Science of Computer Programming 97 (2015) 333–348

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Contracts-refinement proof system for component-based embedded systems

Alessandro Cimatti *, Stefano Tonetta *

Fondazione Bruno Kessler, Trento, Italy

ARTICLE INFO

Article history:
Received 13 April 2013
Received in revised form 8 February 2014
Accepted 16 April 2014
Available online 7 July 2014

Keywords:
Contract-based design
Temporal logics
Embedded systems
OCRA

ABSTRACT

Contract-based design is an emerging paradigm for the design of complex systems, where each component is associated with a contract, i.e., a clear description of the expected interaction of the component with its environment. Contracts specify the expected behavior of a component by defining the *assumptions* that must be satisfied by the environment and the *guarantees* satisfied by the component in response. The ultimate goal of contract-based design is to allow for compositional reasoning, stepwise refinement, and a principled reuse of components that are already pre-designed, or designed independently.

In this paper, we present fully formal contract framework based on temporal logic (a preliminary version of this framework has been presented in [1]). The synchronous or asynchronous decomposition of a component into subcomponents is complemented with the corresponding refinement of its contracts. The framework exploits such decomposition to automatically generate a set of proof obligations. Once verified, the conditions allow concluding the correctness of the architecture. This means that the components ensure the guarantee of the system and the system ensures the assumptions of the components. The framework can be instantiated with different temporal logics. The proof system reduces the correctness of contracts refinement to entailment of temporal logic formulas. The tool support relies on an expressive property specification language, conceived for the formalization of embedded system requirements, and on a verification engine based on automated SMT techniques.

1. Introduction

Embedded systems are continuously growing in complexity, and carry out critical functions. This calls for effective and rigorous methods to find defects early in the development process, to guarantee safety and correctness, and to reduce the costs of certification.

Component-based design is a very promising paradigm, amenable to compositional reasoning and to reuse of components. In order to tame the complexity of embedded systems, the component implementations can be abstracted with properties that specify the behavioral aspects that are relevant for the system-level properties. In this settings, the cost of formal methods is alleviated by compositional verification and reuse of proof.

Contract-based design, first conceived for software specification [2] and now applied also to embedded systems [3–8,1,9,10], structures the component properties into contracts. A contract specifies the properties assumed to be satisfied by the component environment (assumptions), and the properties guaranteed by the component in response (guarantees).

* Corresponding authors.

http://dx.doi.org/10.1016/j.scico.2014.06.011
0167-6423/

Contract-based design comes with multiple advantages: it supports stepwise refinement, compositional reasoning, and a principled reuse of pre-designed or independently designed components. This approach is adopted in several recent projects for embedded systems, such as SafeCer (http://www.safecer.eu), which exploits contracts to enable a compositional certification and reuse of pre-qualified components.

In this paper, we give full account of a contract framework where contracts are tightly integrated within an architectural decomposition of the system, and are specified with temporal logics. The framework provides a formal notion of correctness of the contract refinement, which is reduced to checking the validity of set of (necessary and sufficient) proof obligations. The generated proof obligations are formulae in the same temporal logic used to express the contracts, and can be decided by means of suitable decision procedures.

The approach, which extends and completes the framework presented preliminarily in [1], encompasses both synchronous and asynchronous composition of components, and is based on a generic notion of traces, so that it can be instantiated with different temporal logics.

This framework has some distinguishing features. First, the refinement checks are tightly integrated with the architecture decomposition flow of safety-critical applications. This includes taking into account the connections of components that play a fundamental role in the contract refinement. This allows for an early validation of the choices underlying the architectural decomposition and requirements delegation. Second, our approach supports the automated production of refinement checks. Finally, compared to other approaches (that are typically either theoretic or limited to simple specification patterns), our specialization supports more expressive and general properties.

The approach has been implemented in the tool OCRA [11], which was used to analyze systems from various application domains. The tool instantiates the framework by allowing the specification with two (of many possible) logics: Linear Temporal Logic (LTL) [12], which models discrete traces, and HRELTL, a variant of LTL defined in [13], which models hybrid traces (combining discrete and continuous transitions). In the case study proposed in [7], used as a running example in this paper, OCRA was able to pinpoint some inaccuracies in the original formulation.

This paper is structured as follows. In Section 2, we discuss relevant related work. In Section 3, we present a motivating case study and the envisaged contract-based design that requires the methods proposed in the paper. In Section 4, we present our contract-based framework in terms of sets of generic traces. In Section 5, we define the proof obligations proving that they characterize the correctness of contract refinement. In Section 6, we instantiate the trace-based framework described in the previous section using temporal logics and we describe the support in terms of tool and concrete language to the framework. In Section 7, we evaluate the approach on the case study. In Section 8, we draw some conclusions and outline directions for future work.

2. Related work

Contracts have been first defined in the context of object-oriented programming by Meyer [2]. For software programs, assumptions and guarantees are represented respectively by preconditions and postconditions of functions: preconditions define the assumptions that the function caller must satisfy at the entry point of the function; postconditions define the guarantee that the function provider must satisfy at the exit point. Other used assertions are class- and loop-invariants. In concurrent programs, the interaction between service clients and providers is more complex, and require the use of assertions over execution traces.

As in other works, such as [7,14,9], we separate the architectural design (where the primitive components are specified as black boxes) from the behavioral models of the component (that may be specified in different languages). In these works, however, contracts distinguish between assumption and guarantee only for enabling assume-guarantee reasoning. Thus the semantics of a contract is simply the implication "if the assumption holds, also the guarantee holds". In our work, instead, following the seminal work of Meyer [2] and the recent applications to embedded systems, contracts represent two distinct properties, one for the environment of the component and one for the component itself. If the environment does not satisfy the assumption, then the architecture is not correct.

This paper builds on the recent works presented in [8], from which we inherit some formal notions for contract-based reasoning. While [8] describes when and how contracts can be *composed* (in a bottom-up design process), we focus on the verification conditions necessary to prove that a contract is correctly refined by a specific *decomposition* (in a top-down design process). Moreover, we detail how the architectural connections play a fundamental role in the composition of components and therefore also in the contract refinement. Based on this insight, we provide new theoretical results tailored to the verification of contracts decomposition.

The idea of contract decomposition and refinement is also provided in [7]. Our approach has two important differences. First, [7] adopts a semantics of a contract in terms of implementations, without reference to the notion of environment, and the refinement is defined as trace-set inclusion. This approach is therefore missing the notion of assumption as constraint for the environment and allows to refine a contract by strengthening the assumptions. Another important difference with respect to [7] is that it uses contract patterns converted into automata, while our approach is based on temporal logics.

As in [3], we use a trace-based semantics for contracts. However, we use a concrete property specification language to express the assumption and guarantees of the contracts. Also, differently from [3], we do not assume that contracts are in normal form, but we reduce to normal form just for the refinement checks. This may become important when the negation of assertion is not possible (as for timed and hybrid automata) or when performing syntactic checks of re-