Erweiterte ECA-Regeln für verteilte aktive Datenbanksysteme

Thomas Heimrich[1], Günther Specht[2]

[1] TU-Ilmenau, FG Datenbanken und Informationssysteme, 98684 Ilmenau
[2] Universität Ulm, Abteilung Datenbanken und Informationssysteme, 89069 Ulm

Abstract. Zentrale ECA-Regeln wurden für zentrale aktive Datenbanksysteme entwickelt. In verteilten aktiven Datenbanksystemen tritt zusätzlich das Problem der Unerreichbarkeit von Teilsystemen auf und damit die Unentscheidbarkeit von ECA-Bedingungen, die sich auf entfernte Systeme beziehen. In diesem Papier wird daher eine entsprechende Erweiterung der ECA-Regeln für verteilte aktive Datenbanksysteme vorgeschlagen, um auch bei Unerreichbarkeit und Unentscheidbarkeit der ECA-Bedingungen reagieren zu können. Dazu wird zunächst die ECA-Auswertung zu einer strikten Funktion mit dem Unerreichbarkeitszustand Ω erweitert und dann die ECA-Regeln um Alternative Aktionen AA angereichert. Der Einsatz und Vorteil dieser erweiterten ECA-Regeln wird am Beispiel der Erhaltung der Datenkonsistenz in verteilten aktiven Datenbanksystemen gezeigt.

1 Einführung und Motivationsbeispiel

In allen Dienstleistungsunternehmen gewinnt die Außendienstarbeit und damit das verteilte Arbeiten zunehmend an Bedeutung. Um verteiltes Arbeiten zu ermöglichen, sind autonom arbeitende Rechner einschließlich kleinerer Datenbanken nötig. Globale, gemeinsame Daten mit hoher Änderungsfrequenz werden meist zentral gehalten. Während der Arbeit ist oft eine Online-Verbindung zu diesem Datenbestand nötig. Außendienstmitarbeiter oder Teilsysteme benötigen einen Schreib-/Lesezugriff auf Daten entfernter Systeme. Diese Systeme sind unter Umständen nicht erreichbar. Das führt bisher zu einem unbefriedigenden Wartezustand, wenn nach einem Timeout nicht weitergearbeitet werden kann, weil wichtige Informationen fehlen.
Durch die in diesem Artikel vorgestellte Regelerweiterung können verteilte aktive Datenbanken auch bei einer Unentscheidbarkeit der Ereignis- und Bedingungsauswertung, z.B. aufgrund einer Unerreichbarkeit, reagieren und zusätzliche eigene Aktionen feuern. Auch der Abbruch ist eine Aktion, so daß der klassische Fall subsumiert werden kann. Der ECA-Mechanismus gewinnt somit an Mächtigkeit und Robustheit.
Der Rest des Artikels ist wie folgt aufgebaut: Abschnitt 2 stellt nach einer kurzen Einführung in aktive Datenbanksysteme eine formale Erweiterung der ECA-Regeln um strikte Funktionen und alternative Aktionen vor. Danach wird im Abschnitt 3 gezeigt, dass erweiterte ECA-Regeln auch zur Datenkonsistenzerhaltung in verteilten

aktiven Datenbanksystemen genutzt werden können. Abschnitt 4 fasst die wesentlichen Ergebnisse zusammen.

2 Der ECA-Mechanismus und seine Erweiterung für verteilte aktive Datenbanksysteme

Bisherige ECA-Regeln (event/condition/action) werden hauptsächlich in zentralen aktiven Datenbanksystemen genutzt (vgl. [diet00],[pato98],[piva96]). Einfache Varianten, in denen der Event nur aus Insert, Delete und Update bestehen kann, sind in SQL:99 integriert und in einigen objektrelationalen Datenbanksystemen verfügbar. In verteilten aktiven Datenbanken können sowohl die Ereignisauswertung als auch die Bedingungsauswertung unbestimmt sein, wenn Teilsysteme nicht erreichbar sind. Dies führt bisher nach einen Timeout zum Abbruch, auch wenn dies gar nicht nötig oder wünschenswert ist.

