A FLEXIBLE EVENT-CONDITION-ACTION (ECA) RULE PROCESSING MECHANISM BASED ON A DYNAMICALLY RECONFIGURABLE STRUCTURE

Xiang Li, Ying Qiao and Hongan Wang
Intelligence Engineering Laboratory, Institute of Software, Chinese Academy of Sciences
No. 4, Zhong Guan Cun South Fourth Street, Hai Dian District, Beijing, China

Keywords: Event-condition-action rules, On-the-fly, Active database, Processing structure, Middleware.

Abstract: Adding and deleting Event-Condition-Action (ECA) rules, i.e. modifications of processing structures in an active database are expected to happen on-the-fly, to cause minimum impact on existing processing procedures of ECA rules. In this paper, we present a flexible ECA rule processing mechanism in active database. It uses a dynamically reconfigurable structure, called unit-mail graph (UMG) and a middleware, called Unit Modification and Management Layer (UMML) to localize the impact of adding and deleting ECA rules so as to support on-the-fly rule modification. The ECA rule processing mechanism can continue to work when the user adds or deletes the rules. This makes active database to be able to react to external events arriving at the system during rule modification. We also use a smart home environment to evaluate our work.

1 INTRODUCTION

1.1 Motivations

Event-condition-action (ECA) rules play very important roles in active database since they specify how the active database performs suitable actions in response to events that happened. ECA rule processing mechanism is responsible to execute the predefined ECA rules to find the actions taken to react to external events arriving at the system.

In some responsive system, such as many real-time monitoring systems (e.g., the fire monitoring system, the flood monitoring system etc.), ECA rules need to be modified to reflect the user requirement changes about how to react to occurring events in the system. To guarantee the dependability, the system should still respond to the events that occur during ECA rule modifications. This means the ECA rules should be modified on the fly. In another word, the ECA rules should be added or deleted without stopping the ECA rule processing mechanism.

In this paper, we present a flexible ECA rule processing mechanism based on a dynamically reconfigurable structure. The mechanism is to enable the user to modify ECA rules on the fly in active database. The ECA rules can be added and deleted without stopping ECA rules processing mechanism so that the active database is still able to react to external events arriving at the system during rule modification. The rule processing mechanism converts the specified ECA rules into a dynamically reconfigurable structure, called unit-mail graph (UMG). Furthermore, a middleware, called UMML is developed to modify the UMG according the user command. All these efforts will localize the impact of adding and deleting ECA rules so that allow the dynamical adding and deletion of ECA rules without stopping the ECA rules processing mechanism.

1.2 Related Work

The internal structures used by current ECA rule processing mechanism (Gatziu, 1994) (Dittrich, 2000) (Chakravarthy, 1999) (Chakravarthy, 1993) (Chakravarthy, 1994) (Gehani, 1992) (Krishnaprasad, 1994) are holistic. In such holistic structure, ECA rules cannot be modified on the fly. This makes current ECA rule processing mechanism not suitable for the applications in which both 24 hours and 7 days running and modification of ECA rules are needed.

Li X., Qiao Y. and Wang H. (2009).
A FLEXIBLE EVENT-CONDITION-ACTION (ECA) RULE PROCESSING MECHANISM BASED ON A DYNAMICALLY RECONFIGURABLE STRUCTURE.
In *Proceedings of the 11th International Conference on Enterprise Information Systems - Databases and Information Systems Integration*, pages 291-294
DOI: 10.5220/0001987402910294

The rest of the paper is organized as follows: section 2 addresses the framework of the ECA rule processing mechanism; section 3 evaluates UMG and UMML via a smart home environment; conclusions and future works are stated in section 4.

2 FRAMEWORK OF THE ECA RULE PROCESSING MECHANISM

The framework of the ECA rule processing mechanism is shown in Figure 1. The ECA rule processing mechanism has three layers, i.e., user interface layer, middleware layer and a processing layer. In user interface layer, a visual specification tool, called VST, is provided for the user to specify ECA rules (Qiao, 2007), (Liu, 2008). Furthermore, the user can send the command of adding rules or deleting rules via a command interface.

The XML streams of specified ECA rules and user commands are sent to middleware layer which is also called Unit Modification and Management Layer (UMML). UMML will convert the XML stream of the specified ECA rules into a rule graph, in which the event branch is same to that used in Snoop (Chakravarthy, 1993) (Chakravarthy, 1994) (Chakravarthy, 1999) and the condition branch is equivalent to a Boolean tree. After then, UMML will further convert the rule graph into a modularized and loose-coupling structure for ECA rule processing, called Unit-Mail Graph (UMG), by packing each vertex in the rule graph into unit. When UMML receives the XML stream for the command of adding rules, it converts the rules to be added into temporary UMG and will merge the temporary UMG with the existing UMG by dynamically adding units in the temporary UMG to the unit warehouse. In the same way, when UMML receive the XML stream for the command of deleting rules, UMML will modify the existing UMG by deleting unused units form the Unit Warehouse according to the command.

