Constraint-Based Specifications for System Configuration

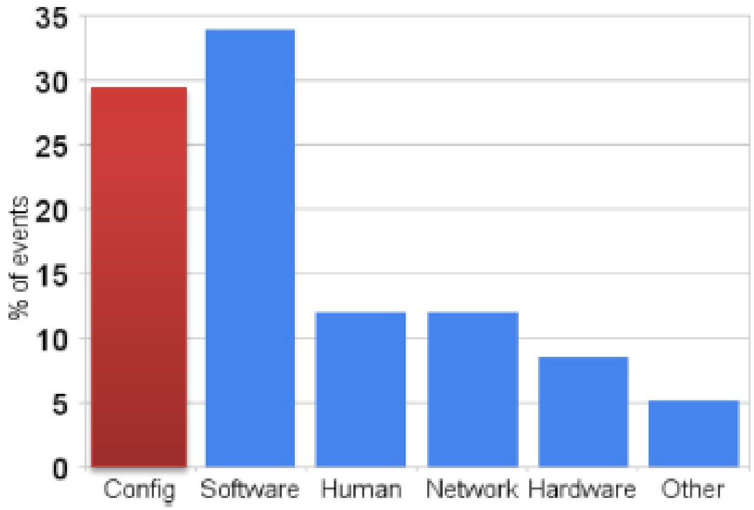
John A. Hewson

Feb 14, 2011

Overview

- Cloud and IaaS configuration
- State-of-the-art: Declarative languages
- Modelling an IaaS problem
- Solving with CSP
- Future work: Semantics, usability, advanced features

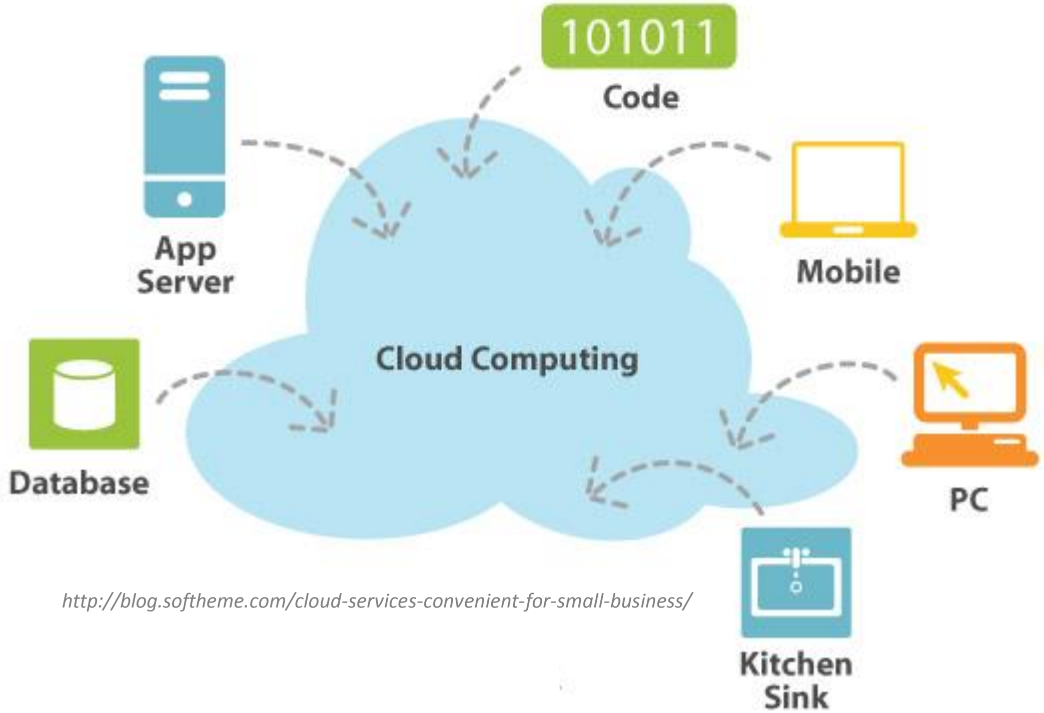
Configuration Errors Matter



Service disruption events by most likely cause at one of Google’s main services, over 6 weeks (2009)

The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Hoelzle & Barroso, 2009.

