Composing Specifications for Coordination

Carlo Montangero\textsuperscript{1} and Laura Semini\textsuperscript{2}

\textsuperscript{1} Dipartimento di Informatica, Università di Pisa, monta@di.unipi.it
\textsuperscript{2} Dip. di Sistemi e Informatica, Università di Firenze, semini@dsi.unifi.it

Abstract. We introduce Oikos\textsuperscript{adtl}, a specification language for distributed systems based on asynchronous communication via remote writings. The language is designed to support the composition of specifications. It allows expressing the global properties of a system in terms of the local properties of the components and of coordination templates. Oikos\textsuperscript{adtl} is based on an asynchronous, distributed, temporal logic, which extends Unity to deal with components and events. We present the specification language and its semantics, introduce a number of compositionality theorems, and discuss some coordination templates. A fragment of a standard case study is used to validate pragmatically the approach, with respect to expressiveness and workability.

1 Introduction

The design of quality software for distributed systems is becoming more and more critical, due to the current impact of software on every technical accomplishment, and the fact that networks pervade any current application. The problem has two facets: the complexity of the systems under development, and the need of continuous update to keep the pace with moving requirements. To avoid old errors and face complexity, we need a design process that helps in getting the right system architecture right from the outset, rather than rely only on afterwards checks. Besides, to cope with change better, the process should naturally support the recording of the complete justification of the development, for future reference.

A design process based on formal refinements, centered on the system architecture, when applied in the early phases of development, would mitigate the problems related to both complexity and change. So, a long term goal of ours is to extend the refinement calculi, traditionally applied to programming in the small \cite{2}, to programming in the large. The approach can be effective only if it comes with simple, workable concepts and notations. A key strategy to cope with complexity, also in design, is “divide and conquer”: express the global properties of a system in terms of the properties that are local to its components and of coordination templates that make it easier for the designer to compose specifications. The other way round, once valuable composition templates are available, they can be exploited to devise refinement templates, to be used top-down in the design.
The work reported here is a first step towards a distributed refinement calculus that integrates in a natural way local refinements (i.e. within a single component) and coordination templates. Our previous work [21] can readily be adapted to work locally. The compositionality results of this paper represent the main ingredient to develop useful coordination, and hence refinement, templates.

To exploit our past experience, we circumscribe our field of application and consider distributed systems that consist of loosely coupled asynchronous long running services, like work-flow systems. These systems are conveniently modeled as multiple data-spaces interacting via remote writings. The multiple data-spaces paradigm is widely regarded as interesting for coordination applications [4, 8, 10]. Brogi and Jacquet [5] argue convincingly for the relevance of asynchronous communications. Still, there is a lack of satisfactory design and analysis techniques for these systems.

We introduce Oikos\textsubscript{adtl}, a specification language for distributed systems based on asynchronous communication via remote writings. The language is designed to support the composition of specifications. It allows expressing the global properties of a system in terms of local properties and of coordination templates. The former are properties exposed by a single component, the latter describe the approach taken to control the interactions of the components. Oikos\textsubscript{adtl} is based on an asynchronous, distributed, temporal logic, which extends Unity [6] to deal with components and events.

The structure of the paper is the following. Section 2 presents the specification language and its semantics. Section 3 introduces a number of composition theorems, and discusses some coordination templates. A fragment of a standard case study from the COORDINA working group is used in Section 4 to show the expressiveness of the language and to validate the selected theorems and templates, pragmatically. Before some concluding remarks, Section 5 discusses related work.

2 Oikos\textsubscript{adtl}

The extensions to Unity that allows Oikos\textsubscript{adtl} to deal with events, are taken from Oikos\textsubscript{tl} [21]. This paper works out the extension to distributed components first proposed in [20, 17].

A component has a name, and it is specified by liveness (something good will happen) and safety (nothing bad will happen) properties of the evolution of its state. An event occurs when a given condition is established in a component. Operators causes\textsubscript{c}, causes, needs, because\textsubscript{c}, and because permit to deal with events. The first two express liveness conditions, the latter express safety conditions. Suffix\textsubscript{c} stands for closely, causes relates an event and a condition, and specifies that the occurrence of the event is sufficient to arrive in a state in which the condition holds; causes\textsubscript{c} requires also the condition to occur in the state in which the event occurred, or in the next one. Needs requires a condition to hold when an event occurs; because also relates an event and a condition, and specifies that the condition is necessary for the event to occur; the condi-