The Reaction Algebra:
A Formal Language for Event Correlation*

César Sánchez¹, Matteo Slanina², Henny B. Sipma¹, and Zohar Manna¹

¹ Computer Science Department, Stanford University, Stanford, CA 94305-9025
{cesar,sipma,zm}@CS.Stanford.EDU
² Google Inc., 1600 Amphitheatre Pkwy, Mountain View, CA 94043
mslanina@google.com

⋆⋆ Current affiliation. This work was done while at Stanford University.

Abstract. Event-pattern reactive programs are small programs that
process an input stream of events to detect and act upon given tem-
poral patterns. These programs are used in distributed systems to notify
components when they must react.

We present the reaction algebra, a declarative language to define finite-
state reactions. We prove that the reaction algebra is complete in the fol-
lowing sense: every event-pattern reactive system that can be described
and implemented – in any formalism – using finite memory, can also be
described in the reaction algebra.

1 Introduction

Interactive computation [6] studies the interaction of computational devices,
including reactive and embedded systems, with their (not necessarily computa-
tional) environment. The most common approach to study interactive computa-
tion is based on machine models such as automata and Turing machines, enriched
with output. In this paper we offer a complementary perspective: the reaction algebra,
a declarative language to describe finite-state reactions. Its relationship
to the machine models is similar to the relationship of regular expressions to
language acceptors.

The practical motivation for a formalization of event-pattern reactive pro-
grams is to offer developers of distributed reactive systems a declarative way
to describe temporal reaction patterns that is both formal and practical. The
advantage of this design approach is that the interaction between components
is made explicit and separate from the application code and can hence be an-
alyzed independently. In addition, the code for pattern detection and reaction
can be generated automatically from the event-pattern expressions and can be
optimized for different objectives, including minimum processing time per event
or smallest footprint.

⋆ This research was supported in part by NSF grants CCR-02-20134, CCR-02-09237,
CNS-0411363, CCF-0430102, and CSR-0615449, and by NAVY/ONR contract
N00014-03-1-0939.

A. Avron et al. (Eds.): Trakhtenbrot/Festschrift, LNCS 4800, pp. 596–619, 2008.
© Springer-Verlag Berlin Heidelberg 2008
**Event-Pattern Reactive Programming.** In recent years the publish/subscribe architecture has become popular in the design of distributed reactive systems. In this architecture, components communicate with each other by events via an event channel. Components publish events to the event channel that may be of interest to other components. Components can also subscribe to the event channel to express interest in receiving certain events. The objectives of the publish/subscribe architecture are flexibility and scalability. Components are loosely coupled and may be added and removed on the fly and activated only when relevant events happen.

Most modern distributed systems are built on a middleware platform, a software layer that hides the heterogeneity of the underlying hardware, offers a uniform interface to the application, and usually provides services that implement common needs. Many middleware platforms provide an event channel that supports the publish/subscribe architecture. There are differences, however, in what kind of subscriptions are supported. Most platforms, including Gryphon [1], Ace-Tao [2], Siena [4], and Elvin [25], support simple “event filtering”: components can subscribe with a list of event types and the event channel notifies the component each time an event of one of those types is published. A slightly more expressive mechanism is “event content filtering”, in which components in their subscriptions can specify predicates over the data included in the event. Notification, however, is still based on the properties of single events.

A more sophisticated subscription mechanism is “event correlation”, which allows subscriptions in the form of temporal patterns. A component is notified only when a sequence of events that satisfies one of the patterns has been published. An implementation of this mechanism must maintain state: it may have to remember events it observed and may even have to store events that may have to be delivered to a component at a later stage. Event correlation is attractive because it separates the interaction logic from the application code and reduces the number of unnecessary notifications. Separation of the interaction logic increases analyzability. It also allows reuse of pattern detection code, thereby simplifying the development of applications. However, providing event correlation as a service requires that it have an intuitive, easy to use description language with a well-defined semantics. The reaction algebra, presented in this paper, aims to provide such a language.

**Example 1.** Fig. 1 shows an example of a small avionics system. It consists of six components that all communicate with the event channel. The purpose of the system is to control the cockpit’s display such that it shows relevant information according to the current mode of operation, in this case tactical mode and navigational mode. In tactical mode, the Tactical Steering (TS) component collects data from the sensors and publishes events with tactical information to be displayed; in navigational mode the Navigational Steering (NS) component collects the data and performs the calculations. The mode of operation is set by the pilot via the Pilot Control component, which publishes an event to the event channel each time the mode is switched.