Computer Languages, Systems & Structures 39 (2013) 67–94

Contents lists available at ScienceDirect

Computer Languages, Systems & Structures

journal homepage: www.elsevier.com/locate/cl

MONACO—A domain-specific language solution for reactive process control programming with hierarchical components

Herbert Prähofer*, Roland Schatz, Christian Wirth, Dominik Hurnaus, Hanspeter Mössenböck

Johannes Kepler University Linz, Christian Doppler Laboratory of Automated Software Engineering, Institute for Systems Software, Altenbergerstrasse 69, A-4040 Linz, Austria

ARTICLE INFO

Article history:
Received 7 March 2012
Received in revised form
21 September 2012
Accepted 25 February 2013
Available online 16 March 2013

Keywords:
Domain-specific languages
Automation control
Reactive programming
Component-based systems

ABSTRACT

In this paper, we present MONACO – a domain-specific language for developing event-based, reactive process control programs – and its visual interactive programming environment. The main purpose of the language is to bring process control programming closer to domain experts. Important design goals have therefore been to keep the language concise and to allow programs to be written that reflect the perceptions of domain experts. MONACO is similar to Statecharts in its expressive power, but adopts an imperative notation. Moreover, MONACO uses a state-of-the-art component approach with interfaces and polymorphic implementations, and enforces strict hierarchical component architectures that support hierarchical abstraction of control functionality. We present the main design goals, the essential programming elements, the visual interactive programming environment, results from industrial case studies, and a formal definition of the semantics of the reactive behavior of MONACO programs in the form of labeled transition systems.

1. Introduction

The demand for languages and tools that support domain experts in implementing process control solutions is increasing [1–3]. In industrial automation, domain experts commonly extend and adapt their control solutions to fulfill the specific requirements at hand. They must intervene in safety-critical, highly dependable systems and are often expected to alter programs while the machine is in operation and to make these changes effective at run time using online-change capabilities. However, domain experts usually lack "deep" software engineering skills and expertise, requiring programming environments to provide extensive support, guidance, and supervision.

Domain-specific languages (DSLs) [4] are a proven approach to bringing programming closer to application domains. They aim to present software in the notations of domain experts and allow a straightforward mapping of application concepts to software solutions. Many domain-specific languages and modeling approaches have emerged in the automation domain [5–8]. For example, domain-specific modeling systems for specifying control behavior in the form of *function block diagrams* [8–10] have been very successful and can be considered state-of-the-art. Specifying reactive behavior, however, has proved to be more challenging. In this context, the Statecharts formalism [11,12], and its derivatives (e.g. [13,14]) are widely used for expressing complex reactive system behavior, but these modeling approaches target software engineering experts rather than domain experts.

Therefore, we have developed a new DSL called MONACO (Modular NOtation for Automation COntrol), the goal of which is to allow implementation of event-based, reactive process control programs in a concise and intuitive manner. The context of this work is a collaboration with Keba AG (www.keba.com), a medium-sized company developing and producing hardware and software

* Corresponding author. Tel.: +43 732 2486 4352.
E-mail addresses: herbert.praehofer@jku.at, herbert.praehofer@gmail.com (H. Prähofer).

1477-8424/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cl.2013.02.001

platforms and solutions for industrial automation. In their practice, development of automation solutions is a multi-stage process involving various stakeholders at different stages of the automation process with varying levels of programming knowledge and capabilities:

- Keba develops and produces a hardware and software platform with associated tool support.
- The hardware and software platforms enable Keba customers (mainly OEMs of manufacturing machines) to realize automation systems for their products. Employees of OEMs, however, are often domain experts with limited software engineering capabilities.
- OEMs also build customizable software systems and electronic control panels for use by machine operators who act as end user programmers.

The Monaco language is specialized to a rather narrow subfield of the automation domain, that is, programming control sequence operations of manufacturing machines. This narrow domain includes all types of automated machines, but excludes bigger automation systems such as whole manufacturing plants. Within the numerous layers of automation systems Monaco is designed to address the layer of event-based control of machine operations between the lower-level continuous control and signal processing layer and the higher-level manufacturing execution layer.

Monaco has been designed to allow the programming of sequences of control operations, to enable parallel activities, and to provide strong support for dealing with exceptions. In its expressive power, Monaco is therefore similar to Statecharts, but adopts an imperative notation similar to other languages in the domain [15]. Most importantly, however, Monaco's distinguishing feature is its *hierarchical component approach*, which allows building automation solutions in an arrangement of upper and subordinate components. Thus, it supports abstraction of reactive control behavior in several hierarchical layers and building reusable components. Hierarchical abstraction of control tasks results in simplifications that bring control programs closer to the perceptions of domain experts.

1.1. Contributions

This paper makes the following contributions:

- It introduces Monaco as a novel programming language with a state-of-the-art component approach for programming reactive control systems. We show that its expressive power is similar to that of Statecharts. Monaco, additionally combines reactive system programming elements with an imperative programming notation and approved language concepts from structured programming.
- It shows that the Monaco language allows implementing control programs in a concise way. In particular, we show how the aligned language features of Monaco facilitate hierarchical abstraction of control functionality.
- We show that Monaco's small set of language elements is both expressive and intuitive. In particular, we demonstrate that Monaco is capable of expressing complex reactive control functionality in a way that coincides closely with domain expert perceptions.
- We show a simple visualization scheme and the interactive development environment for Monaco programs which makes Monaco control programs even more appealing to domain experts.
- We give a formal definition of the semantics of the reactive behavior of Monaco programs in the form of labeled transition systems.

The Monaco language and its visual notation were first published in [16,17], respectively. This paper subsumes and extends this work by providing a detailed presentation of the language, giving a formal definition of the semantics of the language, and presenting results from implementation and case studies.

1.2. Paper outline

The outline of this paper is as follows: Section 2 discusses principal ideas for the design of the language. Section 3 introduces the main language elements of Monaco. Section 4 demonstrates how hierarchical control programs can be built using an example control program for an injection molding machine. In Section 5 the interactive visual programming environment is presented. In Section 6 we give a formal definition of the reactive behavior of Monaco programs. Section 7 discusses implementation and results from industrial case studies. Section 8 compares the approach to related work, and Section 9 concludes with a summary and an overview of follow-up work.

2. Main ideas

In discussions with domain experts at our industrial partners, we learned how they conceptualize automation machines. From these findings, we derived the following language design features:

Imperative notation: The main motivation for designing a new notation for event-based control systems instead of using, for example, the well-established Statecharts formalism was the observation that domain experts think in sequences of control tasks and their coordination rather than in states and state transitions. Further, we observed that in normal