system to. In this application, the Cable-TV symptoms tend to recur with repetitions in season, user group and equipment. We show how to construct an `ActiveCBR` system using an example.

2 `ActiveCBR`: Representation and Algorithms

2.1 Overview of Active Databases

Active database, as a database system with reactive behavior, has been the subject of extensive research recently. The knowledge model of an active database system determines *what* can be said about active rules in the system. In contrast, the execution model indicates *how* a set of rules behaves at runtime. [5] discuss different *sources* of event that describe the happening to be monitored.

The execution of the rules depends on the event-condition and condition-action *coupling modes*, that can be *immediate, deferred* or *detached* [4]. The *transition granularity* describes the relationship between events and the rule execution. It can be *tuple-oriented* or *set-oriented*.

Termination is the key design principle for active rules. Due to the unexpected interactions between rules, termination is difficult to ensure even after a careful design. Triggering graphs are used for reasoning rules about termination. A rule set is *confluence* when any triggering of rules produce a unique final database state independent to the order of execution of the rules. A rule set guarantees *observable determinism* when all visible actions performed by rules are the same for any order of execution of the rules.

Chimera system [3] integrates an object-oriented data model, a declarative query language, and an active rule language. It supports *display*, *generalize*, and *specialize* events in addition to traditional *create*, *delete*, and *modify* primitives to reflect object manipulation.

Generally, the rule execution in active database systems tends to be more difficult to understand and maintain when supporting more facilities. Even in a conservative-designed rule system like Starburst [2], the semantics of rule execution are still quite complex. Rule termination and rule confluence are difficult to realize in a practical design of a large rule set. How to simplify the design and analysis of active rule sets is an important topic in active database research. With the `ActiveCBR` system, much of the user-level semantics are elevated to the CBR level, whereas the efficient rule-triggering mechanism is left to the database level. We will discuss this in detail in the next section.

2.2 Knowledge Representation in the `ActiveCBR` System

The representation of a case has various forms depending on different applications. In the `ActiveCBR` system, we define the case base as:

Definition 1. *A case base in* `ActiveCBR` *system is combination of* $\{\mathcal{C},\ \mathcal{F},\ \mathcal{I}\}$, *where* $\mathcal{C}$, $\mathcal{F}$, *and* $\mathcal{I}$ *are case space, feature space, and index space, respectively.*

We describe the spaces $\{\mathcal{C},\ \mathcal{F},\ \mathcal{I}\}$ in above definition as follow:

- The case space $\mathcal{C} = \{c^m \mid m = 1, \ldots, M\}$ is the set of M case specifications. A specification of a case consists of *Name, Description, Threshold,* and *Solution*.
- The feature space $\mathcal{F} = \{(f_n,\ v_{n,k}) \mid n = 1, \ldots, N;\ k = 1, \ldots, K_n\}$, is the set of feature-value pairs, where N is the total number of features, and K_n is the number of possible values of feature f_n. In the case representation of the `ActiveCBR` system, all the feature values are symbolic. For the features with original numeric context, we transform them into discrete symbolic values.

– The index space $\mathcal{I} = \{\omega(m,n,k) \mid m = 1,2,...,M;\ n = 1,...,N;\ k = 1,...,K_n\}$ is the set of feature-value weights. A weight is a real number between 0 and 1[1]. Therefore, we can consider $\mathcal{I}$ as such a relation:

$$\mathbf{R} : \mathcal{C} \times \mathcal{F} \rightarrow [0,1]$$

Having the definition of the case base, we can represent a case by two parts: the specification part from an element in $\mathcal{C}$ that describes the name, description, threshold, and solution of the case; and the index part from all the elements in $\mathcal{I}$ that related to this case, which describe the similarity property of the case.

Threshold is introduced into the `ActiveCBR` system as a new field in case specification. It represents the minimum score to which the case is detected at runtime and should be fired accordingly. We will discuss the similarity and score computation in the next subsection.

Examples An example case in AI-CBR's travel agents domain[2] is shown in Table 1.

