Available online at www.sciencedirect.com
SCIENCE DIRECT®
Science of Computer Programming 56 (2005) 117–139
Science of Computer Programming
www.elsevier.com/locate/scico

Hierarchical composition of industrial components

Ursula Scheben

FernUniversität Hagen, Lehrstuhl für Programmiersysteme, D-58097 Hagen, Germany

Received 20 November 2003; received in revised form 16 August 2004; accepted 6 September 2004
Available online 22 December 2004

Abstract

In this paper we focus on the hierarchical composition of instances of arbitrary industrial component models yielding new (compound) components with specified capabilities and requirements which can themselves be composed to yield higher level components. For this purpose special component interfaces and component implementations are defined which ensure a smooth integration of industrial components. The component implementations of compound components enable a late binding by referring to enclosed component instances only by their component interfaces. But also explicit bindings between component interfaces and component implementations can be defined. A type system for components is introduced enabling a formal definition of exchangeability and interoperability of components. Using this type system, tools are able to decide which components may be exchanged by others and which components fit together. They can also support the creation of new components from existing ones by checking a new assembly for consistency.

Keywords: Component; Component model; COTS; Hierarchical composition; Component specification; Component reengineering; Typing of components; Services; Plugs; Constraints

1. Introduction

In this paper we deal with component models which support the possibility to define components in a hierarchical manner. This means that a component may in principle be described as a set of subcomponents, their interconnections, and a set of

E-mail address: ursula.scheben@fernuni-hagen.de.

0167-6423/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.scico.2004.11.008

exported subcomponent services which can be accessed by clients of the component. Every subcomponent may itself be atomic or composed from other components. We call component models supporting such concepts *hierarchical*. Hierarchical component models reduce the complexity of building applications from components because an application can be built step by step. Starting from atomic components, new components of higher degree of granularity can be built in every step, ending up in a complete application.

Most of the current industrial component models like JavaBeans [5], Enterprise JavaBeans (EJB) [7], the Corba Component Model (CCM) [15], and also WebServices [19], only provide a flat component model. CCM provides a means to describe a set of interconnected component instances and their mapping to hosts and processes by an assembly descriptor. But this kind of description does not support the building of new components with a dedicated interface to the outer world. For EJBs it is possible to define a set of cooperating EJBs building a part of an application. This is done in a special section of the deployment descriptor. It can especially be defined which implementations to use for EJBs referred to by other EJBs of the set. But as in the case of CCM this is no means to define new components with a dedicated interface to the outer world. In contrast to the above-mentioned flat industrial component models, COM [8] offers a means to hierarchically compose components by aggregation. Aggregation means that a component may provide references to interfaces of an internal (sub)component to its clients. The clients of the aggregating "outer" component *O* do not know about the internal aggregation. They may query a reference to an interface of the aggregated component by a call to the IUnknown interface of *O* as if these interfaces were directly implemented by *O*. Unfortunately there are several drawbacks in this approach: one of them is that components must be aware whether they should be used as aggregates in the future. Possible aggregates have to provide special additional features. Components missing these features cannot be used as aggregates later on. Another drawback is that the means for hierarchical composition provided by COM are only targeted to experienced programmers. They have to use existing programming languages to compose components, which do not support component composition concepts in a first class manner.

For flat component models composition languages like, for example, the Bean Markup Language (BML) [18], CoML [2] or Beanome [6] were developed, targeted to provide a simple way to aggregate and wire together existing component instances to build new components or applications. In such languages components are treated as first class entities. Most of these languages however are targeted to a special industrial component model and are limited in the kind of exported entities. In the BML-language, for example, only methods and events can be exported, not whole interfaces. Also in most cases there exists no possibility to express whether the built components still require services of other components to fulfil their tasks. The component instances declared in a composite are already bound to a special implementation, so that it is difficult to exchange the implementation of one of its constituents later on.

In this context we introduce a hierarchical component model with the following characteristics: component instances communicate to the outer world only through their component interfaces. Component instances may be connected by corresponding entities of their component interfaces to enable an inter-component communication.