2.1 Aktive Datenbanken (eine kurze Wiederholung)

Aktive Datenbanksysteme [diet00] können durch ECA-Regeln auf eintretende Situationen geeignet reagieren. Diese Fähigkeit kann genutzt werden, um Beziehungen zwischen Datenobjekten über Systemgrenzen hinweg zu überwachen.
Reaktionen auf Ereignisse werden dabei durch Regeln spezifiziert. Regeln sind Tripel der Art (Ereignis, Bedingung, Aktion). Andere Bezeichnungen für ECA-Regeln sind auch Trigger oder Alerter.
Ein Ereignis (event) beschreibt ein bestimmtes punktuelles Geschehen. Eine Bedingung (condition) ist ein Prädikat über der Datenbank. Bedingungen legen fest, unter welchen Voraussetzungen man sich für das Auftreten eines Ereignisses interessiert. Bedingungen sind optional und können auch auf true gesetzt werden. Eine Aktion (action) spezifiziert, wie auf die interessierende Situation reagiert werden soll.
Aktive Datenbanken kennen verschiedene Kategorien von Ereignissen. Man unterscheidet primitive und zusammengesetzte Ereignisse. Primitive Ereignisse teilen sich weiter in Datenbankereignisse, Zeitereignisse und abstrakte Ereignisse auf. Datenbankereignisse können beispielsweise Operationen auf der Datenbank oder Transaktionsanfang, -ende, -abbruch sein. Zeitereignisse werden durch das Erreichen eines definierten Zeitpunktes ausgelöst. Abstrakte Ereignisse erlauben es, auf Ereignisse zu reagieren, die außerhalb der Datenbank stattfinden. Allerdings müssen diese Ereignisse dem System explizit mitgeteilt werden. In der Praxis bedeutet das ein explizites Auslösen der Regel durch ein Anwendungsprogramm.
Einfache Ereignisse können zu zusammengesetzten Ereignissen kombiniert werden. Dies geschieht, indem einfache Ereignisse durch logische Operatoren miteinander verbunden werden.

2.2 Anforderungen an den ECA-Mechanismus in verteilten aktiven Datenbanksystemen

2.2.1 Dezentrale Ereigniserkennung

In [piva96] wird eine allgemeine Architektur für heterogene aktive Datenbanksysteme vorgestellt, die eine Ereigniserkennung in verteilten aktiven Datenbanken erlaubt. Die Hauptteile dieser Architektur sind ein zentrales „shared-knowledge Repository“ und eine zentrale Regelbasis. Das shared-knowledge Repository enthält Abbildungsinformationen und Prozeduren. Mit Hilfe dieser Informationen und Prozeduren werden von einem „intelligent agent“ die verschiedenen Datenmodelle, Datenmanipulationssprachen und Objektrepräsentationen der unterschiedlichen Datenbanksysteme integriert. Die systemübergreifenden Regeln werden ebenfalls zentral gehalten. Die lokalen Ereignisdetektoren signalisieren eintretende Ereignisse an eine lokale Komponente. Diese reicht alle Ereignisse, an denen globales Interesse besteht, an die zentrale Regelbasis weiter.
Der Nachteil der beschriebenen Architektur ist der zentralistische Ansatz. Um Ereignisse an die zentrale Regelbasis zu übermitteln, muss eine Verbindung zu dieser bestehen. Entfernt arbeitende Systeme haben diese Verbindung eventuell über längere Zeit nicht. In [piva96] wird eine Pufferung der eingetretenen Ereignisse vorgeschlagen. Der zentrale Ereignisdetektor kann auf diese Ereignisse nur verspätet reagieren. Die verspätete Reaktion kann zu ungewollten Effekten führen, weil sich der Zustand des Datenbanksystems verändert hat. So könnte es vorkommen, dass Attributwerte durch ein verspätetes Update auf einen veralteten Stand gesetzt werden.
Die zentrale Ereigniserkennung ist für ein verteiltes Datenbanksystem mit hohem Autonomiegrad nicht sinnvoll. Für diese Art von Systemen muss eine dezentrale Ereigniserkennung und eine dezentrale Regelbasis gefordert werden.
Der bisherige ECA-Mechanismus und die bisher vorgeschlagenen Architekturen für verteilte heterogene aktive Datenbanksysteme gehen von einer sehr eingeschränkten Autonomie der einzelnen Teilsysteme aus.

2.2.2 Striktheit von ECA-Regeln

In zentrale Datenbanken ist die Auswertung der Ereignisse (event) und Bedingungen (condition) in der ECA-Regel immer möglich. In einem verteilten Datenbanksystem mit hohem Autonomiegrad ist das nicht mehr gegeben. Es ist möglich, dass die Ereignis- und/oder Bedingungsauswertung unbestimmt (Ω) ist. Eine strikte ECA-Regel soll den Fall der Unbestimmtheit berücksichtigen.