Table 1. A sample case of the travel agents domain

Name	TravelCase31
Description	#245
Threshold	85
Solution	Hotel Golden Coast, Attica

Feature	**Value**	**Weight**
JourneyCode	649	0
Price	$1,000-2,499	80
HolidayType	Recreation	35
NumOfPerson	1-2	70
Region	Germany	75
Transportation	By plane	45
Duration	5-7 days	85
Season	Summer	65
Accommodation	Luxury	70

The travel-agents' case base is used to help travel agents to recommend hotel destination for customers based on their individual interest and requirement. The solution of each case is a hotel destination. The similarity property of the example case is described by the nine feature-value-weight triples. Note that all the feature values are symbolic. Some features, such as *Price*, *Duration*, and *NumOfPerson* have original numeric values, but they have been transformed to symbolic representation. In this case base, each feature has only one value with

[1] In the internal representation of the `ActiveCBR` system, the weight are converted to integers between 0 and 100, and the threshold in case specifications is defined as an integer between 0 and 100 as well.

[2] The travel agents case base is downloaded from AI-CBR's case-base archive. (*http://www.ai-cbr.org/cases.html*)

a positive weight. Feature *JourneyCode* is used for indexing purpose only and no positive weight is assigned. (We have omitted other feature-value pairs with zero weight in the table.)

Another example case is from a cable TV domain used by a cable TV company (shown in Table 2). In this domain, a case can have multiple positive weights for different values on a particular feature. For instance, both values *'no picture'* and *'reception'* of the feature *'ProblemType'* are related to the case *'Regional switch (LB) problem'*, but the former has a higher possibility, so it is assigned with a higher weight. We will further discuss the meaning and usage of the feature weights in Section 3.

Table 2. A sample case of the cable TV domain

Name:	Regional switch (LB) problem
Description:	Low band regional switch is breakdown
Threshold:	78
Solution:	Generate ticket for technician

Feature	**Value**	**Weight**
ProblemType	no picture	75
	reception	65
	VCR problem	0
Channels	lower band	80
	upper band	0
	US channel	0
Duration	recent 24 hrs	70
	recent 1 week	45
	not specified	0
Location	particular	85
	not specified	0

Rule Representation The representation of an active rule in the ADB subsystem depends on its underlying RDBMS. Both Oracle and SQL Server use triggers to perform the rule mechanism. A trigger in ADB subsystem is a special kind of stored procedure that is executed automatically when the specified data-modification occurs on the specific table. One trigger can contain one rule or several rules raised by the same event. A rule can be an ECA rule with complete event-condition-action semantics, or an E-A rule, in which the condition is implicitly specified by the database query language in a trigger.

The creation of a trigger is shown in Table 3. In both contexts, the *trigger-event* could be one of the three data manipulation operations, *INSERT, UPDATE* and *DELETE*.

An Example Table *ACBR_TRAVEL_DATA* stores the user data of the travel agent domain. The attributes of *ACBR_TRAVEL_DATA* are one-to-one mapping

Table 3. Trigger representation in SQL Server and Oracle

SQL Server	Oracle
CREATE TRIGGER *trigger-name* **ON** *table-name* **FOR** *trigger-event* **AS** *Transact-SQL block*	**CREATE TRIGGER** *trigger-name* **BEFORE** | **AFTER** *trigger-event* **ON** *table-name* [**FOR EACH ROW** [**WHEN** (*condition*)]] *PL/SQL block*

to the features of the travel agent case base. The distinction between the two is the raw data in user table can be either symbolic or numeric, e.g., attribute *price* can be any positive real number, while the value of feature *Price* is generalized and converted to symbolic value like *high* or *over 8,000*.

Consider an example rule: *If the new record has price over USD8,000, set the Price of FEATURE table to 'high'*. It can be represented as an SQL Server trigger:

CREATE TRIGGER *INSERT_TGGR*
ON *ACBR_TRAVEL_DATA* **FOR** INSERT
AS
BEGIN
/* other rules */
⋮
IF (*new.price* > 8000)
UPDATE *ACBR_TRAVEL_FEATURE* SET *Price* = '*high*'
⋮
/* other rules */
END

Note that multiple rules are generally stored in one trigger for the INSERT event. In Oracle, it is possible to map one rule to one trigger:

