MARK BURGESS

# configuration management: models and myths

## PART 1: CABBAGE PATCH KISS

Mark Burgess is professor of network and system administration at Oslo University College, Norway. He is the author of cfengine and many books and research papers on system administration.

*Mark.Burgess@iu.hio.no*

**WHEN I WAS EIGHTEEN AND FRESH** out of school, I worked for a couple of summers as a gardener in the well-to-do houses and estates of the English countryside, together with a small team of other gardening hands. Each sunrise, we were transported at breakneck speeds around the veins of the Midlands in a rusting station wagon (appropriately known as "estate cars" in the U.K.) by my boss, Ken, a soon-to-retire adventurer with a British Air Force moustache and a love of music and the outdoors. We would discuss the great Russian composers as we clipped away at hedges and tilled over flowerbeds.

As workers, we were somewhat human in our modus; it was occasionally precarious work, and we were not altogether skilled or efficient, especially after the lunch breaks during which we would ritually consume as many rounds of beer at the local pub as there were workers on that particular day (a matter of honor, and not something that I could conceivably survive at present). These were "Bruckner episodes," as Ken noted, ever trying to get me to listen to that rather mediocre composer, whose work he referred to as "traffic-light music." I later learned this meant that just when you thought it was about to finally go somewhere, it would stop, dither, and then attempt to start again. Quite.

I believe I learned several good lessons about maintenance from my stint in the English Country Gardens. The first was "Always let your tool do the work," as Ken used to point out, with a nudge and a wink and then a guffaw. In other words, know how to use the tools of the job rather than breaking your back with manual labor (aside from various carnal interpretations).

The second was about careful execution. Gardening is nothing if not a strenuous job. It seems straightforward, with everything under control, until you mistakenly cut a prize flower in two or fall foul of a mix-up—"Oh, I thought she said *do* destroy the garden gnome"—enshrined among the taskforce as "Always read the destructions." On one occasion, enthusiastic overzealousness was cut short when a friend of mine stepped backward onto a rake (in classic Tom-and-Jerry style) and emptied a wheelbarrow full of earth into the client's newly filled swimming pool. This was not

policy, but we liked to think of it as one of those inevitable once-in-a-lifetime (or -workday) freak accidents. Yes, we would have been naive to believe otherwise.

Gardening was, I think, my first exposure to the idea of *configuration management*, that is, the routine maintenance of everything from the most sublime stately garden to the most ridiculous cabbage patch. Size is not important; the same errors are made in spite of scale. It is very much about seeing how the result of extensively planned and orchestrated order generally falls foul of both growing weeds and the misguided hands of the reputedly infallible maintainer. (In spite of the best-made plans of pubs and men, anything that can go wrong is not necessarily to be blamed on a pint of Murphy's.)

In this series I want to talk about configuration management in system administration from the perspective of a researcher in the field. This is my cabbage patch, a topic that I have been interested in for some fifteen years now; it is also a fascinating theoretical and practical problem, which has been overshadowed in recent years by tool talk and XML incantations. Yet there is much to be said about this topic from a research point of view, and all those eager to rush out and build their own tool should pause to take heed of what has been learned. It's not about tools; it's about principles and assumptions, just as it's about knowing the limitations and learning to deal with them. So I want to dangle a carrot of rationality in front of you, to turn the discussion away from the cosmetics of tools back to the science of the problem.

## What Is a Configuration?

As with any Geek Tragedy we start by introducing the *dramatis personae* and speaking the names of our daemons to show that we do not fear them. We begin with the notion of the *configuration* itself. This seems perhaps to be a rather obvious concept, but, as we know from bitter experience, assuming the obvious to be uninteresting is the best way to avoid seeing the wood for the trees. So what is configuration about? Here are some definitions for configuration, which I found rustling through the weeds:

> The appearance formed by the arrangement and relation of parts.
>
> An arrangement or layout.
>
> Functional or physical characteristics of hardware/software as set forth in technical documentation (from a software document).
>
> The specific assemblage of components and devices that makes up the components of a complete system.

Of these, I like the first and the last the best. It is clear that configuration has to do with identifiable patterns and how the pieces fit together to make a whole. In the last definition we also mix in the idea of a system, that is, that a pattern might actually perform a role within some functional mechanism.

## What Is Configuration Management?

The IEEE Glossary of Software Engineering Terminology (Standard 729-1983) defines configuration management as:

> the process of identifying and defining the items in the system, controlling the change of these items throughout their life-cycle, record-

ing and reporting the status of items and change requests, and verifying the completeness and correctness of items.

This definition is a software engineering definition, but it captures the main features of what we understand by host or network configuration management. I like especially that the definition talks about the process surrounding the configuration (i.e., management) and not merely the definitions or the tools used. In particular, the idea of change management is included as part of the process, as is verification, which implies some kind of monitoring.

Often system monitoring is separated from the idea of configuration implementation. Possibly this is because it was originally assumed that implementation would always be executed by humans. Elaborate monitoring systems have been devised, but these are often read-only. As we shall see later in the series, this is a key challenge to be met.