The Processing Layer is responsible to receive external events, detect composite events, evaluates conditions and performs actions.

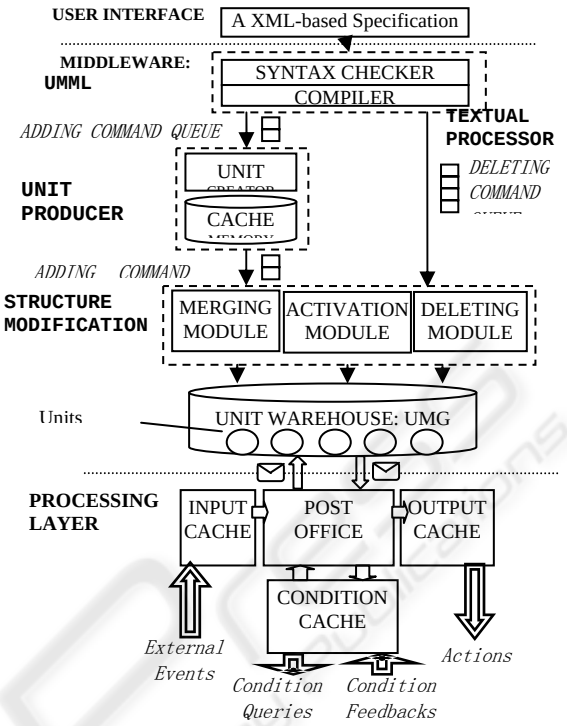

Figure 1: The structure of our flexible framework for ECA rules processing.

2.1 UMG

In order to modularize the rule graph, we pack each vertex in graph into a unit. Instead of direct edges between units, each unit records the identification of the unit that receives the information. (For convenience, we call the unit that receives the information as output unit). Therefore, in each unit, there is an independent space to record output units. Processing ECA rules are translated to passing and processing mails in the UMG.

A unit has a processor to perform the detection of composite events, evaluation of the conditions and other necessary operations. It also has several input ports that allow the processor to provide different ways to process instances from different sources. In addition, processors in units are only able to process instances, but all instances are packed in mails. Thus, a unit has to unpack a mail when it receives the mail. Meanwhile, a unit needs to pack instances to mails if it wants to send these instances to other units. Figure 2(a) shows structure of a unit.

Mails are used to pass the information about event or condition instances to other units. Figure 2(b) shows structure of a mail. A mail has five parts:

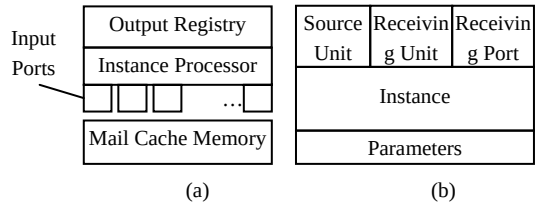

Figure 2: (a) The structure of a unit, and (b) the structure of a mail.

UMG has two independent characters: 1) units are independent of each other. 2) In each unit, the output registry space is independent of the processor.

2.2 UMML

In UMML, In order to add new units or delete unused units correctly, two questions should be considered: 1) how to ensure the correctness of the processing structure in the Unit Warehouse in each step, and 2) how to take the least effect on the logical structure of existed units. Here, the correctness has two aspects: a) in each step of adding or deleting, the system should run correctly, and b) if there are some errors, the framework should repair the processing structure.

The UMML provides four mechanisms:

Loose Coupling mechanism means the two independent characters.

Activation/Inactivation mechanism makes units not be able to send mails to incorrect or unready units.

Top-Down mechanism makes Merging Module add units and the Deleting Module delete units in top-down order.

Roll-Back mechanism makes the rule processing mechanism avoid unexpected errors.

3 CASE STUDY

3.1 ECA Rules

In this section, we will evaluate our framework via smart home system. We develop our processing system with Java in Eclipse. We specify several ECA rules used in smart home with specification tools presented in (Liu, 2008). The detailed requirements for ECA rules in the smart home system are shown in Table 1. Rule1 and Rule 2 are used in normal cases; Rule 3 and Rule 4 are used in emergent cases.

Table 1: ECA rules.

Req1: On “1 hour after the cooker is on” If “the cooker is still on” Do “Trigger alarm”
Req2: On “ 30 minutes after the old man enter the bath room” If “the old man is still in the bath room” Do “Trigger alarm”
Req3: On “2 hour after the cooker is on” If “the cooker is still on” Do “Trigger emergent alarm”
Req4: On “ 1 hour after the old man enter the bath room” If “the old man is still in the bath room” Do “Trigger emergent alarm”

3.2 Evaluations

In the first experiments, we add Rule 3 and 4 when the system is processing Rule 1 and 2. Table 2 shows the results of the experiment.

Table 2: Evaluation Results for Experiment 1.