CREATE TRIGGER *PRICE_HIGH_TGGR*
AFTER INSERT **ON** *ACBR_TRAVEL_DATA*
FOR EACH ROW WHEN (*new.price* > 8000)
BEGIN
UPDATE *ACBR_TRAVEL_FEATURE* SET *Price* = '*high*'
END

3 Algorithms

Figure 1 depicts a high-level view of the two-layer `ActiveCBR` architecture. Briefly, the procedures of case-based reasoning, such as case retrieval, case adap-

tation, and case maintenance, are performed in the higher *CBR Layer*; while the lower *Active Layer* encapsulates the reactive functionality to monitor the alterations of external data sources. The interaction between the two layers is carried through the feature-value pairs that are accessible to both layers. In the higher layer, feature-value pairs are used to describe the similarity between the new problem and the retained cases; while in the lower layer, they reflect the result of data alterations.

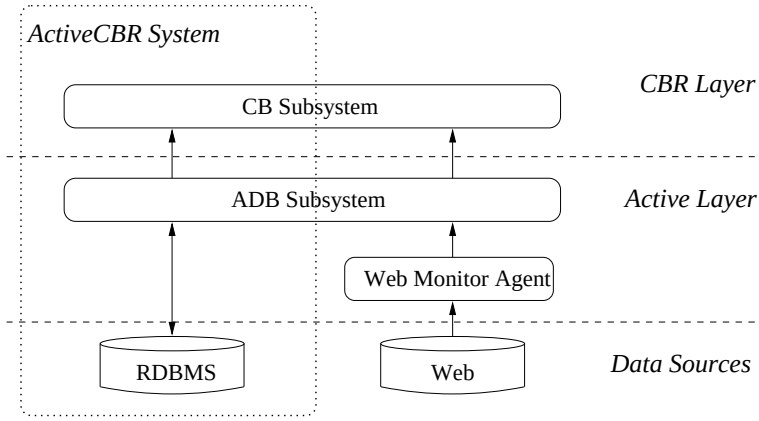

Fig. 1. The `ActiveCBR` architecture

The algorithms in the two subsystems of the `ActiveCBR` systems are independent. The *Case Arthoring* module in the CB subsystem and the *Rule Definition* module in the ADB subsystem primarily work on system reconfiguration for system flexibility. We will mainly, in this section, discuss the algorithms in *Case Firing* module and *Rule Execution* module.

3.1 Algorithm in the CB Subsystem

Suppose we have M cases in the case base $\mathcal{C} = \{c^1, ..., c^M\}$. For the N features $f_1, ..., f_N$, let f_n^m denote the value of the n^{th} feature of the m^{th} case, and f_n^I denote the current input value of the n^{th} feature. Now we have:

Case Firing Algorithm:

1. *For each new case added at runtime, mark it as enabled;*
2. *Retrieve current feature values* f_n^I*;*
3. *For each case* c^m *in* $\mathcal{C}$ *that is marked enabled:*
   (a) *For each feature* f_n*:*
   Calculate the similarity $sim(f_n^m, f_n^I)$*;*
   (b) *Calculate the score of case* c^m*;*

(c) Mark c_m as fired, if the score of c^m is greater than its threshold;
(d) Update the firing history log, if necessary;
4. Visualize case firing monitor.

CBR subsystem provides a user interface to add a new case and change the enabled / disabled status of an existing case at runtime. Before the case retrieval iteration, Step 1 of case firing algorithm examines whether a new case has been added into case base, and if so, marks it as enabled. This operation maximizes the system flexibility to perform real-time knowledge management.

Let κ be the average number of possible values of each feature. The total number of feature-value weights is:

$$| \mathcal{I} | = M * N * \kappa \tag{1}$$

Therefore, the complexity of above case-firing algorithm is $O(MN\kappa)$, i.e., the algorithm is linear in terms of number of cases, number of features, and average number of values for each feature.

