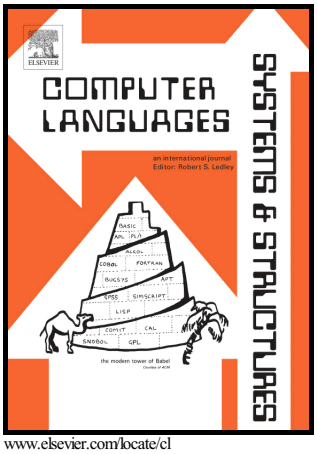

www.elsevier.com/locate/cl

Programming with Event Loops and Control Loops - From Actors to Agents

Alessandro Ricci

PII: S1477-8424(15)00094-9
DOI: http://dx.doi.org/10.1016/j.cl.2015.12.003
Reference: COMLAN204

To appear in: *Computer Language*

Received date: 17 May 2015
Revised date: 15 November 2015
Accepted date: 15 December 2015

Cite this article as: Alessandro Ricci, Programming with Event Loops and Control Loops - From Actors to Agents, *Computer Language* http://dx.doi.org/10.1016/j.cl.2015.12.003

Programming with Event Loops and Control Loops - From Actors to Agents

Alessandro Ricci

DISI – University of Bologna, Italy
mailto: a.ricci@unibo.it

Abstract

Event loops are a main control architecture to implement actors. In this paper we first analyse the impact that this choice has on the design of actor-based concurrent programs. Then, we discuss *control loops* as the main architecture adopted to implement agents, and we frame them as an extension of event loops effective to improve the programming of autonomous components that need to integrate both reactive and proactive behaviours, in a modular way.

Keywords: Event loops, control loops, concurrent programming, actors, agents, agent-oriented programming

1. Introduction

Event loops are a control architecture pervasively adopted to govern the behaviour of applications, in particular in those context where reactivity is an important aspect. Modern examples include Rich Internet Applications, based on HTM5 and JavaScript, and mobile applications, based on e.g., the Android platform. In this architecture, the control flow of an application is logically organized as an infinite loop waiting for events on an event queue. As soon as one event is available, it is fetched and a corresponding event handler is executed, if available. When the event handler execution is terminated, the control flow goes back waiting for the next event.

In the case of *actors* [4, 5, 27], the event loop architecture is brought down to the computational model of the basic first-class abstractions adopted to design the active part of programs, so that a system or application is organized in terms of a possibly large number of active entities whose execution