# Specifying Interaction Constraints of Software Components for Better Understandability and Interoperability

Yan Jin and Jun Han

Faculty of ICT, Swinburne University of Technology, Hawthorn, VIC 3122, Australia
{yjin, jhan}@swin.edu.au

**Abstract.** A vital issue in the correct use of commercial-off-the-shelf (COTS) components is the proper understanding of their functionality, quality attributes and ways of operation. Traditionally, COTS component vendors provide some of this information in accompanying documentation. However, the documentation is often informal and likely contains ambiguous and inconsistent statements. Even equipped with interface descriptions clearly defining the basic aspects of component use, such as operation signatures and operating platforms, this documentation does not provide a mathematically sound means for addressing the behavioural interoperability issues in component-based system design. In this paper, we propose a formal but user-friendly component specification approach which augments commercial IDLs with the capability of capturing component interoperability requirements. This approach uses unambiguous temporal operators to define sequencing and concurrency constraints between component operation invocations. Accordingly, it enables precise specifications of how a component provides its services and the correct way in which its services should be used.

## 1 Introduction

A key feature of component-based software engineering is that it allows the use of independently developed components, especially commercial-off-the-shelf (COTS) components, in constructing software systems. Underlying this independence is the common understanding of a component's capability and ways of operation between the component developer and the component user. Component interface definitions facilitate this understanding and serve as contracts between service provider components and service consumer components. A service consumer component will be able to use the services of a provider component based on its interface definition without knowing its implementation details. On the other hand, the service provider component can be implemented based on its interface definition without knowing the potential users or consumers. As such, component interface definitions play a vital role in ensuring the compatibility between the components of a composite system [7].

Commercial interface definition languages such as CORBA IDL primarily address the signature aspects of software component interfaces, *i.e.* the names, parameters and data types of the provided operations. They do not provide support for capturing the

X. Franch and D. Port (Eds.): ICCBSS 2005, LNCS 3412, pp. 54–64, 2005.

semantic or behavioural aspects of a component, including its usage, capabilities and interaction behaviour. This often poses significant problems in enforcing behavioural interoperability between components when designing component-based systems, especially COTS-based systems. That is, incorrect assumptions about the services of components often lead to incorrect usage, and therefore system failure.

Informal documentation usually accompanies the interface definition to provide supplementary information about component services and their usage. However, informal documentation is often ambiguous and sometimes contains inconsistency. This leads to difficulties in properly understanding and deploying COTS components, automating the component selection process, and developing CASE tools for automated system analysis.

To provide a sound support for component interoperability, richer and unambiguous interface descriptions are required. In addition to operation signatures, they should include service semantics, service qualities and service usage protocols [8]. In particular, the usage protocols describe the rules that govern the component interactions, including the order in which a component's operations are to be invoked so as to facilitate the proper use of the component's services. The specification of these protocols is the focus of this paper.

A body of work has been proposed to explicitly and precisely describe component interaction protocols, including [1, 2, 3, 4, 5, 8, 9, 16, 17, 18]. Most of these approaches employ formalisms with a strong mathematical flavour. This limits their use among software engineers or component developers who usually do not have the required background.

In this paper, we extend our previous work in [8, 9] and present a specification approach to component interoperability requirements. The approach builds on a formal foundation but employs a user-friendly language as the front-end. The component interaction protocol is specified in the form of constraints, using a set of intuitive temporal operators. Each constraint states a sequencing or concurrency relationship between operation invocations, representing a partial view of the protocol on the invocations. As such, this approach allows incremental specification of the interaction protocol. It also supports the run-time validation of component interactions against each individual constraint.

The remainder of this paper is organized as follows. section 2 motivates our work. section 3 presents our specification approach to component interaction. Then, section 4 discusses some relevant issues and ways to address them. This is followed by a presentation of the related work in section 5. Finally, section 6 contains the conclusions and future work.

## 2 Motivation

In this section, we present an example to highlight the need for the precise specification of component interaction protocols. In the subsequent sections, we shall use it to illustrate our approach. This example is an auctioneer component, drawn and adapted from the distributed auction system in [4]. The auctioneer communicates with a number of sellers and bidders. It is able to accept registrations from the bidders, handle auction requests