The linear algorithm can be improved if we have an appropriate approach to cluster the cases [10]. If the clustering is effective, we can guarantee that no case from different clusters can be fired simultaneously. Hence, we need not traverse the whole case base in the case firing algorithm; alternatively, we need just traverse along the clustering path until entering a cluster, and check the firing condition with the cases in this cluster only. On the best case, we can suppose the clustering is even and the search tree is balanced. If, on average, each cluster contains λ cases, we can improve the computation to:

$$O(\lambda * log(\frac{M}{\lambda}) * N * \kappa) \tag{2}$$

3.2 Algorithm in the ADB Subsystem

The algorithm in the ADB subsystem depends on the trigger property of the underlying relational database. We implement the `ActiveCBR` system based on two RDBMS: SQL Server and Oracle.

The rules in Oracle support two distinct granularities, *row-level* and *statement-level*, corresponding to the instance-oriented semantics and the set-oriented semantics, respectively. They also can be executed either *before* or *after* the triggering operation. Thus, there are four possible combinations by combining the two granularities and the two evaluation times, i.e., *before* and *after* [9]. Besides the rules, database built-in integrity checking is also executed when a database manipulation occurs.

The rule processing algorithm in Oracle is:

Oracle Rule Processing Algorithm:

1. Execute the statement-level before-rules;
2. For each row in the target table:
(a) Execute the row-level before-rules;

(b) *Perform the modification of the row and row-level referential integrity and assertion checking;*
(c) *Execute the row-level after-rules;*

3. *Perform the statement-level referential integrity and assertion checking;*
4. *Execute the statement-level after-rules.*

4 Example of `ActiveCBR` System in Operation

In this section, we give a comprehensive example in the cable TV domain to demonstrate how the `ActiveCBR` system works.

The user data is stored in data tables such as *ACBR_CABLE_DATA*. The content of *ACBR_CABLE_DATA* is modified at runtime, e.g., insertion when new problem reported, updating when the problem solved later, and deletion when record out-of-date. Consider that two new tuples with problem id (pid) *6742* and *6743* are inserted into *ACBR_CABLE_DATA* (marked with * in Table 4). An *INSERT* event occurs accordingly.

Table 4. User data of cable TV domain

pid	uid	active	type	channel	time	location	solved
6732	263	Y	No picture	6	10/05 19:06	Burnaby	N
6733	546	Y	Reception	all	10/13 12:51	N.Van	Y
6734	649	Y	VCR	15	10/17 21:34	N.Van	N
6735	032	Y	VCR	n/a	10/19 02:25	Burnaby	N
6736	382	N	Reception	50	10/19 21:45	Burnaby	N
6737	234	Y	Reception	all	10/20 16:23	W.Van	N
6738	271	Y	Reception	9	10/20 20:42	Burnaby	N
6739	031	Y	No picture	13	10/20 22:19	Burnaby	N
6740	740	Y	VCR	11	10/20 22:43	UBC	N
6741	638	Y	VCR	28	10/20 23:19	SFU	N
6742*	957	Y	Reception	3	10/20 23:32	Burnaby	N
6743*	271	Y	Reception	6	10/20 23:57	Burnaby	N

The preprocessing performs slicing and dicing operations on user data to reduce the size of query table. For instance, in slicing operation, the tuples with 'N' value of *active* attribute are excluded from further query; in dicing operation, all the attributes that are not related to rule conditions such as *pid* and *solved* are removed as well. A temporary table #USERDATA is created in the preprocessing, and the number of tuples in #USERDATA is counted into variable *@num*.

```
SELECT type, channel, time, location
   INTO #USERDATA                    /* user data after preprocessing */
   FROM ACBR_CABLE_DATA
   WHERE active = 1
SELECT @num = COUNT(*)               /* number of tuples */
   FROM #USERDATA
```

Next, rule conditions are evaluated upon the *#USERDATA* table. In the `ActiveCBR` system, the values of features of the higher level case base are updated dynamically by the rule action from the lower level active database. For each feature in the feature space, there are several active rules to be evaluated, but only one of the conditions could be true, then the feature is set to a corresponding value.

Consider the case in Table 2. For feature *Duration*, its value can be *'recent 12 hrs'*, *'recent 3 days'*, and *'not specified'*. The rule used to update feature *Duration* to *'recent 12 hrs'* can be described as:

on INSERT
if At least 1/3 of total tuples and at least 5 tuples are reported within the last 12 hours.
then Update feature *Duration* with value *'recent 12 hrs'*.