## State and Configuration

In computer science we talk a lot about states. The idea of a state is part of the fundamental theory of computation, and most of us have an intuitive idea of what is meant by the concept, so can we relate configurations to states? What these ideas have in common is that they are all alternative values of a variable quantity, such as a register. A configuration is somehow a pattern formed by a number of state variables. Let's recap the different ideas of state to give the devils their names.

The simplest idea of state is that of a *scalar* value, or a single item. For example, file attributes in UNIX are simple register-like values. When we write `chmod 664 file-object` we are defining the state of the mode register for the file object. The state of the register belongs to the set {000,001,002, . . . ,776,777}, which is called the *alphabet* of the state. We think of each value as a separate symbol of an alphabet.

When the possible values the variable can take are limited, we speak of a finite number of states. An algorithm that uses and manipulates the values is then known as a Finite-State Machine.

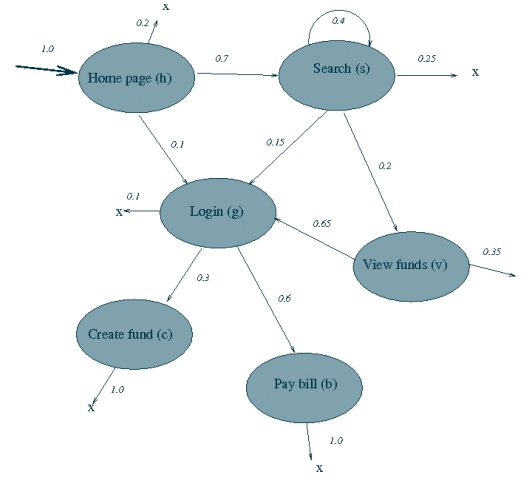


**FIGURE 1: A FINITE-STATE SYSTEM FOR AN INTERNET BANK WEB SITE, WITH TRANSITION PROBABILITIES MARKED IN. ARROWS REPRESENT POSSIBLE TRANSITIONS.**

In a dynamical system, change is represented by changes of state. In other words, certain variables or registers change their values from one symbol in the alphabet to another. The different state values can be shown as locations (see Figure 1), and transitions can be drawn as labeled arrows, where the labels remind us of the cause of the transition (i.e., the operation that was executed that resulted in the change). Most of us remember the state diagrams for process execution in the kernel, labeled with names such as "ready," "waiting," "dispatched," "idle," and "zombie."

Transitions between the states occur as a result of different actions. Sometimes we control those changes, and sometimes we merely observe the changes. This is an important thing to remember, as it is often assumed that once a value has been set, its state remains fixed until we alter it. But because of the external environment of a system, that is not true. We might plant the seeds, but they grow all by themselves, thanks to rain and sun.

What about "containers" more complicated than scalar registers? Well, permissions of more contemporary file systems have more complicated attributes than the UNIX file system. Access Control Lists are *lists* of scalar values, whose state can include lists of user identities to be granted access. The system process table is a list of state information, split into different columns, that is, formed from different categories or *types* of information. These are embellishments on the idea of a list. Okay, so we need lists, but these are not much more complicated than scalars.

*Text files* and *databases*, however, reach a new level of complexity. Text files are ubiquitous for holding configuration information on UNIX-like operating systems. Databases are the other universal approach for storing values.

The contents of files and databases are not of a fixed length, so we cannot think of them as mere registers. If they are ASCII-encoded files, then we can say that they are composed from a finite alphabet but that is not the same as having a finite number of states. File size is only limited by memory.

The most pressing feature of files is that they have *structure*. Many UNIX files have a line-based structure, for instance. XML files, which are rampantly popular, have a structure based not on lines but on a parenthetical grammar. Most programming languages are not reliant on starting a new line for each new statement either: Their formatting is also based on a scheme of statements organized in nested parentheses.

What is important about the structure of information is not what it looks like, but how easy or difficult it is to understand the information in these structures—not just for humans, but for computer algorithms too. Any tool or scheme for determining the content of a file object or database must deal with the real complexity of the structure used by that file object to store and represent information.

Let's abstract this idea to get a better idea of what we mean. With structured state information, we are not just deciding the color of a single rose for our garden; rather, we are planning the landscaping of the entire estate. We cannot expect a single spade to deal with the complexity of this garden. The tools that create and maintain such a structure need considerably more intelligence. This idea has been studied at length in computer science, because it is a problem of far-reaching and general importance.

## Patterns, or Variations of State

As a state changes, either in time or across different locations or objects, it forms a pattern. It paints a picture or sows a garden. The pattern might be

so complex that it seems inconceivably random, or it might form some kind of simple and recognizable structure. The structure might be best comprehended as one-dimensional, two-dimensional, etc. As you see, there are any number of characterizations we might use to accurately describe the pattern.

