

# Twelve Theses on Reactive Rules for the Web

François Bry and Michael Eckert

Institute for Informatics, University of Munich, Germany

Workshop 'Reactivity on the Web'  
Munich, 31st March 2006

*Reactivity*, the ability to detect and react to events, is essential in many information systems.

Many Web information systems such as

- online marketplaces,
- adaptive systems (e.g. recommender and eLearning systems), and
- Web services,

react to events.

# Contents

- 1 The Need for ECA Rules on the Web
- 2 Event Communication Paradigms
- 3 Specifying Composite Events: Towards High-Level Event Query Languages
- 4 Specifying Conditions: Embedding a Web Query Language
- 5 Specifying State-Changing Actions
- 6 Structuring Rules and Rule Programs
- 7 Miscellanea

# The Need for ECA Rules on the Web

## Thesis 1

High-level reactive languages are needed on the Web. Event-Condition-Action (ECA) rules are well-suited to specify reactivity on the Web. They are better suited than production rules for a large class of Web applications.

HTTP provides an infrastructure for *exchanging events* or messages.

SOAP provides conventions for exchanging *structured and typed information* on the Web as XML messages.

Complementing HTTP and SOAP with *high-level languages for updates and reactivity* is needed for both standard Web and Semantic Web applications.

On the Web, high-level reactive languages are needed that

- abstract away network communication and system issues,
- ease the specification of complex updates of Web resources (e.g., XML, RDF, and OWL data),
- are convenient for specifying complex flows of actions and reactions.

On the Web, ECA rules, are more appropriate than production rules without explicit reference to events:

- Real-world reactive rules often come with an explicit event.
- Events exchanged as messages between Web nodes are a natural communication paradigm.
- Events can carry data between Web nodes that is relevant for the condition and action part of a rule.
- ECA rules allow an easy handling of errors and exceptional situations that can conveniently be expressed as (special) events.

## Thesis 2

Reactive Web rules should be processed locally and act globally through event-based communication and access to persistent Web data.

Approaches assuming a central rule processing entity are not suitable for the Web's highly distributed and loosely coupled architecture.

Global behavior can be achieved by using event-based communication.

Reactive rules should be able to access data from anywhere on the Web.

# Event Communication Paradigms

## Thesis 3

Events are best exchanged directly between Web sites in a push manner.

Because of the Web's decentralized architecture, events must be exchanged directly, point-to-point, between Web sites.

Events are best sent in a push manner from the Web sites where the event originates to other Web sites. Periodical polling is less favorable.

# Specifying Composite Events: Towards High-Level Event Query Languages

## Thesis 4

Events are volatile data and should be kept distinct from persistent data.

Standard data on the Web are retrieved upon request in a pull manner, are persistent, and can be modified. It typically signifies a state of the world.

Events are communicated between Web nodes (typically in a push manner), volatile, and not modifiable. They are typically used to signal state changes.

Persistent data is like written text. Event data is like spoken words.

## Thesis 5

Recognizing composite events is essential for a reactive Web language.

Composite events are conveniently specified by (event) queries.

There are (at least) four complementary dimensions to event queries:

- data extraction,
- event composition,
- temporal conditions, and
- event accumulation.

In a carefully developed application, well-chosen atomic events might suffice. On the Web, different applications may have to cooperate. Situations which have not been considered in an application's design must then be inferred from several atomic events.

Composite events as patterns of events do not exist explicitly “by themselves”. Rather they are implicit and the patterns are conveniently specified by event queries.

There are at least four complementary dimensions to an event query language:

- 1 *Extraction*: Event messages contain data that is relevant to whether and how to react.
- 2 *Event composition*: Event queries must support composition constructs such as the conjunction, disjunction, and negation of events (more precisely of event queries).
- 3 *Temporal conditions*: Event queries must be able to express temporal conditions such as “*events A and B happen within 1 hour and A happens before B.*”
- 4 *Event accumulation*: Event queries must be able to accumulate events, to aggregate data, and to detect repetitions, e.g.:  
“*the average over the last 5 reported stock prices raises by 5%*”  
“*3 server outages have been reported within 1 hour.*”

## Thesis 6

A data-driven, incremental evaluation of event queries is the approach of choice.

A natural evaluation of standard queries is query-driven.

A natural evaluation of event queries is data- i.e. event-driven.

# Specifying Conditions: Embedding a Web Query Language

## Thesis 7

Data from persistent Web resources plays an essential role for Web reactivity. A reactive language thus should embed or build upon a Web query language.

The natural place to deploy a Web query language in ECA rules is the C (condition) part. It should also be used to query data in atomic events in the E (event) part of ECA rules.

## Criteria to consider:

- What is the query language's notion of answers?
- Can answers be used to “parameterize” further queries or the action?
- What evaluation methods for queries are possible?
- Which data models are supported (XML, RDF, OWL)?
- How does the query language deal with identity?
- Which reasoning or deductive capabilities does the query language provide?

# Specifying State-Changing Actions

## Thesis 8

The Web is a dynamic, state-changing system. Reactions to state changes (events) through reactive rules are state-changing actions such as updates to persistent data. Reactive rules are needed where compound actions can be constructed from primitive actions.

# Structuring Rules and Rule Programs

## Thesis 9

Development and maintenance of reactive rule programs can be considerably supported by structuring mechanisms such as:

- branching in rules,
- deductive rules for event queries and Web queries,
- procedural abstractions for actions, and
- grouping of rules.

# Miscellanea

## Thesis 10

Identity of data items is an issue for reactive languages due to their ability to react to changes of data objects on the Web.

Monitoring data items (or objects) and react to their changes need to be aware of their identity.

Two approaches:

- *Extensional Identity*: identity defined in terms of an object's structure or value (its "extension").
- *Surrogate (or Object) Identity*: identity defined independently from an object's extension as an external surrogate.

## Thesis 11

Meta-programming and meta-circularity, i.e. the ability to use rules to exchange and evaluate (other) rules, are needed in some important cases.

*Policy-based trust negotiations* require meta-programming.

## Thesis 12

Reactivity in the Web's open and uncontrolled world requires language support for authentication, authorization, and accounting.

Service accounting often leads to a “double reactivity:”

- The service itself is in general reactive.
- the accounting service reacts to uses of the reactive service.

Acknowledgments to members of REVERSE, of the W3C Rule Interchange Format (RIF) Working Group, and to Tim Furche, Paula-Lavinia Pătrânjan, and Inna Romanenko.

Thank you!