A Calculus for Component Evolvability (Extended Abstract) ⋆

Mario Bravetti[1], Cinzia Di Giusto[2], Jorge A. Pérez[3], and Gianluigi Zavattaro[1]

[1] Laboratory FOCUS (Università di Bologna / INRIA)
[2] INRIA Rhône-Alpes, Grenoble, France
[3] CITI - FCT New University of Lisbon, Portugal

Abstract. We describe ongoing work on a formal framework for reasoning about *dynamically evolvable* aggregations of components. The framework relies on an extension of CCS with primitives describing components and evolvability capabilities. We propose and investigate two correctness properties: *bounded* and *eventual adaptation*. While bounded adaptation ensures that at most k errors will arise in future states—including those reachable as a result of dynamic reconfigurations—, eventual adaptation ensures that the system will eventually reach a state from which no other error will arise (i.e., only finitely many errors can occur). We study the decidability of these two adaptation properties in six different variants of the calculus, which represent different evolvability patterns.

Summary. The deployment of applications by the *aggregation* of elementary blocks (modules, components, web services, ...) is a long-standing principle in software engineering. This work addresses the *correctness* of aggregations of *components* which are subject to *evolvability* and *adaptation* concerns. We use the term "component" in a broad sense, as it refers to elementary blocks such as web services in cloud computing scenarios, but also to analogous concepts in related settings, such as services in service-oriented computing or long-running processes in workflow management. Here we introduce a process calculus with components and primitives for evolvability, and use it to study the decidability of verification problems associated to the correctness of aggregations of components.

Approach. We introduce a variant of Milner's CCS [3] without restriction and relabeling, and extended with a primitive notion of component. In this calculus, called *Evolvable CCS* ($\mathcal{E}$ in the sequel), $a[P]$ denotes the component called (or located at) a with state given by process P. Name a acts as a *transparent locality*: P can evolve on its own as well as interact freely with its surrounding environment. Components can be reconfigured by means of interactions with *update actions*. An update action $\widetilde{a}\{U\}$ reconfigures a component located at a with the behavior defined by U, which can be thought of as a *context* with zero or more holes denoted by $\bullet$. More precisely, the reconfiguration or *evolution* of a is realized by the interaction of component $a[P]$ with the update action $\widetilde{a}\{U\}$, which leads to process $U\{^{P}/_{\bullet}\}$, i.e., the process obtained by replacing every hole in U by P.

⋆ Supported by the EU integrated project HATS, the Fondation de Coopération Scientifique Digiteo Triangle de la Physique, and FCT / MCTES (CMU-PT/NGN44-2009-12) - INTERFACES

We propose two correctness properties for $\mathcal{E}$ processes. The first one, *k-bounded adaptation* (abbreviated $\mathcal{BA}$) ensures that, given a finite k, at most k errors can arise during the system evolution. The second property, termed *eventual adaptation* (abbreviated $\mathcal{EA}$), is similar but weaker: it ensures that the system will eventually reach a state from which no other error will arise (that is, only finitely many errors can occur).

The $\mathcal{E}$ calculus provides a generic framework for reasoning about component aggregations and their correctness. We investigate the decidability of $\mathcal{BA}$ and $\mathcal{EA}$ in several variants of the calculus, which are obtained via two orthogonal characterizations. The first one is *structural*, and distinguishes between *static* and *dynamic* topologies of aggregations. In a static topology, the number of components does not vary along the evolution of the system: components cannot be destroyed nor new components can appear. In contrast, in the more general dynamic topology this restriction is not considered. The second characterization is *behavioral*, and concerns *update patterns*—the context U in an update action $\widetilde{a}\{U\}$. As hinted at above, update patterns determine how a component will evolve after an update action. In order to account for different evolvability patterns, we consider three kinds of update patterns, which determine three families of $\mathcal{E}$ calculi—denoted $\mathcal{E}^1$, $\mathcal{E}^2$, and $\mathcal{E}^3$, respectively. The first update pattern admits all kinds of contexts, and so it represents the most expressive form of update. In particular, holes $\bullet$ can appear behind prefixes. The second update pattern forbids such guarded holes in contexts. In the third update pattern we additionally require contexts to have at least one hole, thus preserving the existing behavior (and possibly adding new behaviors): this is the most restrictive form of update. In our view, the variants of $\mathcal{E}$ derived from these two characterizations capture a fairly ample spectrum of scenarios that arise in the joint analysis of correctness and adaptation concerns in component aggregations.

Results. We have obtained the following (un)decidability results for the variants of $\mathcal{E}$:

	Dynamic Topology	Static Topology
$\mathcal{E}^1$	$\mathcal{BA}$ undec / $\mathcal{EA}$ undec	$\mathcal{BA}$ undec / $\mathcal{EA}$ undec
$\mathcal{E}^2$	$\mathcal{BA}$ dec / $\mathcal{EA}$ undec	$\mathcal{BA}$ dec / $\mathcal{EA}$ undec
$\mathcal{E}^3$	$\mathcal{BA}$ dec / $\mathcal{EA}$ undec	$\mathcal{BA}$ dec / $\mathcal{EA}$ dec

The decidability of eventual adaptation is proved by resorting to Petri nets, while for bounded adaptation we consider the theory of *well-structured transition systems* [2,1] and its associated results. The undecidability results are obtained via encodings of Minsky machines [4], a well-known Turing complete model.

References

1. P. A. Abdulla, K. Cerans, B. Jonsson, and Y.-K. Tsay. Algorithmic analysis of programs with well quasi-ordered domains. *Inf. Comput.*, 160(1-2):109–127, 2000.
2. A. Finkel and P. Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256(1-2):63–92, 2001.
3. R. Milner. *Comunication and Concurrency*. Prentice Hall, 1989.
4. M. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, 1967.