Available online at www.sciencedirect.com
ScienceDirect
Data & Knowledge Engineering 60 (2007) 530–546
DATA & KNOWLEDGE ENGINEERING
www.elsevier.com/locate/datak

A concurrent rule scheduling algorithm for active rules

Ying Jin [a,*], Susan D. Urban [b], Suzanne W. Dietrich [b]

[a] California State University, Sacramento, Department of Computer Science, 6000 J Street, Sacramento, CA 95819, USA
[b] Arizona State University, Ira A. Fulton School of Engineering, Department of Computer Science and Engineering, Tempe, AZ 85287, USA

Received 13 July 2004; received in revised form 11 April 2005; accepted 4 February 2006
Available online 2 May 2006

Abstract

The use of rules in a distributed environment creates new challenges for the development of active rule execution models. In particular, since a single event can trigger multiple rules that execute over distributed sources of data, it is important to make use of concurrent rule execution whenever possible. This paper presents the details of the integration rule scheduling (IRS) algorithm. Integration rules are active database rules that are used for component integration in a distributed environment. The IRS algorithm identifies rule conflicts for multiple rules triggered by the same event through static, compile-time analysis of the read and write sets of each rule. A unique aspect of the algorithm is that the conflict analysis includes the effects of nested rule execution that occurs as a result of using an execution model with an immediate coupling mode. The algorithm therefore identifies conflicts that may occur as a result of the concurrent execution of different rule triggering sequences. The rules are then formed into a priority graph before execution, defining the order in which rules triggered by the same event should be processed. Rules with the same priority can be executed concurrently. The IRS algorithm guarantees confluence in the final state of the rule execution. The IRS algorithm is applicable for rule scheduling in both distributed and centralized rule execution environments.

Keywords: Active rules; Concurrent rule execution; Rule scheduling algorithm; Confluence analysis

1. Introduction

Active database systems extend traditional databases by supporting mechanisms to automatically monitor and react to events that are taking place either inside or outside of the database system [28,38]. Active rules form the core of any active database system. An active rule, also known as an event-condition-action (ECA) rule, typically consists of three parts: an event, a condition, and an action. An event describes an occurrence that causes a rule to be triggered. The condition is a query over data sources that is checked when the rule is triggered, declaring the set of circumstances that must exist for the action of the rule to be processed. The action is executed in response to condition evaluation and can be used to modify data, retrieve data, or perform application procedures. Active rules were originally designed in the context of centralized database

* Corresponding author. Tel.: +1 916 2786250; fax: +1 916 2786774.
E-mail addresses: jiny@ecs.csus.edu (Y. Jin), s.urban@asu.edu (S.D. Urban), dietrich@asu.edu (S.W. Dietrich).

0169-023X/$ - see front matter
doi:10.1016/j.datak.2006.02.007

environments and have been used for integrity constraint maintenance and view maintenance as well as general monitoring and notification of specific database states and activities. Comprehensive information about active databases can be found in [28,38].

In addition to the design of architectures for the execution of active rules [38], past research on active rules has focused on the design of rule execution models [8,13,18,37,38]. A rule execution model determines how a set of rules behaves at runtime. Additional research on active rules has investigated properties of active rule behavior, such as *termination* [1,2,6,23,35,36] and *confluence* [1,2,5–7,24,36]. The termination property guarantees that a set of rules will not result in infinite, cyclic rule execution [2]. The confluence property guarantees that the final result of rule execution does not depend on the order in which rules are chosen for execution [2].

Active rules currently exist in commercial database systems in the form of triggers [31]. More recently, active rules have proven useful for controlling activities in centralized and distributed workflow systems [9,10,15,22,25] and for supporting event-based, application integration in distributed environments [11,12,17,26,27]. Our own research has investigated the use of active rules, known as integration rules (IRules), for developing a declarative event-based approach to component integration [14,32–34]. Through the support of the distributed rule execution engine that is provided by the IRules environment [19,20], an application can automatically respond to events from remote components by testing conditions over distributed components and invoking global transactions that execute over distributed sources. The integration rule processor developed as part of the IRules project has been designed for the integration of components with well-defined interfaces based on the enterprise java beans component model [16].

The use of rules in a distributed environment such as that of the IRules project creates new challenges for the development of active rule execution models. In particular, since a single event can trigger multiple rules that execute over distributed sources of data, it is important to make use of concurrent rule execution whenever possible. Our work has developed the integration rule processing (IRP) algorithm and the integration rule scheduling (IRS) algorithm. The IRP algorithm is based on an algorithm originally presented in [7], using *execution cycles* and *levels within cycles* to control the nested execution of integration rules in a distributed environment. The results of the IRP algorithm are reported in [19,21]. The IRS algorithm enhances the IRP algorithm with a static approach for scheduling the sequential and concurrent execution of multiple rules triggered by the same event. An important aspect of the IRS algorithm is that it guarantees confluence for concurrent rule execution.

This paper presents the details of the IRS algorithm. When rules are initially compiled, the algorithm identifies rules conflicts by analyzing the read and write sets of each rule in the set of rules triggered by an event. Non-conflicting rules can be executed in parallel. But for any two non-conflicting rules, r_1 and r_2, in a rule set, if r_2 triggers r_3, r_1 can potentially conflict with r_3 if r_1 and r_2 are allowed to execute concurrently. As a result, the algorithm makes use of the triggering graph to include the cascaded rules triggered by this initial rule set in the analysis process, assigning priorities to the analyzed rules. The prioritized rules are formed into a priority graph, defining the order in which rules triggered by the same event should be processed. Rules with the same priority can be executed concurrently. Furthermore, the IRS algorithm guarantees confluence in the final state of the rule execution. Confluence for the concurrent execution of rules has not been addressed in past research on confluence analysis. An added benefit of the IRS algorithm is that it is applicable for rule scheduling in both distributed and centralized rule execution environments.

The rest of this paper is organized as follows. Section 2 presents existing research on confluence analysis and compares the IRS algorithm with existing research. Section 3 presents an overview of the IRS algorithm along with assumptions and terminology. Section 4 describes the data access algorithm for conflict analysis of a rule set and the cascaded rules associated with each rule in the rule set. Section 5 elaborates on the priority graph construction algorithm for adding priorities to a rule set based on the conflict analysis described in Section 4. Section 6 illustrates how to analyze priority graphs during rule execution to schedule the sequential versus concurrent execution of rules. Section 7 proves the correctness of the IRS algorithm, illustrating how the algorithm guarantees confluence for rules that are allowed to execute concurrently. The paper concludes in Section 8 with a summary of the contribution of the IRS algorithm and a discussion of future research.