2.3 Erweiterung des ECA-Mechanismus für verteilte aktive Datenbanksysteme mit hohem Autonomiegrad

Ziel der Erweiterung ist ein ECA-Mechanismus, der auch bei Unerreichbarkeit von Teilsystemen ein Weiterarbeiten ermöglicht. Dies wird erreicht durch die Striktheit der Ereignis- und Bedingungsauswertung.

Im verteilten aktiven Datenbanksystem kann man ECA-Regeln nach der Art der Bedingung (condition) unterscheiden. Die Bedingungsauswertung von (condition) C sei formal die Funktion *f(C)*, die in den Wertebereich {true, false} abbildet (Gleichung (1)). Bezieht C auch entfernte Teilsysteme mit ein, so sind die Werte der Teilbedingungen aus den entfernten Systemen auch Parameter von *f(C)* (Gleichung (2)). Parameter für *f(C)* können Gleichungen, Ungleichungen, boolsche Werte, sowie die Verknüpfung der genannten Parameter durch boolsche Operatoren sein.

Für die Bedingungsauswertung von C gilt:

$$f(C) \rightarrow \{true, false\} \quad (1)$$

$$f(C)=f(f(ct_l) \text{ and } f(ct_2) \text{ and } \ldots \text{ and } f(ct_n) \text{ and } f(ct_l)) \quad (2)$$

ct_i $(0 \le i \le n)$ Bedingung für das Teilsystem i
ct_l Bedingung für das lokale Teilsystem

In verteilen aktiven Systemen können Teilsysteme zum Zeitpunkt der Bedingungsauswertung unerreichbar sein. $f(ct_i)$ kann demnach in Gleichung (2) unbestimmt sein. Der Definitions- und Wertebereich von *f(C)* wird um das Element Ω (unbestimmt) erweitert, um diesen Sachverhalt abbilden zu können. Das Ergebnis einer strikten Funktion ist Ω, wenn ein Eingabeparameter Ω ist [bawo81].

Es gilt für alle Bedingungen C:

$$f(C) \rightarrow \{true, false, \Omega\} \quad (3)$$

C steht hier für alle Bedingungen, die sich auf das lokale System und/oder entfernte Teilsysteme beziehen. Formal macht die Einführung von Ω aus *f(C)* eine totale Funktion. *f(C)* ist dadurch für alle Funktionsparameter (einschließlich Ω) definiert und liefert immer einen definierten Wert.
Analog zur Bedingungsauswertung kann die Ereignisauswertung als strikte Funktion *g(E)* definiert werden. *g(E)* bildet dann in den Wertebereich {true, false, Ω} ab (Gleichung (4)).

$$g(E) \rightarrow \{true, false, \Omega\} \quad (4)$$

true, wenn das Ereignis E eingetreten ist
false, wenn das Ereignis E nicht eingetreten ist
Ω, wenn unbestimmbar ist, ob das Ereignis E eingetreten ist

Gleichung (5) beschreibt das Auslösen einer Aktion (Feuern einer ECA-Regel).

$$if \{g(E) \wedge f(C)\} \text{ then A ausführen else A nicht ausführen } fi \quad (5)$$

Der Fall, dass einer der Parameter in der *if*-Bedingung Ω ist, führt bisher zur Abarbeitung des *else*-Zweiges. Der Fall einer Unbestimmtheit in der *if*-Bedingung wird bisher nicht explizit berücksichtigt. Wir erweitern daher den ECA-Mechanismus um eine alternative Aktion (AA), die im Ω-Fall ausgeführt wird[1]. Diese kann selbst wieder Ereignisse auslösen.

Erweiterte ECA-Regel:
Eine erweiterte ECA-Regel ist durch die Blöcke *Event, Condition, Action, alternative Action* definiert. *Alternative Action* wird an Stelle der unter Action definierten Reaktion ausgeführt, wenn die Bedingungsauswertung von C unbestimmt (Ω) ist oder die Ereignisauswertung von E unbestimmt (Ω) ist. Eine erweiterte ECA-Regel wird zu einer herkömmlichen ECA-Regel, wenn keine *alternative action* definiert ist.

