Active Database Management Systems

Mariano A. Cilia
Technische Universität Darmstadt, Germany and UNICEN, Argentina

INTRODUCTION

As it is well known, business requirements are changing faster than applications can be created and/or modified. Most of these requirements are in the form of or are related to business rules. Business rules are precise statements that describe, constrain and control the structure, operations and strategy of a business. They may be thought of as small pieces of knowledge about a business domain. They offer a way of encapsulating business semantics and making them explicit in the same way that databases enable the separation of data from application programs.

Traditionally, business rules have been scattered, hard-coded and replicated by different applications. The lack of a formal approach to the management of business rules and a standard business rule language has made it virtually impossible to create, modify and manage business rules in a flexible way. As a result, it has been difficult to adapt applications to new requirements quickly.

Isolating the business rules from the application code enables developers to easily find and modify the pertinent rule(s) when a policy change is needed or when application requirements change. This makes it possible to quickly change rules without modifying the rest of the application code, thereby enhancing maintainability.

In the last decade, one of the trends in database technology has focused on extending conventional database systems (DBMSs) to enhance their functionality and to accommodate more advanced applications. One of these enhancements was extending database systems with powerful rule-processing capabilities. These capabilities can be divided into two classes: *deductive*, in which logic-programming-style rules are used to provide a more powerful user interface than that provided by most database query languages (Ceri, Gottlob, & Tanca, 1990), and *active*, where production-style rules are used to provide automatic execution of predefined operations in response to the occurrence of certain events (Act-Net Consortium, 1996; Dayal , Buchmann & McCarthy, 1988). The latter is particularly appropriate for enforcing business rules as it has been demonstrated in Ceri and Widom (1996) and Paton (1999). Database systems enhanced with active capabilities are known as *active databases systems*, or aDBMSs for short.

By means of active database systems, general integrity constraints encoded in applications have been moved into the database system in the form of rules. These rules go beyond key or referential integrity constraints. Active databases support the specification and monitoring of general constraints (rules), they provide flexibility with respect to the time of constraint checking, and they provide execution of compensating actions to rectify a constraint violation without rolling back the involved transaction. Additionally, support for external events and actions were introduced mostly to satisfy the requirements of monitoring applications.

As a consequence, applications sharing the same database system and data model can also share business rules. In this way, the business knowledge that was dispersed in many applications in the form of programming code is now represented in the form of rules and managed in a centralized way. Consequently, when business rules change, only those affected rules must be modified in the aDBMS.

BACKGROUND

Historically, production rules were used first to provide automatic reaction functionality. Production rules are Condition-Action rules that do not break out the triggering event explicitly. Instead, they implement a polling-style evaluation of all conditions. In contrast, Event-Condition-Action (ECA) rules explicitly define triggering events, and conditions are evaluated only if the triggering event was signaled. Active databases adopted this event-driven approach to avoid unnecessary and resource-intensive polling in monitoring database changes and applications.

In active relational databases, events were modeled as changes of the state of the database, i.e., *insert*, *delete* and *update* operations on tables (Hanson, 1989; Stonebraker, Jhingran, Goh, & Potamianos, 1990). This basic functionality is common fare in commercial DBMSs today. In object-oriented systems, more general primitive events were defined: temporal events, both absolute and relative; method invocation events; and user-defined events (Dayal & Blaustein, 1988; Gatziu & Dittrich, 1993; Zimmermann & Buchmann, 1999).

In addition to these primitive events, more complex situations that correlate, aggregate or combine events can be defined. This is done by using an event algebra

(Zimmer & Unland, 1999) that allows the definition of so-called *composite* or *complex events*.

By means of a rule definition language (RDL), ECA rules are specified. This language provides constructors for the definition of rules, events, conditions and actions. Once rules are specified with an RDL, the rule base should be checked according to the following properties: termination, confluence and observably deterministic behavior (Aiken, Widom, & Hellerstein, 1992). For instance, termination analysis ensures that the current rule base is safe and correct with respect to the intended reactive behavior. In general, these properties cannot be proven automatically, but an analyzer might assist in detecting inconsistencies. After this verification step, rules are ready to be processed.

