Available online at www.sciencedirect.com
SCIENCE DIRECT®
Theoretical Computer Science 331 (2005) 367–396
Theoretical Computer Science
www.elsevier.com/locate/tcs

A calculus for reasoning about software composition

Franz Achermann, Oscar Nierstrasz*

Software Composition Group, University of Bern, Switzerland

Abstract

Although the term *software component* has become commonplace, there is no universally accepted definition of the term, nor does there exist a common foundation for specifying various kinds of components and their compositions. We propose such a foundation. The Piccola calculus is a process calculus, based on the asynchronous π-calculus, extended with *explicit namespaces*. The calculus is high level, rather than minimal, and is consequently convenient for expressing and reasoning about software components, and different *styles* of composition. We motivate and present the calculus, and outline how it is used to specify the semantics of Piccola, a small composition language. We demonstrate how the calculus can be used to simplify compositions by partial evaluation, and we briefly outline some other applications of the calculus to reasoning about compositional styles.

Keywords: Software components; Process calculi; Software architecture

1. Introduction

Component-Based Software Development (CBSD) offers us the promise of flexible applications being constructed from stable, robust software components. But how are components plugged into an application? How do we specify the way in which components are configured and composed?

In addition to components, we clearly need some means to specify compositions of components. A *composition language* [39] is a language for specifying operators for connecting

* Corresponding author.
E-mail address: oscar.nierstrasz@acm.org (O. Nierstrasz).

0304-3975/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2004.09.022

components (i.e., "connectors"), *glue* abstractions for adapting component interfaces, and *scripts* that instantiate and connect components. Piccola [4,6] is a prototype for such a composition language, and JPiccola is an implementation which targets the composition of Java software components [38].

A key challenge for a composition language is to offer a means to answer the question, *What, precisely, do we mean by composition*? There are many different notions of component composition and interconnection in practice, so a composition language must offer a neutral foundation in which different forms of composition can be expressed. We therefore need a *semantic foundation* for specifying compositional abstractions. Components may be configured and adapted in many different ways, which may or may not have an impact on the resulting composition. We therefore also need to reason about *equivalence* of different expressions of composition.

Drawing from our experience modeling various component models, we have developed the Piccola calculus as a tool for expressing the semantics of software composition and for reasoning about equivalence of compositions. The Piccola calculus extends the asychronous π-calculus [32,45] with *forms*—first-class, extensible namespaces [5]. Forms are not only convenient for expressing components, but play other important roles as well.

This calculus serves both as the semantic target and as an executable abstract machine for Piccola. In this paper we first motivate the calculus by establishing a set of requirements for modeling composition of software components in Section 2. Next, we address these requirements by presenting the syntax and semantics of the Piccola calculus in Section 3. In section 4 we provide a brief overview of the Piccola language, and summarize how the calculus helps us to define its semantics. In Section 5, we show how the calculus helps us to reason about Piccola compositions and optimize the language bridge by partial evaluation while preserving its semantics. Finally, we conclude with a few remarks about related and ongoing work in Sections 6 and 7.

2. Modeling software composition

We take as our starting point the view that

$$\text{Applications} = \text{Components} + \text{Scripts},$$

that is, component-based applications are (ideally) made up of stable, off-the-shelf components, and scripts that plug them together [6]. Scripts (ideally) make use of high-level connectors that coordinate the services of various components [3,36,52]. Furthermore, complex applications may need services of components that depend on very different architectural assumptions [18]. In these cases, *glue code* is needed to adapt components to different architectural styles [50,51].

A foundation for modeling software components must therefore be suitable for expressing compositional styles, scripts, coordination abstractions and glue code. Fig. 1 summarizes the requirements, and illustrates how Piccola and the Piccola calculus support them.