Modelling System Administration Problems with CSPs

John A. Hewson and Paul Anderson

School of Informatics, University of Edinburgh, United Kingdom
john.hewson@ed.ac.uk, dcspaul@ed.ac.uk

Abstract System administrators increasingly use declarative, object-oriented languages to configure their systems. Introducing constraints to such a language and automatically generating a valid system configuration is an area of active research. We describe our work towards creating ConfSolve, an object-oriented configuration language which can describe constraints over valid configurations, solution of which is provided by compilation into a CSP. We evaluate our solution against a simple virtual machine allocation problem, with promising results for automating Infrastructure as a Service (IaaS) systems.

1 Introduction

Configuration of large computing installations is increasingly performed by automated tools, which make use of declarative, object-oriented languages [4,7]. These tools replace low-level shell scripts describing how to achieve a system state, with a high-level declarative model of the goal state of a system. Such tools have proven popular amongst system administrators configuring large sites, and are used to configure workstations, servers, and routing hardware. Infrastructure is increasingly virtualised, allowing more of it to be dynamically reconfigured to provision new services, reduce consumed resources, or respond to a hardware failure. Yet much of this work is done by hand, which is inefficient and error-prone.

Recently there has been interest in extending such tools to include variables which are not given an explicit value by a system administrator, but instead have constraints specified over them. Research into implementing basic versions of such systems has made use of constraints solved via SAT [8,5], the ECLiPSe [4] constraint logic programming system, and an adapted form of CSP [2].

There is a need for a general-purpose system configuration language which can be used to model a broad range of configuration problems involving constraints, in an easy-to-use manner. As the current state-of-the-art system configuration languages are object-oriented, we believe that any such configuration language must also be. It is desirable to compile constraints written in such a language into a lower-level description of the problem, for which CSP is a natural candidate.

We have developed a proof-of-concept system configuration language, in which constraints over valid solutions are described, and valid concrete configurations generated, via a CSP solver. Our goal is to create a language general enough to describe a large range of configuration problems, in a manner natural to a system administrator, and simpler than writing a model in a constraint programming language. This paper describes our initial work towards that goal.

2 The ConfSolve Compiler

The architecture of the ConfSolve compilation process is shown in figure 1. First, a system configuration specification written in ConfSolve is compiled into the standard constraint programming language MiniZinc [6]. It is then solved using the Gecode [3] constraint solver, the output of which is parsed back into the ConfSolve compiler, and combined with the original ConfSolve model to produce structured text output similar to JavaScript Object Notation (JSON). We use the structured text to generate configuration files for the system configuration tool Puppet [7], and the visualisation tool GraphViz. By generating configuration files for Puppet, we avoid the need for ConfSolve to perform system configuration tasks itself, so we need concern ourselves only with the compilation process.

Figure 1. The architecture of ConfSolve. A ConfSolve instance is a structured text file representing a single solution.

2.1 The ConfSolve Language

ConfSolve aims to bridge the gap between existing object-oriented system configuration languages and solver input languages such as MiniZinc. It is designed to be as familiar to system administrators as possible, providing object-orientation with classes, inheritance, reference types, and enumerations, as well as a more Java-like syntax, though this is still evolving. The remainder of the section introduces the ConfSolve language by way of an example. A full grammar is given in figure 2; we intend to provide a complete formal description in a later paper.

An example system administration task which could benefit from constraint solving is the allocation of virtual machines onto an array of homogenous servers. This set-up is typical of enterprises which have adopted IaaS (infrastructure as a service), or cloud computing datacenters. A virtual machine is referred to as a *role* and a physical machine as a *machine*; the goal is to allocate each role to