Die Gleichung (5) sieht damit wie folgt aus:

$if\{g(E) \wedge f(C)\}$	*then*	A ausführen
elseif $if\{g(E) = \Omega \vee f(C) = \Omega\}$	*then*	alternative Aktion ausführen
	else	keine Aktion ausführen
fi		

3 Einsatz von Erweiterten ECA-Regeln zur Erhaltung der Datenkonsistenz

Im Folgenden wird gezeigt, wie erweiterte ECA-Regeln genutzt werden können, um die Datenkonsistenz in einem verteilten aktiven Datenbanksystem zu gewährleisten.

3.1 Spezifikation von Konsistenzbedingungen

Abhängigkeiten zwischen Daten können allgemein durch das D^3 *(data dependency descriptor)* genannte Tupel <S,D,P,C,A> beschrieben werden (vgl. [rusi91]). Dabei steht S für die Quell- und D für die Zielobjekte. Quell- und Zielobjekte können beliebige Datenbankobjekte sein (z.B. Tabellen, Tupel, Attributwerte).
P ist ein Prädikat, das die Datenabhängigkeiten zwischen Quell- und Zielobjekten beschreibt. Auf die ECA-Regeln bezogen kann der Zeitpunkt der Erfüllung von P als ein Ereignis betrachtet werden.
C spezifiziert eine Bedingung, deren Eintreten zur Ausführung der Aktion A führt, kann aber auch einen Zeitpunkt vorgeben, zu dem P erfüllt sein muss. C spezifiziert i.A. keine Konsistenzbedingungen. C sagt also nicht aus, unter welchen Bedingungen die Abhängigkeiten zwischen Quell- und Zielobjekten erfüllt sind (siehe Bsp. unten).

---

[1] Abweichend von der Definition in [bawo81] genügt es, wenn die Fallunterscheidung nur bezüglich der Bedingung und des eintretenden Verzweigungsfalles strikt ist (und nicht global strikt).

A ist eine Aktion, die weitere Aktionen aufrufen kann. A muss durchgeführt werden, um die Konsistenz des Gesamtsystems zu erreichen. Diese Aktion sichert ab, dass P erfüllt wird.
Die Verwendung des Tupels <S,D,P,C,A> soll an einem Beispiel erläutert werden. Die Quellobjekte seien die Attribute s_1 bis s_n, die sich in verschiedenen Datenbanken auf verteilten Rechnern befinden. Diese Rechner können Laptops sein, die nicht permanent erreichbar sind. Das Zielobjekt soll das Attribut *d* sein. Ziel- und Quellobjekte sind in einem konsistenten Zustand, wenn $d >= s_1 + ... + s_n$ (z.B. Eingeplanter Geldbetrag zur Regulierung eines Versicherungsschadens muss größer/gleich der Summe aller Teilschäden sein.) gilt. Diese Konsistenzbedingung soll nur gelten, wenn das Attribut *c* größer 100 ist. Das Attribut *c* kann sich ebenfalls in einer entfernten Datenbank befinden.

Die Notation mit Hilfe des Tupels <S,D,P,C,A> sieht wie folgt aus:

S:	$s_1 , ... , s_n$	Quellkomponenten
D:	*d*	Zielkomponente
P:	$d >= s_1 + ... + s_n$	Konsistenzbeziehung zwischen Quell- und Zielkomponenten
C:	*c>100*	Konsistenzanforderung
A:	$d := s_1 + ... + s_n$	Aktion

In verteilten Datenbanken kann immer der Fall der Unerreichbarkeit von einzelnen Systemen eintreten. Das kann dazu führen, dass im obigen Beispiel P und/oder C unbestimmt sind. Dadurch ist auch unbestimmt, ob die Aktion A ausgeführt werden soll.
Wir erweitern daher das Tupel <S,D,P,C,A> ebenfalls um alternative action (AA), um auch bei Unbestimmtheit von C und/oder P eine definierte Aktion ausführen zu können.
In obigem Beispiel kann eine alternative Aktion z.B. das Attribut *d* auf einen Maximalwert setzten. Die Notation des Beispiels mit Hilfe des Tupels <S,D,P,C,A,AA> sieht wie folgt aus:

S:	$s_1 , ... , s_n$	Quellkomponenten
D:	*d*	Zielkomponente
P:	$d >= s_1 + ... + s_n$	Konsistenzbeziehung zwischen Quell- und Zielkomponenten
C:	*c>100*	Konsistenzanforderung
A:	$d := s_1 + ... + s_n$	Aktion
AA:	*d:=Maximalwert*	alternative Aktion

3.2. Abbildung auf erweiterte ECA-.Regeln

Erweiterte ECA-Regeln können Data Dependency Descriptoren der Form <S,D,P,C,A,AA> direkt auswerten. Die folgende Regel zeigt die allgemeine Abbil-

dung des Tupels <S,D,P,C,A,AA>, auf eine erweiterte ECA-Regel sowie die Abbildung des obigen Beispiels.