ECA RULE PROCESSING

Rule execution semantics prescribe how an active system behaves once a set of rules has been defined. Rule execution behavior can be quite complex, but we restrict ourselves to describing only essential aspects here. For a more detailed description, see Act-Net Consortium (1996) and Widom and Ceri (1996).

All begins with event instances signaled by event sources that feed the complex event detector, which selects and consumes these events. *Consumption modes* (Chakravarthy, Krishnaprasad, Anwar, & Kim, 1994) determine which of these event instances are considered for firing rules. The two most common modes are *recent* and *chronicle*. In the former, the most recent event occurrences are used, while in the latter, the oldest event occurrences are consumed. Notice that in both cases a temporal order of event occurrences is required. Different consumption modes may be required by different application classes.

Usually there are specific points in time at which rules may be processed during the execution of an active system. The *rule processing granularity* specifies how often these points occur. For example, the finest granularity is "always," which means that rules are processed as soon as any rule's triggering event occurs. If we consider the database context, rules may be processed after the occurrence of database operations (small), data manipulation statements (medium), or transactions (coarse).

At granularity cycles and only if rules were triggered, the *rule processing algorithm* is invoked. If more than one rule was triggered, it may be necessary to select one after the other from this set. This process of *rule selection* is known as *conflict resolution*, where basically three strategies can be applied: (a) one rule is selected from the fireable pool, and after rule execution, the set is determined again; (b) sequential execution of all rules in an evaluation cycle; and (c) parallel execution of all rules in an evaluation cycle.

After rules are selected, their corresponding conditions are evaluated. Notice that conditions can be expressed as predicates on database states using a query language, and also external method invocations can be used. The specification of conditions can involve variables that will be bound at runtime with the content of triggering events. If a condition evaluates to true, then the action associated with this rule must be executed. Actions can be any sequence of operations on or outside of a database. These operations can include attributes of the triggering event.

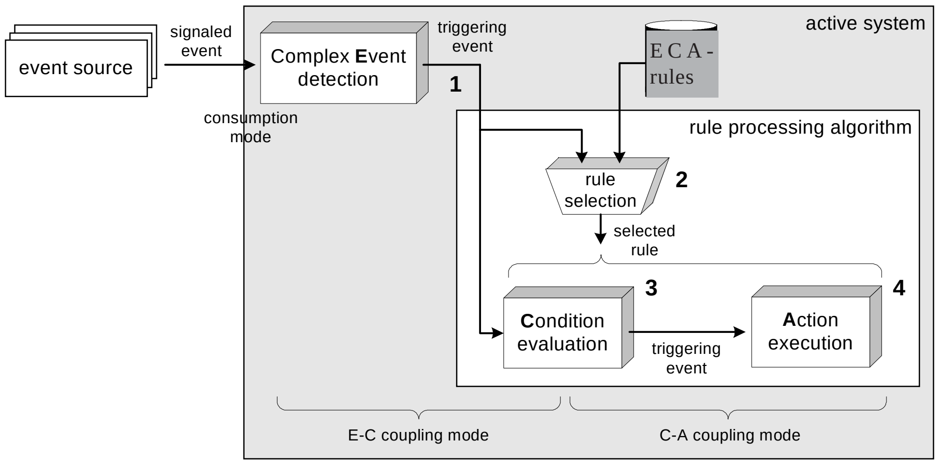
For some applications it may be useful to delay the evaluation of a triggered rule's condition or the execution of its action until the end of the transaction, or it may be useful to evaluate a triggered rule's condition or execute its action in a separate transaction. These possibilities lead to the notion of *coupling modes*. One coupling mode can specify the transactional relationship between a rule's triggering event and the evaluation of its condition while another coupling mode can specify the transactional relationship between a rule's condition evaluation and the execution of its action. The originally proposed coupling modes are immediate, deferred and decoupled (Dayal & Blaustein, 1988).

