Towards PCC for Concurrent and Distributed Systems (Work in Progress)

Anders Starcke Henriksen Andrzej Filinski
Department of Computer Science
University of Copenhagen, Denmark
{starcke, andrzej}@diku.dk

Abstract

We outline some conceptual challenges in extending the PCC paradigm to a concurrent and distributed setting, and sketch a generalized notion of module correctness based on viewing communication contracts as economic games. The model supports compositional reasoning about modular systems and is meant to apply not only to certification of executable code, but also of organizational workflows.

1 Introduction

The notion of proof-carrying code is relatively well understood for sequential programs, though substantial technical challenges evidently remain in constructing practically useful PCC systems for realistic application domains. It would clearly be desirable to extend these techniques and results to also allow certification of components of concurrent and distributed systems, but here it is not so clear what a formal certification should even mean in principle, let alone how to realize it in practice.

The immediate difference from a sequential setting is that components are typically running on separate systems, communicating only by exchanging messages. Thus, instead of Hoare-style pre- and post-conditions for a piece of code, we must in general specify its behavior by *communication contracts*: descriptions of which messages may or must be sent between components, depending on the entire previous communication history between them (or some more compact representation thereof). While this change obviously imposes some additional technical difficulties, it does not yet significantly strain the basic notion of certified correctness.

Rather, we consider the main problem with extending PCC to a truly distributed (both physically and administratively) setting to be that even certified components or modules may ultimately depend on external services, whose internal structure will never be available for inspection or certification. Hence, correctness guarantees will in general not be absolute, but only contingent upon correct behavior of some uncertifiable components. But unlike the traditional setting, the client of the certified module will in general not be in a position to assume responsibility for correctness of all submodules used by that module, because those may be provided by third parties, selected by the module implementor, and in principle unknown to the original client.

To reason about such systems in a compositional way, we propose a refined, quantitative model of contract conformance, in which component implementors are rewarded by each other for behavior in accordance with the agreed-upon contract, and penalized for deviations. A properly implemented (i.e., "correct") component is then one that ensures that any fines it may incur for incorrect behavior will be at least matched by the fines it can collect from any faulty subcomponents it depended on. In particular, such a component will never implement a high-assurance service (i.e., one with a high penalty for failure) by relying on a low-assurance one.

Incidentally, such a view of correctness may be relevant even for non-distributed systems, in that a monetary representation of safety or security warranties is inherently more robust in the real world than the mathematically ideal notion of absolute correctness that pure PCC in principle promises. In particular, this “putting your money where your mouth is” notion of contract conformance allows seamless integration of modules that have been machine-verified, but are not equipped with independently checkable certificates.

2 A game-theoretic model

To concretize this notion of correctness, we have developed an abstract, game-theoretic model of specification conformance for communicating modules. Here, communication actions are seen as moves, contracts are interpreted as game rules (including any applicable rewards or fines associated with particular moves), and implementations, or processes, correspond to strategies [1]. In the model, we consider both games and strategies at a very abstract level, namely as infinite-state automata; in practical realizations, both contracts and processes would be expressed in suitable languages with more internal structure, to help guide reasoning about conformance.

In general, a module will play simultaneous games with multiple peers, formalized as a *contract portfolio*; a certificate represents a proof that the overall strategy is fiscally sound, i.e., that the cumulative payoff from all the module’s games remains non-negative at all times. In particular, a correct module will never be the *first* to break a communication contract, but it may be forced to do so by the previous failure of another module. Semantic correctness, i.e., that a process p satisfies communication contracts $c_1,...,c_n$ is captured by a coinductively defined relation $\models p : c_1,\dots,c_n$.