Event:	P ist nicht erfüllt	Zeitpunkt, zu dem d >= s_1 + ... + s_n erstmals nicht erfüllt ist.
Condition:	C	*c>100*
Action:	A	d := s_1 + ... + s_n
Alternative Action:	AA	*d :=Maximalwert*

3.3 Vorteile erweiterter ECA-Regeln

Durch die erweiterten ECA-Regeln kann eine Architektur ohne zentrale Ereigniserkennung und Regelbasis erstellt werden. Jedes Teilsystem muss dazu aus einem aktiven Datenbanksystem bestehen. Die einzelnen Teilsysteme müssen Ereignisse auch über die Systemgrenzen hinweg erkennen können.
Jedes Teilsystem kann seine Konsistenzbedingungen in Form von erweiterten ECA-Regeln spezifizieren. So entsteht eine dezentrale Regelbasis. Die Ereigniserkennung ist ebenfalls dezentral, weil jedes Teilsystem auch Ereignisse in entfernten Teilsystemen erkennen kann.
Durch die Erweiterung der ECA-Regeln kann jedes Teilsystem auch bei unbestimmter Ereignis- und Bedingungsauswertung reagieren. Dadurch wird ein hoher Autonomiegrad der Teilsysteme unterstützt. Nur durch die Beachtung der unbestimmten Ereignis- und Bedingungsauswertung kann die Datenkonsistenz in verteilten Datenbanksystemen erreicht werden.

4 Zusammenfassung

Dieser Artikel schlug eine Erweiterung der bekannten ECA-Regeln vor. Herkömmliche ECA-Regeln wurden um das Element *alternative action* erweitert. Die Alternative Aktion wird ausgeführt, wenn die Ereignis- und/oder Bedingungsauswertung unbestimmt ist. Dadurch liefert eine erweiterte ECA-Regel immer eine definierte Reaktion. Herkömmliche ECA-Regeln können dagegen auf eine unbestimmte Ereignis- und/oder Bedingungsauswertung nicht reagieren.
Dies ist insbesondere in verteilten aktiven Datenbanken wichtig, in denen Teile zur Bedingungsauswertung nicht erreichbar sein können. Ein praktischer Einsatz wurde am Beispiel der Erhaltung von Datenkonsistenz gezeigt.

Literaturverzeichnis

[bawo81] Bauer F.L., Wössner H.: *Algorithmische Sprachen und Programmentwicklung*, Springer-Verlag 1981

[conr97] Conrad S.: Förderierte Datenbanksysteme - Konzepte der Datenintegration, Springer-Verlag 1997.

[diet00] Dittrich K. R., Gatziu S.: *Aktive Datenbanksysteme - Konzepte und Mechanismen*, dpunkt.verlag 2000

[hela96] Helal A. A., Heddaya A A., Bhargave B. B.: *Replication Techniques in Distributed Systems*, Kluwer Academic Publishers 1996.

[oezs99] Oezsu M. T., Valduriez P.: *Principles of distributed database systems*, 2nd ed. Prentice-Hall 1999.

[pato98] Paton N. W.: *Active Rules in Database Systems*, Springer-Verlag 1998

[piva96] Pissinou N., Vanapipat K.: *Active Database Rules in Distributed Database Systems*. Intl. Journal of Computer Systems, 11(1), January 1996, pp. 35-44

[rusi91] Rusinkiewicz M., Sheth A., and Karabatis G.: *Specifing interdatabase dependencies in a multidatabase environment*. IEEE Computer, 24(12), December 1991. Special Issue on Heterogeneous Distributed Databases, pp. 46-53.

[zimm01] Zimmermann J.: *Konzeption und Realisierung eines aktiven Datenbanksystems: Architektur, Schnittstellen und Werkzeuge*, Logos-Verl, 2001