To sum up, an ECA-rule-processing mechanism can be formulated as a sequence of four steps (as illustrated in Figure 1):

1. **Complex event detection:** It selects instances of signaled events in order to detect specified situations of interest where various event occurrences may be involved. As a result, these picked instances are bound together and signaled as a (single) complex event.
2. **Rule selection:** According to the triggering events, fireable rules are selected. If the selection is multiple, a conflict resolution policy must be applied.
3. **Condition evaluation:** Selected rules receive the triggering event as a parameter, thus allowing the condition evaluation code to access event content. Transaction dependencies between event detection and the evaluation of a rule's condition are specified using Event-Condition coupling modes.
4. **Action execution:** If the rule's condition evaluates to true, the corresponding action is executed taking the triggering event as a parameter. Condition-action dependencies are specified similarly to Step 3.

It should be noticed that Step 1 (complex event detection) can be skipped if rules involve primitive events only.

Figure 1. Schematic view of the ECA-rule-processing mechanism



TOOLS

Several tools are required in order to adequately support the active functionality paradigm (Act-Net Consortium, 1996). For instance, a *rule browser* makes possible the inspection of the rule base; a *rule design assistance* supports the process of creation of new rules and is complemented with a *rule analyzer* that checks for a consistent rule base; and a *rule debugger* monitors the execution of rules.

FUTURE TRENDS

Modern large-scale applications, such as e-commerce, enterprise application integration (EAI), Internet or intranet applications, and pervasive and ubiquitous computing, can benefit from this technology, but they impose new requirements. In these applications, integration of different subsystems, collaboration with partners' applications or interaction with sensor devices is of particular interest.

It must be noticed that in this context, events and data are coming from diverse sources, and the execution of actions and evaluation of conditions may be performed on different systems. Furthermore, events, conditions and actions may not be necessarily directly related to database operations. This leads to the question of why a full-fledged database system is required when only active functionality and some services of a DBMS are needed.

The current trend in the application space is moving away from tightly coupled systems and towards systems of loosely coupled, dynamically bound components. In such a context, it seems reasonable to move required active functionality out of the active database system by offering a flexible service that runs decoupled from the database. It can be combined in many different ways and used in a variety of environments. For this, a component-based architecture seems to be appropriate (Collet, Vargas-Solar, & Grazziotin-Ribeiro, 1998; Gatziu, Koschel, von Buetzingsloewen, & Fritschi, 1998;), in which an *active functionality service* can be seen as a combination of other components, like complex event detection, condition evaluation and action execution. Thus, components can be combined and configured according to the required functionality, as proposed by the unbundling approach in the context of aDBMSs (Gatziu et al.) or by using a service-oriented architecture (SOA) as described in Cilia, Bornhövd, and Buchmann (2001).

CONCLUSION

Active databases support the specification of business rules in the form of ECA rules. The business knowledge that was dispersed in many applications in the form of programming code is now represented in the form of rules and managed in a separated way. This facilitates the adaptation to new requirements and improves the maintainability of systems.

As mentioned above, the current trend of active functionality technology shows a shift towards loosely coupled and distributed systems and the availability of such functionality as a service.

REFERENCES

Act-Net Consortium. (1996). The active database management system manifesto: A rulebase of ADBMS features. *ACM SIGMOD Record*, *25*(3), 40-49.

Aiken, A., Widom, J., & Hellerstein, J. (1992). Behavior of database production rules: Termination, confluence, and observable determinism. *Proceedings of ACM International Conference on Management of Data (SIGMOD)*, 59-68.

Ceri, S., Gottlob, G., & Tanca, L. (1990). *Logic programming and databases*. Springer.

