Chapter XXXI
Assurance for Temporal Compatibility Using Contracts

Omkar J. Tilak
Indiana University-Purdue University Indianapolis, USA

Rajeev R. Raje
Indiana University-Purdue University Indianapolis, USA

Andrew M. Olson
Indiana University-Purdue University Indianapolis, USA

ABSTRACT

Software realization of a large-scale Distributed Computing System (DCS) is achieved through the Component-based Software Development (CBSD) approach. A DCS consists of many autonomous components that interact with each other to coordinate each system activity. The need for such coordination, along with requirements such as heterogeneity, scalability, security, and availability, considerably increases the complexity of code in a distributed system. This chapter depicts a formal method to specify component interactions involving temporal constraints. Using the component interactions, various types of temporal interaction compatibility classes are defined. A simple case study is presented that indicates the benefits of the component interaction specifications discussed in this chapter.

INTRODUCTION

A major component of quality in software is assurance: a system’s ability to perform its job according to the specification and to handle abnormal situations (McGraw, 1999). Assurance is particularly important in the CBSD method because of the special role given by the method to reusability. It is important to obtain reusable software components whose assurance can be trusted in order to create a reliable system from such components.

Reliable software systems can be built using several methods (Musa, 1999). Static typing, for example, is a major help for catching inconsistencies before they have had time to become defects. Reusability also helps, using component libraries produced and (presumably) validated, rather than developing new software for the same purpose. But this is not enough. To be sure that DCS will perform properly, a systematic approach to specifying and implementing object-oriented software elements and their relations in a software system is needed. One such approach is known as design by contract, in Beugnard & Jezequel, et al., 1999. Under the Design by Contract theory, a software system is viewed as a set of communicating components whose interaction is based on precisely defined specifica-
Assurance for Temporal Compatibility Using Contracts

Some benefits of Design by Contract include the following:

- A better understanding of the DCS.
- A systematic approach to building defect-free DCS.
- An effective framework for debugging, testing and, more generally assuring quality.
- A method for documenting software components.

Typically, contracts for software components have been specified at the syntactical level, which provides a description of type information for a particular method of a component. In a DCS, many other aspects of a component need to be considered and hence, Kumari (2004) and Zaremski et al. (1997) have proposed a multi-level contract for components. It augments the basis syntactic level of contract with behavior, synchronization, and quality of service. Thus, the use of such a multi-level contract, during the design and development of a component, increases the assurance about a component provides a better assurance about a DCS made of such components.

The four levels of contract mentioned above are certainly important for developing a component that is to be used in a DCS; however, many DCS are time-sensitive. Hence, it is equally important to consider the timing constraints as a part of a multi-level contract for a component. Another level for component contracts called the temporal level, is proposed in Tilak (2006). In addition to providing the formal mechanism for describing the temporal level contract, Tilak (2006) also discusses the issues of compatibility and replaceability. This chapter presents the details related to the issue of compatibility and how it plays a role in providing timing-related assurance about DCS made from time-sensitive individual components.

This chapter is organized as follows. Section 2 presents a brief overview of the related work and section 3 gives the design details of the Temporal Interaction Contract specifications. Section 4 describes the algorithms to check compatibility between Temporal Interaction Contracts. Section 5 describes an example for an experimental analysis of these concepts. Section 6 provides the conclusion of this research work with the possible future enhancements.

RELATED WORK

Components with Contracts

Beugnard and Jezequel, et al. (1999) define a general model for software contracts for components. These contracts provide parameters against which the components can be verified and validated. Four classes of contracts are defined in the context of software components: syntax, behavioral, synchronization and quantitative. This classification is based on the increasingly negotiable properties of the various levels of contract. We use the idea of multi-level contracts and extend it, by creating a Temporal Interaction Contract, to describe the component interactions and provide a compatibility analysis. This research briefly mentions a technique to represent component interactions using high-level Petri nets. However, it does not deal with matching component specifications for the purpose of analyzing compatibility. Also, it does not take into consideration the time constraints associated with component interactions. The goal of the present chapter’s work is to provide operators to perform the component compatibility analysis based on temporal component interaction specifications of components.

Synchronization and Quality of Service Specification and Matching of Software Components

Kumari (2004) extends the research described in Zaremski and Wing (1997) by providing a mechanism for the formal specification and matching of a component’s properties at the synchronization and the Quality of Service levels. Two types of matches are defined: replaceability or compatibility. Replaceability is the ability of two components to replace each other within a system and compatibility is the ability of two components to interoperate, communicate, and cooperate with each other when brought together to form a system. Although these two efforts provide matching techniques for syntax, semantics, synchronization, and QoS level in the component contract, they do not take into consideration the timing constraints associated with the component interaction. The chapter extends the concepts of specification matching, described in Kumari (2004) and Zaremski and Wing (1997), by providing compatibility criteria in the context of temporal interaction constraints.