# **Constraint-based Specifications for System Configuration**

THE UNIVERSITY of EDINBURGH INFORMATICS

John A. Hewson The University of Edinburgh

Microsoft Research

# Configuration is complex, errors are expensive

62% of enterprise network downtime is caused by human error

Vankee Group (2004)

\$100,000 minimum estimated loss per hour of downtime for 42% of US organisations

28% of network disruptions at Google are caused by configuration error

The Datacenter as a Computer (2000)

\$2,000 estimated loss every second at PayPal in a one hour hardware outage

The Risk Factor, IEEE Spectrum Blogs (2009)

# A unified, constraint-based approach is proposed

### **1** Business Policy

Human-readable, high-level declarative constraint language. Hierarchical *components* provide re-usability and structure similar to object-oriented programming languages. *Type refinement* allows constraints to be tightened on subtypes. Typed first order logic is used to specify constraints.



Intermediate representation of the policy, in a standardised XML format, intended to be read only by machines. Purpose is to decouple the policy language and compiler from the CSP Solver (step 4) to allow other new or existing policy compilers to target the rest of the framework (steps 4-7)

**5** Configuration Model

Standardised XML representation of the configuration values found by the CSP Solver. Decouples the CSP solver and application-specific configuration file generation.

Allows other tools to target the output translators (step 6).

**7** Configuration Files

Application-specific or system-specific configuration files, in any required format. Standard formats such as XML, SML, CIM MOF, or UML diagrams.















## 2 Policy Compiler

Compiles the policy language into a standardised intermediate language. Syntax checking and basic type checking are performed. Basic errors are reported to the user quickly. Multiple policy files are processed as one large business-wide policy. Could be implemented in any programming language.

### **4** Generative CSP Solver

Extends the **Constraint Satisfaction Problem (CSP)** method with the ability to generate new variables and constraints dynamically, in a manner similar to object instantiation in object-oriented programming.

**Decidable**: a solution can always be found if it exists, as all domains are finite, and the component hierarchy is acyclic and non-recursive

Component hierarchy prevents an exponential increase in CSP search space.

- **Compatability constraints** restrict attribute values, such as x < 7.
- Resource constraints add a cost to a component, and limit its upper bound.
- **Soft constraints** indicate a preference for particular values (Heuristic).
- **Optimisation constraints** define a function which must be maximised.

## 6 Output Translators

Translates the standardised XML configuration model into application-specific configuration file formats. Alternatively, could perform some configuration action. Can be quickly implemented in any programming language.

# High-level configuration is a complex problem

### Autonomics

The process above is designed to be encapsulated in a self-managing *autonomic element*, allowing distributed and hierarchical control of sub-systems. No specific architecture or schema is required when re-using our framework, a major limitation with existing systems.

### Reconfiguration

When the business policy changes, the solver should find the configuration closest to the current solution, to cause minimal changes. We plan to investigate using local search and heuristics within the CSP solver.

### Federation

Policy and configuration information can come from a variety of sources inside an organisation. A constraint-based system can automatically synthesise composite constraints and detect conflicts.

### Security and Validity

As constraints are defined on each component, and the solver is computationally complete, any configuration which compiles successfully is guaranteed to be valid under those constraints. Compliance with a security policy can therefore be proved.

#### About

First year PhD research. Work so far has been identifying a methodology with sufficient expressivity which is decidable, and yet conceptually simple.

### Work Remaining

Implement the above system using F#, including the design of a suitable policy language, type checker, and solver.