Ceri, S., & Widom, J. (1996). Applications of active databases. In J. Widom & S. Ceri (Eds.), *Active database systems* (pp. 259-291). Springer.

Chakravarthy, S., Krishnaprasad, V., Anwar, E., & Kim, S. (1994). Composite events for active databases: Semantics, contexts and detection. *Proceedings of the International Conference on Very Large Databases (VLDB)* (pp. 606-617).

Cilia, M., Bornhövd, C., & Buchmann, A. (2001). Moving active functionality from centralized to open distributed heterogeneous environments. In *Lecture notes in computer science: Vol. 2172. Proceedings of the International Conference on Cooperative Information Systems, CoopIS* (pp. 195-210).

Collet, C., Vargas-Solar, C., & Grazziotin-Ribeiro, H. (1998). Towards a semantic event service for distributed active database applications. In *Lecture notes in computer science: Vol. 1460. Proceedings of the International Conference on Databases and Expert Systems Applications, DEXA* (pp. 16-27).

Dayal, U., Blaustein, B., Buchmann, A., Chakravarthy, S., Hsu, M., Ledin, R., et al. (1988). The HiPAC project: Combining active databases and timing constraints. *ACM SIGMOD Record, 17*(1), 51-70.

Dayal, U., Buchmann, A., & McCarthy, D. (1988). Rules are objects too. In *Lecture notes in computer science: Vol. 334. Proceedings of the 2nd International Workshop on Object-Oriented Database Systems* (pp. 129-143).

Gatziu, S., Koschel, A., von Buetzingsloewen, G., & Fritschi, H. (1998). Unbundling active functionality. *ACM SIGMOD Record*, *27*(1), 35-40.

Gatziu, S., & Dittrich, K. R. (1993). Events in an active object-oriented database system. *Proceedings of the 1st International Workshop on Rules in Database Systems (RIDS'93)* (pp. 23-29).

Hanson, E. (1989). An initial report on the design of Ariel: A DBMS with an integrated production rule system. *ACM SIGMOD Record*, *18*(3), 12-19.

Paton, N. (Ed.). (1999). *Active rules in database systems*. Springer.

Stonebraker, M., Jhingran A., Goh, J., & Potamianos, S. (1990). On rules, procedures, caching and views in data base systems. *Proceedings of the 1990 ACM International Conference on Management of Data (SIGMOD)* (pp. 281-290).

Widom, J., & Ceri, S. (Eds.). (1996). *Active database systems: Triggers and rules for advanced database processing*. Morgan Kaufmann.

Zimmer, D., & Unland, R. (1999). On the semantics of complex events in active database management systems. *Proceedings of the 15th International Conference on Data Engineering (ICDE'99)* (pp. 392-399).

Zimmermann, J., & Buchmann, A. (1999). REACH. In N. Paton (Ed.), *Active rules in database systems* (pp. 263-277). Springer.

KEY TERMS

Business Rules: They are precise statements that describe, constrain and control the structure, operations and strategy of a business. They may be thought of as small pieces of knowledge about a business domain.

Consumption Mode: It determines which of these event instances are considered for firing rules. The two most common modes are recent and chronicle.

Coupling Mode: It specifies the transactional relationship between a rule's triggering event, the evaluation of its condition and the execution of its action.

ECA Rule: It is a (business) rule expressed by means of an event, a condition and an action.

Event: It is an occurrence of a happening of interest (also known as primitive event). If the event involves correlation or aggregation of happenings then it is called a complex or composite event.

Event Algebra: Composite events are expressed using an event algebra. Such algebras require an order function between events to apply event operators (e.g., sequence) or to consume events.

Rule Base: It is a set of ECA rules. Once this set is defined, the aDBMS monitors for relevant events. The rule base can be modified (new rules can be added, or existent rules can be modified or deleted) over time.

Rule Definition Language (RDL): Set of constructs for the definition of rules, events, conditions and actions.