The rule can be represented as a block of SQL statements in the *INSERT* event trigger:

```
    ⋮
SELECT @npart = COUNT(*)
    FROM #USERDATA
    WHERE DATEDIFF(hour, time, GETDATE()) < 12
IF ((@npart * 3 > @num) AND (@npart > 5))
BEGIN
    UPDATE ACBR_CABLE_FEATURE
        SET value = 'recent 12 hrs'
        WHERE feature = 'Duration'
END
    ⋮
```

After all rules are executed, we have the runtime feature values as shown in row *Current Value* in Table 5.

The result of similarity computation is listed in the last row in Table 5. Accordingly, we can compute the score of this case. In the above example, the result score is 96. Since it is greater than the threshold 78, the case *Regional switch (LB) problem* is marked as *'fired'*.

Table 5. Runtime feature values and similarity computation of case *Regional switch (LB) problem*

	ProblemType	**Channels**	**Duration**	**Location**
Current Value f_n^I	reception	lower band	recent 12 hrs	particular
Importance Weight w_n^m	75	80	70	85
Feature Value Weight $\omega(n, m, f_n^I)$	65	80	70	85
similarity $sim(f_n^m, f_n^I)$	0.87	1.00	1.00	1.00

The system will fire and visualize every case whose score exceeds its threshold at runtime.

5 Conclusions and Future Work

This work is originally based on the need to improve traditional 'passive' case-based reasoning system to have active property. The 'active' property has two related meanings: first, the system should be capable of responding to an external event. The response should be made in real-time fashion within a limit of time.

The contributions of this work are two-fold. The combined knowledge representation in different level improves system performance by reducing the size of rule set and the confliction between the rules by factoring out the two sets of knowledge bases. Also, the combined system allows case-based reasoning to apply to real-time database environment.

A prevailing trend of distributed computing demands a distributed architecture for our `ActiveCBR` applications. Several models are under consideration, such as using one CB subsystem via multiple ADB subsystems or multiple CB subsystems being centralized with a predominant CB system. To complete distributed tasks effectively, the `ActiveCBR` system should be enhanced to have capacity of integrating multiple-source information, performing inter-subsystem communication, and avoiding possible information bottleneck.

Acknowledgment

We thank NSERC and IRIS for their support for this research.

References

1. A. Aamodt and E. Plaza. Foundational issues, methodological variations, and system approaches. *Artificial Intelligence Communications*, 7(1):39–59, 1993.

2. A. Biliris. The performance of three database storage structures for managing large objects. In *ACM SIGMOD Conference on the Management of Data*, San Diego, CA, 1992. 5
3. S. Ceri, P. Fraternali, S. Paraboschi, and L. Branca. Active rule management in Chimera. In *Active Database Systems - Triggers and Rules For Advanced Database Processing*, pages 151–76. Morgan Kaufman, 1996. 5
4. O. Diaz and A. Jaime. EXACT: an extensible approach to active object-oriented databases. *VLDB Journal*, 6(4):282–295, 1997. 5
5. O. Diaz, A. Jaime, N. W. Paton, and G. Qaimari. Supporting dynamic displays using active rules. *ACM SIGMOD Record*, 23(1):21–26, 1994. 4
6. D. Gentner. Structure mapping: a theoretical framework for analogy. *Cognitive Science*, 7:155–70, 1983.
7. J. L. Kolodner. *Case-Based Reasoning*. Morgan Kaufmann Publisers, Inc., 1993.
8. D. B. Leake, A. Kinley, and D. Wilson. Learning to improve case adaptation by introspective reasoning and CBR. In *Proceedings of the First International Conference on Case-Based Reasoning*. Springer-Verlag, 1995.
9. K. Owens and S. Adams. Oracle 7 triggers: Mutating tables? *Database Programming and Design*, 7(10):31–49, 1994. 10
10. J. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986. 10
11. Y. Shoham. An overview of agent-oriented programming. In J. M. Bradshaw, editor, *Software Agents*, pages 271–90. AAAI Press, 1997.
12. I. Watson and F. Marir. Case-based reasoning: A review. *The Knowledge Engineering Review*, 9(4):355–81, 1994.