Local Reasoning about Reconfigurable Component-based Systems

Radu IOSIF (VERIMAG, Grenoble)
mailto:Radu.Iosif@univ-grenoble-alpes.fr

1 Background

Nowadays computer systems are highly complex. Most of them are intrinsically distributed and dynamically reconfigurable (Big Data, Internet of Things, Cloud Computing, Smart Cities), concurrent at both the physical level of CPU cores and the logical level of threads, engaging in complex interactions not only between their own hardware and software parts, but also with their networking environment. Moreover, we are expecting an increase of several orders of magnitude in the size and complexity of such interdependent systems, on which most aspects of today's life depend. This raises important technical challenges, not entirely understood.

Designing and understanding such complex systems is only possible due to their *modularity*: a system is hierarchically organized as an architecture of components, whose internal details are encapsulated within simple well-defined interfaces. For instance, a program is datatype modular if its memory accesses are calls to containers with algebraic data type interfaces (such as, e.g., stacks, queues, maps), whereas a concurrent system is thread modular if it consists of replicas of few behavioral patterns, interacting via well-defined composition operators. The modularity of a component-based computer system is instrumental in performing *updates* (replacing one or more components with newer versions having the same interfaces) and *reconfigurations* (changing the coordinating architecture by adding new components, removing obsolete versions or even changing the topology of interactions, e.g. from a token ring to a star).

Because such complex systems control many aspects of human life (e.g. airline traffic, power grids, banking and social networks), it is important to ensure their correctness à priori, by *design*, and also à posteriori, by *verification*. For instance, a denial-of-service type of error caused by multiple computers trying to connect to the same service at once resulted in a massive power blackout on Northeastern United States on August 14, 2003. Unfortunately, there are currently no modeling or verification methodologies that can guarantee an acceptable level of confidence or ensure correctness after an update has been done. For instance, widely used Architecture Description Languages (described by the ISO/IEC/IEEE 42010 standard, see [3] for a survey) often have informal (English written) rules to describe coordination of processes according to their rôle, with virtually no associated logical semantics or formal verification methodology.

The components of a modular system can be viewed, at a higher level of abstraction, as *resources*, that can be either static or dynamic (a piece of data or a process), physical or logical (CPUs or threads), having simple atomic (a memory cell) or more elaborated structure (such as a linked list or a pipeline). The main operation on a resource domain is *composition*, understood as the product of a partial monoid; composition is usually associative, has neutral element and it is very often commutative as well. However, because some resources are incompatible for composition, the operation is only partially defined. Efficient reasoning about resources in a computer system relies on two interdependent aspects:

(i) *locality*, which is the ability to describe the effect of an update only from the parts of the system's configuration that are involved, while ignoring the ones unchanged, and

(ii) *compositionality*, which is the ability to join the results of local analyses into a global condition capturing the correctness requirement for the entire system.

2 Distributed Systems with Reconfigurable Architectures

Building on top of the Behavior-Interaction-Priorities (BIP) framework [1], the Dynamic Reconfigurable BIP (DR-BIP) [4] model considers that the degree of dynamism of a system can be understood as the interplay of the dynamic changes in three independent aspects:

1. The ability of describing coordination of parametric systems, i.e. systems in which a finite but unbounded number (unknown à priori) of replicated components interact, for instance $m \geq 1$ producers and $n \geq 1$ consumers.

2. Adding and deleting components and managing their interactions, depending on dynamically changing conditions. For instance, in a reconfigurable ring of $n \geq 2$ components, one needs to remove a component after a failure and add the component after recovery. However, even changing the number of components may cause concurrency errors. For instance in some particular architectures, the absence of deadlocks may depend on whether the number of active components is even or odd.
3. The most challenging aspect is allowing services to seamlessly roam and continue their activities on any available device or computer, also known as "fluid architectures" or "fluid software" [5]. Supporting migration of components requires a disciplined management of dynamically changing coordination rules. Thus, self-organizing systems use different *motifs* to adapt their behavior to meet a global property.