Moreover, the model allows compositional reasoning about correctness, in the sense that two modules may cooperate (by establishing an internal communication contract between them) to implement an external specification. For a contract c, let $\overline{c}$ represent the same contract, but with the roles of the two players interchanged. Then if $\models p_1 : c_1,\dots,c_n,c'_1,\dots,c'_{m_1}$ and $\models p_2 : \overline{c_1},\dots,\overline{c_n},c''_1,\dots,c''_{m_2}$, where only the internal contracts $c_1,...,c_n$ mention any internal communication links between processes p_1 and p_2, the concurrent composition $p_1 \| p_2$ (straightforwardly definable) will satisfy $\models p_1 \| p_2 : c'_1,\dots,c'_{m_1},c''_1,\dots,c''_{m_2}$. This can be seen as a generalization of the sequential composition rule of Hoare logic, where commands c_1 and c_2 satisfying $\{A\}c_1\{C\}$ and $\{C\}c_2\{B\}$ can be composed into $c_1;c_2$ satisfying $\{A\}c_1;c_2\{B\}$, with no mention of the intermediate assertion C.

3 Towards certification

At this stage, we have only started looking at how to formally represent proofs that a module implementation is formally correct with respect to its contract portfolio, i.e., a sound proof system $\vdash p : c_1,\dots,c_n$ for semantic correctness. However, once the novel notion of correctness is taken into account, the property we are certifying is ultimately still a traditional safety property; in particular, any failures to satisfy a contract portfolio will be manifestly observable in finite time. In other words, we do not consider unquantified liveness properties such as fairness, or even absence of deadlock; rather, it is only the failure to send a required message by a specific deadline that constitutes an actionable breach of contract. Thus, in principle, standard code-certification techniques should apply without requiring major modifications.

It may be worth considering the significance of code certification in a software-as-service model in the first place: if warranties are ultimately performance-based rather than absolute, do we really need certificates at all? We do, of course, in essentially the same instances as for sequential settings, namely for code not executed by its original creator. That is, in some situation a client (interpreted broadly) wants to not only engage in a game with a contractor, but to actually purchase (or otherwise obtain) that contractor’s full strategy, meaning the actual executable code, together the the relevant service agreements with any subcontractors, who now become responsible to the original client directly. In this case, the client will generally want an absolute correctness proof for the purchased strategy, which can be integrated modularly with that of the client's other games.

4 Context and perspectives

This work was developed in the context of the project *TrustCare: Trustworthy Pervasive Healthcare Services* (`www.trustcare.eu`). Part of the broader goals of the correctness model was that the notion of processes and contracts should not be limited to executable computer code and communication, but be able to serve as a general model of interacting actors (both human and organizational), with internal *workflow* procedures for achieving specific objectives, subject to external constraints. The prototypical scenario is the organization of tasks in a hospital, with the responsible actors representing patients and staff members (doctors, nurses, orderlies, receptionists, pharmacists, etc.), interacting not only by exchange of information, but also by physical actions, such as medical tests and procedures.

In general, the interdependencies between these actions may be quite complex: individual patients may suffer from multiple conditions, drugs and treatments may interact in complex ways, test results may be time sensitive, etc. Thus, developing and certifying a set of workflows and best-practice guidelines for individual staff members, to ensure that all patients are treated safely and efficiently, is a non-trivial task, well suited for (at least partial) automation. The quantitative model also allows the representation of relative importance of potentially conflicting constraints; for instance, (combinations of) actions that would significantly endanger a patient's life would be assigned higher negative payoffs than those which are merely wasteful of resources; and best-practice workflows will ensure that robust checks and balances are in place that help avoid potentially dangerous outcomes resulting from isolated minor errors by individual actors.

One might expect that, as patient records become increasingly electronic, formal certification of large-scale institutional workflows would become a licensing requirement for health care providers, in line with basic hygiene and staff training requirements. However, at the present time, we are targeting the model primarily towards certification of traditional code.

Acknowledgment The authors gratefully acknowledge the substantial contributions of Tom Hvitved of the *3gERP (Third-generation Enterprise Resource Planning)* project to the development of the contract model.

References

[1] Anders Starcke Henriksen, Tom Hvitved, and Andrzej Filinski. A game-theoretic model for distributed programming by contract. In *Workshop on Games, Business Processes, and Models of Interaction*, Lübeck, September 2009. Gesellschaft für Informatik. To appear.