

Final Semantics for Event-Pattern Reactive Programs

César Sánchez, Henny B. Sipma, Matteo Slanina, and Zohar Manna*

Computer Science Department,
Stanford University, Stanford, CA 94305-9025
{cesar, sipma, matteo, zm}@CS.Stanford.EDU

Abstract. Event-pattern reactive programs are front-end programs for distributed reactive components that preprocess an incoming stream of event stimuli. Their purpose is to recognize temporal patterns of events that are relevant to the serviced program and ignore all other events, outsourcing some of the component's complexity and shielding it from event overload. Correctness of event-pattern reactive programs is essential, because bugs may result in loss of relevant events and hence failure to react appropriately.

We introduce **PAR**, a specification language for event-pattern reactive programs. We propose a new approach for defining such languages in terms of observations and actions. This approach applies standard techniques from coalgebra to obtain instances of the corecursion and coinduction principles. Corecursion is used to formally define the operational semantics of **PAR**, and coinduction allows to prove general equivalences between (ground and parameterized) **PAR** programs.

This is the first of a series of papers in which we study questions of expressive completeness, complexity, and formal verification techniques for specification languages of event-pattern reactive programs.

1 Introduction

Reactive programs are software components that maintain an ongoing interaction with their environment. With the introduction of middleware technologies and the emphasis on component-based systems, this interaction is increasingly performed through *events*. Reactive components, which can range from simple sensors to sophisticated monitors or controllers, operate relatively autonomously and communicate using events. This gives rise to *publish-subscribe* architectures, in which producer components publish events to the middleware and consumer components subscribe with it to receive relevant events.

Different subscription policies are possible. The simplest uses a list of event types and/or senders that can be syntactically matched by an attribute in the event. This is known as *event filtering* and is available in most popular platforms,

* This research was supported in part by NSF grants CCR-01-21403, CCR-02-20134, CCR-02-09237, CNS-0411363, and CCF-0430102, by ARO grant DAAD19-01-1-0723, and by NAVY/ONR contract N00014-03-1-0939.

including GRYPHON [1], ACE-TAO [18], SIENA [3], and ELVIN [19]. A more sophisticated policy is *content filtering*, in which the subscription contains a list of predicates on the data of the event. This approach is especially popular in active databases and stock market applications. With these policies every single event is either discarded or dispatched, independently of the event history. Another extension of event filtering, orthogonal to content filtering, is *event correlation*, the approach studied in this paper. Here, subscriptions may contain temporal patterns of either attributes or content predicates on events.

Event correlation is attractive for several reasons: it may substantially reduce unnecessary component activations, thereby improving the performance; it separates event-pattern recognition from event processing, thus increasing analyzability of component interactions; it allows automatic synthesis of the pattern recognition code, thus reducing ad-hoc implementations and improving reusability. At present, some middleware platforms provide limited forms of event correlation services. Unfortunately, formal semantics are not given, making their use risky: unclear semantics or incorrect implementations may result in the loss of important events, potentially causing failure to respond to critical situations.

In a previous paper [16] we introduced ECL, a language to specify event correlation patterns, developed under the DARPA PCES program for the Boeing Boldstroke [20] platform to support mission-critical avionics applications. We gave a formal semantics in terms of correlation machines, an extension of finite-state transducers that enabled direct translation into event-processing code. Prototype implementations were integrated in ACE-TAO [18] and FACET [9].

In this paper we shift focus from implementation to analysis, in particular program equivalences. Practical applications need to determine whether a given pattern expression can be replaced by a simpler one, or merged with that of another component, without affecting its behavior. Correlation machines, however, are not well suited to answer these questions, since they explicitly model operational details, such as parallelism, into the semantics. Instead, we are interested in behavioral equivalences, in which two programs are considered equivalent if they exhibit the same notification behavior.

We intend to study languages for event-pattern reactive programming algebraically, influenced by the pioneering work on languages for the study of concurrency, mostly process algebras [13,8,2] and Hennessy-Milner logic [7]. The main difference is that we specifically design our languages to be deterministic, because we want to synthesize executable behaviors from the expressions, while every reasonable concurrency model is intrinsically nondeterministic.

Coalgebra is a convenient framework to study dynamic systems, and, in general, systems with hidden state spaces, where only the observable behavior is of interest [10]. For example, in [15] Rutten shows how equivalence of regular expressions can be analyzed in this framework. He constructs an automaton, whose states correspond to languages, that is final with respect to all other automata; then he shows that language equivalence can be reduced to proving bisimilarity of their corresponding states in this final automaton. In this paper we develop a