Fact-Oriented Business Rule Modeling in the Event Perspective

Peter Bollen

Department of Organization & Strategy, University of Maastricht
P.O. Box 616, 6200 MD, the Netherlands
p.bollen@os.unimaas.nl

Abstract. In this article we will focus on the business rule modeling constructs and methodology in the fact-oriented approach for the third perspective from the IFIP CRIS framework: the *behaviour-oriented perspective* or *event perspective.*

1 Introduction

In order to define the fact-oriented modeling constructs for the event perspective, we will formalize the definition of derivation rules as they are currently used in fact-oriented modeling languages (e.g ORM [1]). The derivation rules that constitute an application's process base can be fully formalized whenever an appropriate references structure for the (static) information grammar has been put in place. We note how the pre-condition references the object types that are specified in the process argument and possibly references the (ingredient) fact types that must be contained in the application information grammar. We note that the post-condition specifies what the result will be of the execution of the derivation rule, whenever the pre-condition evaluates to true. The *create* operator is defined as follows: a fact instance that will be 'created' has subsequently to be inserted to the application's information base. If the projected information base after the proposed insert transaction will violate the application's conceptual schema or information grammar, a created fact will **not** be added to the application's information base. The rule-body of the derivation rule contains the explicit derivation logic that 'computes' the value(s) for the 'derived' role for the fact instance(s) that will be created.

Dr1: Derive credibility status <(arg1, customer)>

IF	There exists an instance of Ft2 SUCH THAT FT2.R2 = 'arg1'
AND	There exists at least one instance of Ft3 (where ' FT3.R1' is SUCH THAT there exist an instance of Ft2 SUCH that Ft2.R2 =' arg1') [pre-condition]

THEN	Create an instance of fact type Ft1 SUCH THAT Ft1.R2= 'arg1' AND Ft1.R1= DRbody1 [post-condition]

DRbody1:=	IF there exists at least one instance of Ft3 SUCH THAT Ft3.R2 = 'bad' THEN ' not credible' ELSE 'credible' [rule body]

2 The extension of ORM with Event-Condition-Action (ECA) modeling constructs

Although the execution of the derivation rule is constrained by the *pre-conditions* and *post-conditions*, there still remain degrees of freedom with respect to *when* and in *what sequence* these derivation rules or *information base update processes* (IBUPs) can be executed. Therefore, an additional modeling construct is needed, to specify ***when*** the instances of derivation rules from the *conceptual schema* will be executed. These 'rule' executions will be triggered by events. For example the occurrence of an event instance that an *insurance application* is created will 'trigger' the derivation rule: *derive customer credibility*:

```
ON     insurance application is created
THEN  derive customer credibility
```

Definition 1. An *event type* is a set of events in the application subject area, each of these events can lead to the execution of one or more derivation rules.

Definition 2. An *event type argument set* of a given event type specifies all occurrences of object types, instances of which should be supplied for an event instance of the event type.

An *event* can start the execution of a derivation rule or IBUP (in some cases) under (a) condition(s) on the information base. In the population constraints from the application information model we have modeled the 'invariant' business rules that must hold for every information base state. For example the business rule that states that *every insurance application must state the insurance type*. In the pre-condition of the derivation rule(s), the business rules are modeled that specify what ingredient fact instances should be available in order to 'compose' or 'derive' the resulting fact instance(s) in the derivation rule [2: p.1519]. In the event perspective we will model the business rules that contain the knowledge under what condition (on the application information base) an event of an event type will trigger a specific derivation rule or IBUP. An example event description for the insurance application example will look as follows:

```
ON E1:insurance application is created (arg1:application)
THEN  derive customer credibility (arg1:customer)

ON E2: new day(arg1: date, arg2: month)
IF C1: (E2.arg1= '1' AND E2.arg2= 'january')
THEN   derive customer credibility (arg1:customer)
```

Definition 3. A *guard condition* is a proposition on the information base.

The proposition in the guard condition can contain a reference to one or more instances of the event argument.