	Before adding new ECA rules		**After adding new ECA rules**	
Rules in processing system	Rule 1 and Rule 2		Rule 1, Rule 2, Rule 3 and Rule 4	
Number of units	32		40	
Input events for cooker	Turn on the cooker		Turn on the cooker	
	After 1 hour	After 2 hour	After 1 hour	After 2 hour
Actions for cooker	Alarm	None	Alarm	Emergent alarm
Input events for bath room	Enter bath room		Enter bath room	
	After 30 minutes	After 1 hour	After 30 minutes	After 1 hour
Actions for bath room	Alarm	None	Alarm	Emergent alarm

In the second experiment, we will delete Rule 1 and 2 when the system is processing Rule 3 and 4.

Table 3: Evaluation Results for Experiment 2.

	Before deleting invalid ECA rules		**After deleting unused ECA rules**	
Rules in processing system	Rule 1, Rule 2, Rule 3 and Rule 4		Rule 3 and Rule 4	
Number of units	40		24	
Input events for cooker	Turn on the cooker		Turn on the cooker	
Actions for cooker	After 1 hour	After 2 hour	After 1 hour	After 2 hour
	Alarm	Emergent alarm	None	Emergent alarm
Input events for bath room	Enter bath room		Enter bath room	
Actions for bath room	After 30 minutes	After 1 hour	After 30 minutes	After 1 hour
	Alarm	Emergent alarm	None	Emergent alarm

Obviously, Rule 1 and Rule 3 share some units and Rule 2 and Rule 4 share some units.

From Table 2 and Table 3, we can observe two processes:

- Process 1: when adding Rule 3 and Rule 4, the processing system keeps working correctly and performs correct actions for Rule 1 and Rule 2. Adding Rule 3 and Rule 4 does not affect any existing units' working, although Rule 3 and Rule 4 reuse several existing units.
- Process 2: when deleting Rule 1 and Rule 2, the processing system keeps working correctly and performs correct actions for Rule 3 and Rule 4. Deleting Rule 1 and Rule 2 does not affect valid units' working; meanwhile some unused units are deleted from the processing system.

Here, the processing system keeps working correctly in the process of adding and deleting. Thus, UMG and UMML can ensure the correctness of the processing system. In other words, the framework with UMG and UMML is effective.

4 CONCLUSIONS AND FUTURE WORKS

In this paper, we present a flexible Event-condition-action (ECA) rule processing mechanism including a dynamic reconfiguration structure UMG and a middleware for modifying and managing UMG called UMML. UMG has the two independent characters. UMML provides four mechanisms to ensure that the framework for rules processing keeps working when ECA rules are modified. Furthermore, we use a smart home system to evaluate our work.

Units are independent, so the UMG can be used easily in distributed environments. Using the UMG and the UMML in distributed environments is our future work.

ACKNOWLEDGEMENTS

This paper is supported by National Nature Science Foundation of China (Grant No. 60873073) and France Telecom (Grant No. 46135653).

REFERENCES

Gatziu, S., Dittrich, K.R., 1994. Detecting Composite Events in Active Databases Using Petri Nets. In *Proc. of the 4th International Workshop on Research Issues in Data Engineering: Active Database Systems*. IEEE Press.

Dittrich, K., Fritschi, H., Gatziu, S., Geppert, A., Vaduva, 2000. Technical report: SAMOS in Hindsight. In *Experiences in Building an Active Object-Oriented DBMS*.

Chakravarthy, S., Le, R., Dasari, R., 1999. ECA Rule Processing in Distributed and Heterogeneous Environments. In *Symposium on Distributed Objects and Applications*. IEEE Press.

Chakravarthy, S., Mishra, D., 1993. Snoop: An Expressive Event Specification Language For Active Databases. Technical report, Dept. of Comp. and Info. Sci., Univ. of FL, 1993.

Chakravarthy, S., Krishnaprasad, V., Anwar, E., Kim, S.K., 1994. Composite events for active databases: Semantics, contexts and detection. In *Proceedings of the 20th International Conference on Very Large Databases*.

N. H. Gehani, H. V. Jagadish, O. Shmueli, 1992. Event Specification in an Object-Oriented Database. In *Proceedings International Conference on Management of Data*.

V. Krishnaprasad. Event Detection for Supporting Active Capability in an OODBMS: Semantics, Architecture, and Implementation. Master's thesis, Database Systems R&D Center, CIS Department, University of Florida, E479-CSE, Gainesville, FL 32611, March 1994.

Qiao, Y., Zhong, K., Wang, H., Li, X., 2007. Developing Event-condition-action Rules in Real-time Active Database. In *Proceedings of ACM symposium on applied computing*, pp.511-516.

Liu, W., Qiao, Y., Li, X., Zhong, K., Wang, H., Dai, G., 2008. A Visual Specification Tool for Event-Condition-Action Rules Supporting Web-Based Distributed System. In *Proceedings of the Tenth International Conference on Enterprise Information Systems*, pp.246-251.