Self-Growing Software from Architectural Blueprints

Ada Diaconescu
CNRS LTCI, Télécom ParisTech
Paris, France
<name.surname> @ telecom-paristech .fr

Bassem Debbabi and Philippe Lalanda
LIG laboratory, University of Grenoble
Grenoble, France
<name.surname> @ imag .fr

Software applications are commonly built as a set of heterogeneous, interconnected modules. In the presented research, the challenge we're addressing is rooted in two seemingly antagonistic requirements of such systems. First, computing systems must continuously provide a predefined set of business functions and QoS properties. Second, computing systems must often modify their internal composition and configuration, in order to survive and adapt to runtime changes. Transforming a system's structure and contents during its execution while not impeding on its core functionalities constitutes a difficult and risky task, at best.

From this perspective, we are developing an autonomic framework – CUBE (cube.forge.imag.fr), for enabling software systems to self-grow their internal compositions so as to preserve their functions in various contexts. CUBE was designed considering the following system requirements: 1) a software system must always ensure a *Core Set* of functions and QoS properties; 2) a system may be executing at various degrees of optimisation, offering an additional *Optional Set* of functions and QoS properties; 3) a system must *survive* and *adapt* to changes in its internal composition and external execution environment (so as to continue to ensure 1 and 2). Hence, CUBE aims to enable software systems to self-grow into *instances* that provide their Core requirements, while maximising Optional requirements. Moreover, CUBE endows software systems with several Self-* capabilities – e.g. self-repair, self-optimisation and self-adaptation. Self-* processes enable systems to dynamically transform themselves so as to still meet their requirements when changes occur.

For achieving these capabilities, CUBE relies on the ***decentralised runtime instantiation*** of ***architectural blueprints***. An *architectural blueprint*, defines the Core Set of constraints that all system instances must meet – i.e. a set of heterogeneous types, interconnections and QoS properties. A *decentralised interpreter* consisting of multiple *Autonomic Managers (AMs)*, or agents, concurrently creates an overall system instance that conforms to the architectural blueprint, while being adapted to the current environment. Namely, each AM creates: a partial instance for one blueprint section; and, further AMs for handling adjacent blueprint sections. Existing AMs self-organise so as to interconnect their partial instances into a coherent system. In this manner, the self-growing process progresses from several initial AMs to an entire software application. As changes occur (e.g. internal component crashes or lack of external resources) the concerned AMs transform the existing instance for the new conditions, while still expressing the blueprint (archetype).

In this approach, the architectural blueprint represents the generic features that will inevitably occur in all system instances. Yet, each system instance can be unique with respect to various composition and configuration details (e.g. exact module implementations or number of instances). While archetypal constraints are imposed by system designers, instantiation details are left to the self-* processes to work-out during runtime. Hence, CUBE combines the control capabilities of "traditional" software engineering methods for ensuring Core system properties (i.e. what can be known at system design time), with the flexibility characteristics of self-organising methods for ensuring survivability and self-adaption features (i.e. what can*not* be predicted in advance and must be decided at runtime).

To better illustrate our approach we make an analogy with a natural system - i.e. trees, which seem to feature many of the qualities we are aiming for. In this approximate analogy, a tree can be viewed as a self-growing system whose Core capability is the extraction and self-organisation of surrounding resources (e.g. C, O, H, N, ... + solar energy) into a certain structure. This structure is capable of: surviving - using resources to stay alive; growing - self-organising extra resources to enlarge itself; and self-replicating - producing seeds for reproducing itself. A set of self-* capabilities ensures that trees can carry-out their Core functions in a wide range of environmental conditions. E.g., self-repair - use resources to fix injuries; self-optimisation - maximise growth and optimise shape so as to gather more resources and create more seeds.

While each tree is different (i.e. different *phenotypes* - crown and trunk shapes and sizes, or number of seeds) all trees feature a common set of traits that ensure the same Core characteristics (e.g. all trees have a trunk and a crown; and will produce seeds). This important property is warranted by the common *genotype* that generally "defines" all trees of a given species. In this context, the genotype can be viewed as the system's design part that is considered "worth" keeping and instantiating (and possible to keep and instantiate) across generations. This part has proven successful in the past and hence provides an advantage if imposed as a starting point in the future. Its absence would imply starting from zero at each instance, and evolving only as far as each individual's lifetime permitted. This would severely limit supported system complexity. Finally, the life-time process that maps a genotype into individual phenotypes ensures the individual adaptability needed within a targeted environmental space.

Similarly, CUBE'S architectural blueprint will define well-proven, core design patterns, which will provide an essential starting point for self-organising systems. Imposing such archetypes guarantees that self-organising systems will provide their core functions. Hence, we believe this solution may enable the applicability of self-organising approaches to more functionally-complex computing systems.

This work has been funded by MINALOGIC (Rhône-Alpes) as part of MEDICAL research project