3 The Impulse Mapper

In many cases the derivation rules are executed by users from different user groups in the same organization. The *external schema* for the event perspective for such a user group might contain *compound impulses*. This means that an event will trigger two or more derivation rules at the same time. In order to abstract from externally imposed "ways of working" we will have to atomize these *compound impulse types*. Each *atomic impulse* containing exactly **one** (primitive) *event* [3], exactly **one** *derivation rule* or *IBUP* and the condition under which the derivation rule or IBUP will be executed. We will call the effect of an event occurence into the execution of one derivation rule or IBUP (eventually under a condition on the information base) an *impulse* (instance). It is this definition of an impulse that allows us to look at an impulse as a specific type of 'business constraint' (see the discussion in Bollen [4: p.112-113]) without having to worry about run-time implementation issues like *code generation* [5], *message sending* [3: p.132] and *software components* (e.g. *event handler* [6]).

```
Algorithm 1:Fact-oriented behavioral modeling procedure
BEGIN    Take the first user group in application subject area
 WHILE    still user groups left in
         Take the first derivation rule or IBUP from conceptual schema
         WHILE   still derivation rules/IBUP's in conceptual schema.
          Ask the users in the Sphere of Influence what event type(s)
          invoke such a derivation rule or IBUP
          Check whether such an event type is already listed.
         IF event type not listed
          THEN For each event type determine the event type argument
          ELSE For each event type that evokes such a process:
               determine the condition on the IB and event
               argument under which the der. Rule is instantiated.
               IF the condition is different from an existing
                  condition on the same event type and derivation
                  rule/IBUP
               THEN Make a combined condition which contains the
                    old condition type and the new condition type
               ELSE  the impulse is already defined.
               ENDIF
               For each (relevant) impulse determine the impulse mapper
               IF parts of such an impulse mapper can not be determined
               THEN redefine the part of the event argument such that an
                  impulse mapper can be defined
               ENDIF
                 For each impulse define the event-condition
                 and the condition-process trigger type (if relevant)
        ENDIF
          take next derivation rule/IBUP
      ENDWHILE
       take next user group in sphere of influence
  ENDWHILE
  END
```

Events that do not have the potential to 'trigger' derivation rules from the application's conceptual schema or IBUP's are not relevant for the description of the

behavioural perspective in a given application subject area [7 : p.3]. We can now classify all impulses that have the same *event type*, the same *derivation rule or IBUP* and the same *condition type* into a set of impulse instances that belong to the same *impulse type*.

An impulse type contains an event type, a condition type, and a derivation rule (or a conceptual process type in general) or IBUP.

Definition 4. An *impulse mapper* is a construct that transforms values of event type arguments and fact instances from the application information base into instantiation values for the *argument set(s)* for the derivation rule or IBUP.

Example:	
Event type	*Et1: insurance application created (arg1: application).*
Derivation rule	*Dr1: determine customer credibility (arg1:customer).*
Guard Condition type	*C1 : Ft4.R1 ='car' (where FT4.R2='E1.arg1')*
Impulse mapper	*Dr1.arg1:=Ft2.R2 (where Ft2.R1='E1.arg1'*

In algorithm 1 the modeling procedure for deriving the business rule model in the behaviour-oriented perspective is given.

4 Conclusions

In this article we have generalized the declarability of business rules in the data-oriented an process-oriented perspectives to the event perspective, by defining the basic ECA oriented modeling constructs and the concept of *impulse mapper* that provides the semantic connection between the model in the event perspective on one hand and the models in the process- and data-oriented perspectives on the other hand. In addition an explicit modeling procedure was provided.

References

1. Halpin, T., *Information Modeling and Relational Databases; from conceptual analysis to logical design*. 2001, San Francisco, California: Morgan Kaufmann.
2. Bollen, P., *Conceptual process configurations in enterprise knowledge management systems*, in *Applied computing 2006*. 2006, ACM: Dijon, France.
3. Bassiliades, N. and I. Vlahavas, *Processing production rules in DEVICE, an active knowledge base system.* Data & Knowlege Engineering, 1997. 24: p. 117-155.
4. Bollen, P., *On the applicability of requirements determination methods*, in *Management and Organization*. 2004, University of Groningen: Groningen. p. 219.
5. Dietrich, S.W., et al., *Component adaptation for event-based application integration using active rules.* Journal of Systems and Software, (in press).
6. Pissinou, N., K. Makki, and R. Krishnamurthy, *An ECA object service to support active distributed objects.* Information Sciences, 1997. 100: p. 63-104.
7. Paton, W., . ed. *Active rules in database systems*. Monographs in computer science, ed. D. Gries. 1999, Springer: New-York.