Chapter 25
Robust Network Services with Distributed Code Rewriting

Thomas Meyer
University of Basel, Switzerland

Christian Tschudin
University of Basel, Switzerland

ABSTRACT

Nature does not know the concept of a dedicated controlling instance; instead, "control" is an emergent phenomenon. This is in stark contrast with computer networking where protocol control loops are (seemingly) in charge: while the functional aspect of a networking service can be well mastered, the dynamic behavior is still difficult to understand and even control. In this chapter, we present a methodology how to design distributed software systems that are dynamically stable and robust in execution. It is based on continuously replicating a system's own code base in order to thwart unreliable execution and even accidental code changes. The crucial part is to design the system such that it regulates its own replication. This can be achieved by an execution environment inspired by chemistry to which we add the concept of self-rewriting programs (Quines). With a link load balancing example we show how to exploit competition and cooperation in a self-rewriting service implementation.

INTRODUCTION

Natural systems are able to dynamically construct redundancy by assembling and reproducing their components. Often, components exist in several copies (flocks, but also blood or nerve cells), exploiting parallelism and minimizing the impact of the loss of a single item. For singular components (e.g. bones) and in order to fight the problem of aging, redundancy is achieved over time through procreation, yielding a new and possibly modified copy. In computer science however, software is considered to be static (and without wear). This view is recent: Back in the 1940s, von Neumann (1966) developed a theory of self-reproducing automata. He described a universal constructor, a machine able to produce a copy of any other

DOI: 10.4018/978-1-4666-4301-7.ch025

machine whose soft- and hardware blueprint is provided as input. Being universal, the constructor is also able to generate a copy of itself.

Considerable research on self-replication was carried out on the framework of Cellular Automata (CA), in which remarkable results were achieved, also in terms of robustness and self-repair (Tempesti, Mange, & Stauffer, 1998). However, these results are hard to transfer from CA to the world of today's computer software. In the 1960s, with the desire to understand the fundamental information-processing principles and algorithms involved in self-replication, researchers started to focus on self-replicating code: how textual computer programs are able to replicate independent from their physical realization. The existence of self-replicating programs is a consequence of Kleene's second recursion theorem (Kleene, 1938), which states that for any program *P* there exists a program *P'*, which generates its own encoding and passes it to *P* along with the original input. The simplest form of a self-replicating program is a Quine, named after the philosopher and logician Willard van Orman Quine, and made popular by Hofstadter (1979): A Quine is a program that prints its own code. Quines exist for any programming language that is Turing complete and it is a common challenge for students to come up with a Quine in their language of choice. The *Quine Page* provides a comprehensive list of such programs in various languages (Thompson, 2010).

Contribution

In this work, we put Quines in a parallel execution environment, permitting an ensemble of Quine copies to achieve surprising robustness with respect to code and packet loss and even execution errors. Our contribution consists in the demonstration of an operational system based on Quines that runs highly reliable network services with provable dynamic properties. More precisely, we will introduce an artificial chemistry embodied as interconnected "molecule vessels" in which we place carefully crafted self-replicating programs. Packets, or "molecules", react with each other and produce new packets, thus executing the program. Useful computations are piggybacked to the Quine structures in order to implement the network services. Due to the special scheduling of the reactions in the artificial chemistry according to the "law of mass action" in real chemistry, our system inherits the dynamic properties from chemistry such that we can apply the related analysis tools that were developed in the past two centuries. The law of mass action links the microscopic (scheduling) events with the observable behavior at macro scale. Using perturbation analysis, we can then proceed in identifying equilibria and their stability.

Structure of this Chapter

We present our work along the following argumentation path: After having highlighted the context of our approach and related work, we proceed with introducing a new "style" of implementing network services that we call *chemical networking protocols*. Next, we present "chemical Quines" and we extensively study their long-term stability, both in a single node as well as in a distributed setting. We also look at competing Quines and show that the aggressive growth of Quines leads to a winner-takes-all dynamics. We then investigate cooperative couplings of Quines and put these insights to work with a link-load-balancing service for which we show its resilience to packet *and* code loss.

CONTEXT AND RELATED WORK

In this section, we reference the relevant corner stones for our work where we could draw important insights, namely self-reproduction, fault tolerance, artificial chemistries and their dynamics, the dynamics of competing populations and finally cooperation patterns. The programming language "Fraglets", which we have used to implement our