Science of Computer Programming 115–116 (2016) 225–246

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

A tag contract framework for modeling heterogeneous systems

Thi Thieu Hoa Le [a,*], Roberto Passerone [a], Uli Fahrenberg [b], Axel Legay [b]

[a] DISI, University of Trento, Italy
[b] INRIA/IRISA, Rennes, France

ARTICLE INFO

Article history:
Received 1 April 2014
Received in revised form 8 June 2015
Accepted 10 June 2015
Available online 15 July 2015

Keywords:
Contract
Heterogeneity
Tag
Synthesis
Component model

ABSTRACT

Development of distributed systems can be supported effectively by a contract-based methodology as contracts can ensure interoperability of components and adherence to specifications. Such development can become very complex since distributed systems can consist of components which are heterogeneous in terms of computational and interactive model. Several frameworks, both operational and denotational, have been proposed to handle heterogeneity using a variety of approaches. However, the application of those frameworks to contract-based design has not yet been investigated. In this work, we adopt the operational mechanism of tag machines to represent heterogeneous systems and construct a full contract model. We introduce heterogeneous composition, refinement, dominance and compatibility between contracts, altogether enabling a formalized and rigorous design process for heterogeneous systems. Besides, we also develop a method to synthesize or refine the component models so that their composition satisfies a given contract.

1. Introduction

A distributed system often consists of a collection of components which are developed and executed across different computing nodes in a distributed manner. Development of such systems can be supported effectively by design methodologies such as those based on *contracts* [1]. A contract-based methodology rests on a model of the system that allows the designer to specify constraints on the environment of each component, called the component *assumptions*, as well as what can be (or must be) guaranteed given the satisfaction of such constraints, called the component *guarantees*. This explicit separation between assumptions and guarantees is the key in supporting the development of distributed systems.

On top of this, when components are distributed across different companies in the supply chain, and they are part of a complex system, they are also likely to be developed using different models and interaction mechanisms, thereby making the model of the whole distributed system *heterogeneous*. Heterogeneity may imply different models of time, from simple precedence relations to discrete and continuous real time, as well as different semantics of synchronization, such as synchronous and asynchronous, and rendez-vous vs. broadcast [2]. This aspect aggravates the integration activities, given that it is difficult to define a complete virtual system that can be executed and analyzed prior to the physical implementation. To deal with heterogeneity, several modeling frameworks have been proposed oriented towards the representation and simulation of heterogeneous systems, such as the Ptolemy framework [3], or towards the unification of their interaction paradigms,

* Corresponding author.
E-mail address: lethithieuhoa@gmail.com (T.T.H. Le).

http://dx.doi.org/10.1016/j.scico.2015.06.004
0167-6423/

such as those based on *tagged events* [4]. The latter can capture different notions of time, e.g., physical time, logical time, and relate them by mapping tagged events over a common tag structure [5].

Due to the significant inherent complexity of heterogeneity, there have been only very few attempts at addressing heterogeneity in the context of contract-based models. For instance, the HRC model from the SPEEDS project[1] was designed to deal with different viewpoints (functional, time, safety, etc.) of a single component [6,7]. However, the notion of heterogeneity in general is much broader than that between multiple viewpoints, and must take into account diverse interaction paradigms. In our previous work, we have laid the foundation for a modeling methodology which is contract-based and heterogeneous [8]. Our methodology enables different distributed components to be specified as different contracts and the whole system model to be built by composing its component contract models. In addition, the underlying modeling mechanism of *heterogeneous tag machines* [9], which supports our methodology, allows the component models to be specified in different timed models and interaction styles. In this paper, we further develop our framework to cover also aspects related to design and development, and explore the issue of how to correctly decompose the specification hierarchically into a refined implementation.

Typically, development of distributed systems can be done in a bottom-up manner, i.e., by *composing* simpler predefined components together. Alternatively, they can be developed in a top-down fashion, by successively *decomposing* or *refining* the system into simpler components which are then independently developed and implemented. This approach is well supported by a contract-based methodology: the decomposition is correct as long as the composition of the contracts associated with the sub-components satisfies or refines the contract of the intended system. When this condition is *not* satisfied, i.e., the composition of the contracts of the sub-components does not refine the system contract, designers must adjust the component contract specifications until the inconsistency with the system model is cleared. In this work, we discuss which steps must be taken to check consistency of a decomposition with the specification, and to adjust the component specification when this is not satisfied.

More formally, we deal with the problem of checking if a contract $\mathcal{C}$ is correctly decomposed (or refined) into a pair $\{\mathcal{C}_1, \mathcal{C}_2\}$ of heterogeneous contracts. When this is not the case, we discuss methods to update the contract set in order to make the composition refine $\mathcal{C}$ if necessary. To this end, we study *decomposing conditions* under which the contract decomposition can be verified, and thereby propose a *synthesis strategy* for fixing wrong decompositions. In particular, this paper extends our previous work with a synthesis capability for the tag contract framework to provide for extra flexibility in component design and reusing.

The rest of the paper is organized as follows. We first review in Section 2 work on contract and synthesis which is related to our approach. In Section 3, we recall notions of tags and heterogeneous tag machines (TMs) and their composition. In Section 4, we present our contract framework for modeling heterogeneous systems with a full set of operations and relations such as contract satisfaction, contract refinement, contract dominance and contract compatibility. We also recall that a shorter version of this section without formal proofs and the ability of checking contract compatibility appeared at the FOCLASA workshop [8]. Based on the decomposition into a pair of *two* contracts provided in Section 4.4, we further extend our framework to be capable of synthesizing the component contracts and propose decomposing conditions for a pair of contracts. In particular, we suggest a strategy for synthesizing those contracts in order to make their composition satisfy a predefined contract in Section 5. Finally we conclude in Section 6.

2. Related work

Contracts were first introduced in Meyer's design-by-contract method [1], based on ideas by Dijkstra [10], Lamport [11], and others, where systems are viewed as abstract boxes achieving their common goal by verifying specified contracts. Such technique guarantees that methods of a class provide some post-conditions at their termination, as long as the pre-conditions under which they operate are satisfied. De Alfaro and Henzinger subsequently introduced interface automata [12] for documenting components thereby enabling them to be reused effectively. This formalism establishes a more general notion of contract, where pre-conditions and post-conditions, which originally appeared in the form of predicates, are generalized to behavioral interfaces. The central issues when introducing the formalism of interface automata are compatibility, composition and refinement. Separating assumptions from guarantees, which was somewhat implicit in interface automata, has then been made explicit in the contract framework of the SPEEDS HRC model [6,13]. A separation between specifying assumptions on expected behaviors and guarantees to achieve them at run time has recently been applied to the handling of synchronization requirements to improve the component-based development of real-time high-integrity systems [14].

The relationship between specifications of component behaviors and contracts is further studied by Bauer et al. [15] where a contract framework can be built on top of any *specification theory* equipped with a composition operator and a refinement relation which satisfy certain properties. Trace-based [6] and modal contract [16] theories are also demonstrated to be instances of such a framework. This formalization enables verifying if a contract can be decomposed into two other contracts by checking if that contract can *dominate* the others. In this work, we take advantage of this formalization and dominating notion and adapt them to construct our tag contract framework and its synthesis functionality. In particular, we extend these notions to a heterogeneous context.

[1] www.speeds.eu.com.