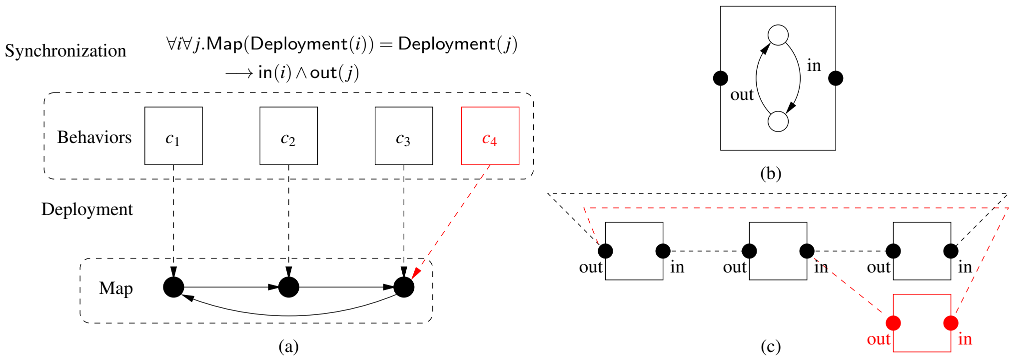

Figure 1: Architectural Motif in DR-BIP

In the DR-BIP [4] framework, a *motif* (Fig. 1a) is an elementary unit used to describe dynamic architectures, that encapsulates:

- the behavior of a set of components, that are replicas of a small set of *component types*, usually represented as finite-state machines (Fig. 1b),
- the interaction rules defining the coordinating architecture (Fig. 1c shows the architecture induced by the interaction rule from Fig. 1a)
- the reconfiguration rules dictating the allowed modifications to the configuration of a motif, i.e. creation, deletion and migration of components.

A motif has two layers: the set of *behaviors* (active components) and the *map* (a graph modeling the physical network) on which these components are deployed. A component is an instance of a finite-state machine component type, whose actions are triggered by interaction ports (e.g. *in* and *out* in Fig. 1b). The *deployment* is a partial function that assigns components with nodes in the map. The synchronization rules define the interactions between the ports (e.g. the synchronization rule in Fig. 1a states that the *out* port of each component interacts with the *in* port of the component adjacent to it in the map). For instance, because the map in Fig. 1a is a ring and there is exactly one component deployed on each node of the map, the synchronization rule describes a proper token-ring architecture (Fig. 1c). If a new component (Fig. 1a in red) is added to the set of behaviors, the synchronization rule breaks the token-ring structure (Fig. 1c in red).

A notion of composable partial architectures and a resource logic for this domain has been recently introduced in [2] by several members of Verimag. This work is a starting point for the development of an effective method for the specification and verification of dynamically reconfigurable component-based systems.

3 Challenges and Goals

The main goal of this project is the development of a logic-based modeling and verification methodology for systems consisting of *active resources* (e.g. processes, threads), as opposed to the more traditional view of resources being just passively shared among actors (e.g. chunks of memory). In this new setting, resources may dynamically engage in communication (interactions), migrate between the physical nodes of the network, take ownership and/or have knowledge of other resources, etc. This requires defining

1. new semantic domains equipped with composition operators that capture the above mentioned aspects of dynamically reconfigurable modular systems, whose definition is usually more complex than disjoint union (aggregation) of resources, and

2. new resource logics that allow to reason about the dynamic updates of the system in a compositional way; since we consider very large scale systems (possibly spread over the internet) the only reasonable approach to the design and verification is considering isolated modules independently of the environment in which they evolve.

During this internship you are expected to actively contribute to the development of the theory and tool support of the methodology, engage in research and participate to scientific seminars.

4 How to Apply

Send your CV and a transcript of your university grades to:

`mailto:Radu.Iosif@univ-grenoble-alpes.fr`

References

[1] A. Basu, S. Bensalem, M. Bozga, J. Combaz, M. Jaber, T. Nguyen, and J. Sifakis. Rigorous component-based system design using the BIP framework. *IEEE Software*, 28(3):41–48, 2011.

[2] M. Bozga, R. Iosif, and J. Sifakis. Local reasoning about parametric and reconfigurable component-based systems, 2019.

[3] J. S. Bradbury. Organizing definitions and formalisms for dynamic software architectures. Technical report, In Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems Newport, 2004.

[4] R. El Ballouli, S. Bensalem, M. Bozga, and J. Sifakis. Programming dynamic reconfigurable systems. In *FACS'18*, pages 118–136, 2018.

[5] A. Taivalsaari, T. Mikkonen, and K. Systä. Liquid software manifesto: The era of multiple device ownership and its implications for software architecture. In *Proceedings of the 2014 IEEE 38th Annual Computer Software and Applications Conference*, COMPSAC 14, page 338343, USA, 2014. IEEE.