Arguably the most basic question we can ask is the following: Is the pattern formed from *discrete symbols*, such as individual flowers or entries in a database, as in the specification in Figure 2?

```
xxxxxxx----------
xxxxxxx----------
xxxxxxx----------
xxxxxxx----------
-------------------------
-------------------------
-------------------------
```

**FIGURE 2: DISCRETE SYMBOLS USED TO BUILD DISCRETE SYMBOLS. HERE THE ALPHABET IS {X,-, } AND WE PLANT OUR FLOWER BEDS TO REPRESENT A FLAG.**

Or is it a *continuous variation*, such as the curve of the path through the rolling mounds or the changing level of the system load average (see Figure 3)?

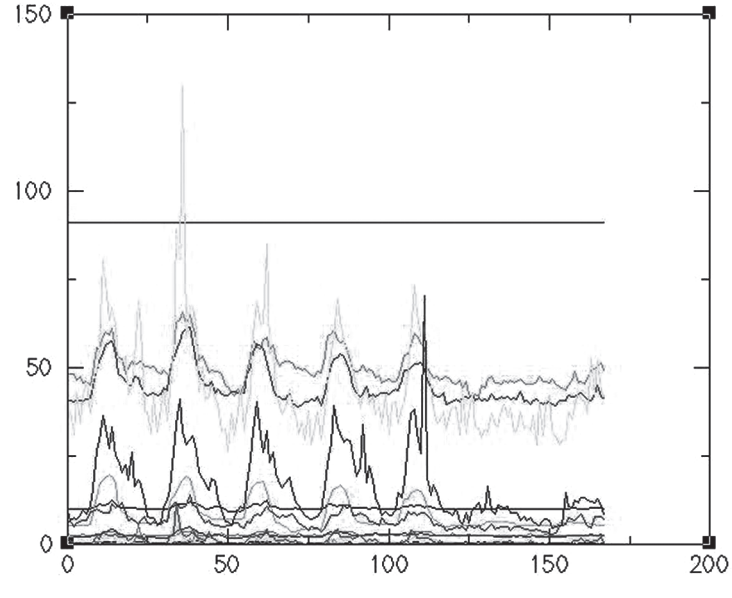


**FIGURE 3: A CONTINUOUS CHANGE OF STATE, SHOWING THE CHANGE IN AVERAGE LOAD LEVEL OF CONNECTIONS TO VARIOUS SERVICES OVER THE COURSE OF A WEEK.**

This choice between discrete and continuous points exemplifies a fundamental dichotomy in science: the competition between *reductionism* and *holism*. That is to say, discrete values generally emerge from trying to break things down into atomic mechanisms, whereas continuous values come about by stepping back for a broader, approximate view. Variables that try to capture average behavior, for instance, vary approximately continuously (as in Figure 3). The theory for describing patterns in these categories is

rather different in each case, which explains why intrusion detection and anomaly detection are so difficult.

Patterns are everything to us, not just in configuration management, but in the management of anything. We organize things according to patterns. Patterns are how we understand when things are the same or different. There are patterns in time (repetition, scheduling, etc.) and there are patterns in space (e.g., disk management, directory management, and cluster management). When we manage things poorly it is often because we have no good way of describing and therefore measuring the patterns of the resources that we need to deal with. Indeed, it is a paradox that beings that are so good at recognizing patterns are often quite inept when it comes to utilizing them. This is odd indeed, because humans are excellent language processors, and the principal way of describing discrete patterns is with language. The study of computer languages is the study of patterns. Continuous patterns are an essential part of our visual and auditory recognition. The language of these is the calculus of differential mathematics.

What discrete and continuous patterns have in common is that patterns of either kind are difficult to identify. Our brains are incomprehensibly successful at identifying patterns (so much so that we see patterns even when they are not there), but finding algorithms for recognizing and even classifying patterns and for associating meanings with them (semantics) can be one of the most difficult problems to solve.

Does that mean we should not try? Is management a waste of time? Clearly it is not. Much has been accomplished throughout history by our willingness to forego complexity and employ simple patterns that we can comprehend more easily. Of course, tackling this issue involves some sacrifice, but identifying the patterns offers greater predictability and hence reliability. It is an important lesson that the real intellectual achievement is to simplify a problem to its core essence—anyone can make something more complicated. Indeed, science or natural philosophy is about looking for *suitably idealized approximations* to complex patterns, not about wallowing in detail. Also in configuration management, we must forego complexity to achieve reliable management. This is a theme I'll be discussing in future issues.

## COM:POSTscript

At risk of turning this into a Bruckner episode, we must leave it there for this time, before the fruits of this batch get us embroiled in a jam. In the next part of the series I want to talk about the ways in which patterns can be classified by algorithms. This is an important step toward the automation of configuration management. We'll see why the only real tool we have for pattern matching symbolically is the regular expression and why intrusion-detection systems are trying to solve a hopeless task—identifying who is a pod among the gardeners!

Gardens can be impressive tiered sculptures full of botanical variety, jungles rich in diversity, or scrapyards full of junk. It's the same with our computer configurations. Which is easier to understand? Which is better to live with? These are not easy questions to answer, but they are among the questions we shall try to address in upcoming issues.