Cloud Computing = Platform as a Service (PaaS)



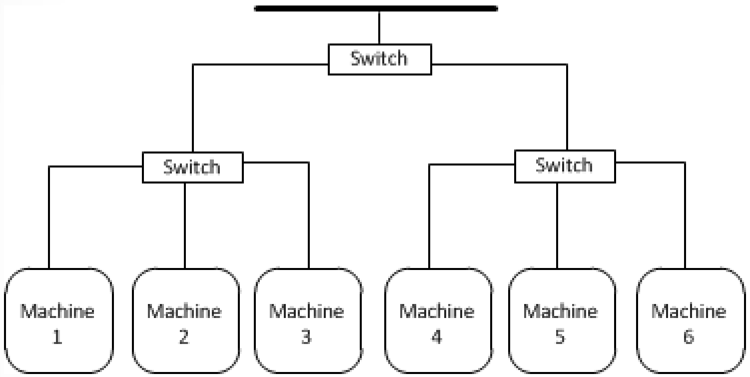
http://blog.softheme.com/cloud-services-convenient-for-small-business/

e.g. Amazon S3, Google AppEngine. Not off-site VMware or Xen.

Why? Because individual cloud machines are not meant to be reliable.

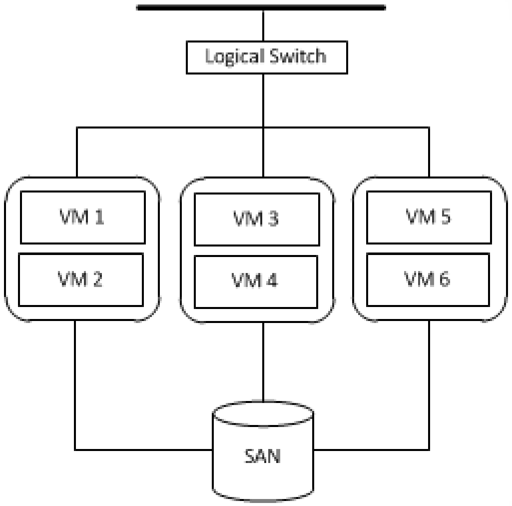
Infrastructure as a Service (IaaS)

Traditional



vs.

IaaS



- Saves on infrastructure costs (both CapEx and OpEx)
- VMware is used by 98% of Fortune 500 companies
  http://www.vmware.com/company/customers/
- Can even move running VMs in near-realtime

Configuration – solutions?

Rise of declarative tools for UNIX:

- LCFG (1993, Anderson, University of Edinburgh)
  - configures your DICE machine!
- Cfengine (1993, Burgess, Oslo University College)
- Bcfg2 (2004, Desai, Argonne National Laboratory)
- Puppet (2005, Kanies, Independent)

THE UNIVERSITY of EDINBURGH informatics

Puppet

- Used at major web companies: Twitter, match.com, Zynga
- Open Source (GPL)
- Configures UNIX-like systems, abstracting over differences
- Declarative language. For example, we write

```
package {'apache':
    ensure => installed
}
```

instead of

```
sudo apt-get -y install apache
```

What's Missing?

- Constraints!
- The ability to *verify* that a configuration conforms to a model
- The ability to ***infer*** valid configurations from a model
  - Much more powerful
  - Now required for IaaS and cloud-scale systems, as the problems are too time-consuming for humans to solve.
- *Let's look at an example…*

Some IaaS Problems in the Enterprise

- How can we assign VMs to physical machines?
  - With CPU, RAM, I/O requirements
  - With co-location requirements (*e.g.* distribute redundant VMs)
  - In Compliance (*e.g.* following credit card data rules)
  - Following Firewall rules (or changing them)
- How can we optimise:
  - The VM assignments above
  - Latency between pairs of machines
  - Power consumption
  - Licensing (*e.g.* per-CPU)
  - Robustness (*e.g.* redundancy)
  - Performance (*e.g.* database cache)
  - SLAs (*e.g.* minimise cost of violation)

Example: Problem

- Service–Machine Allocation
- 4 Services
- 3 Machines
- Each machine has a fixed:
  - *Scalar* amount of RAM
  - *Scalar* number of CPUs
  - *Boolean* set of capabilities (*e.g* RAID5, Gigabit Ethernet)
- Each service has fixed requirements over these values

- **Q:** *Which services run on which machines?*

Existing Machine Capabilities

Machine	Capability	MachineCapabilities
Monster	IsIISEnabled	0
Monster	IsSQLEnabled	1
Monster	HasDualProc	1
Monster	HasQuadProc	1
Monster	HasRAID5	1
Monster	HasGigEther	0
Chatter	IsIISEnabled	1
Chatter	IsSQLEnabled	0
Chatter	HasDualProc	0
Chatter	HasQuadProc	0
Chatter	HasRAID5	0
Chatter	HasGigEther	1
Typical	IsIISEnabled	0
Typical	IsSQLEnabled	0
Typical	HasDualProc	1
Typical	HasQuadProc	0
Typical	HasRAID5	1
Typical	HasGigEther	0

Existing Machine Metric

Machine	Metric	MachineMetric
Monster	Memory	16384
Monster	CPU	12
Chatter	Memory	1024
Chatter	CPU	2
Typical	Memory	2048
Typical	CPU	3

Microsoft Solver Foundation - http://www.solverfoundation.com/

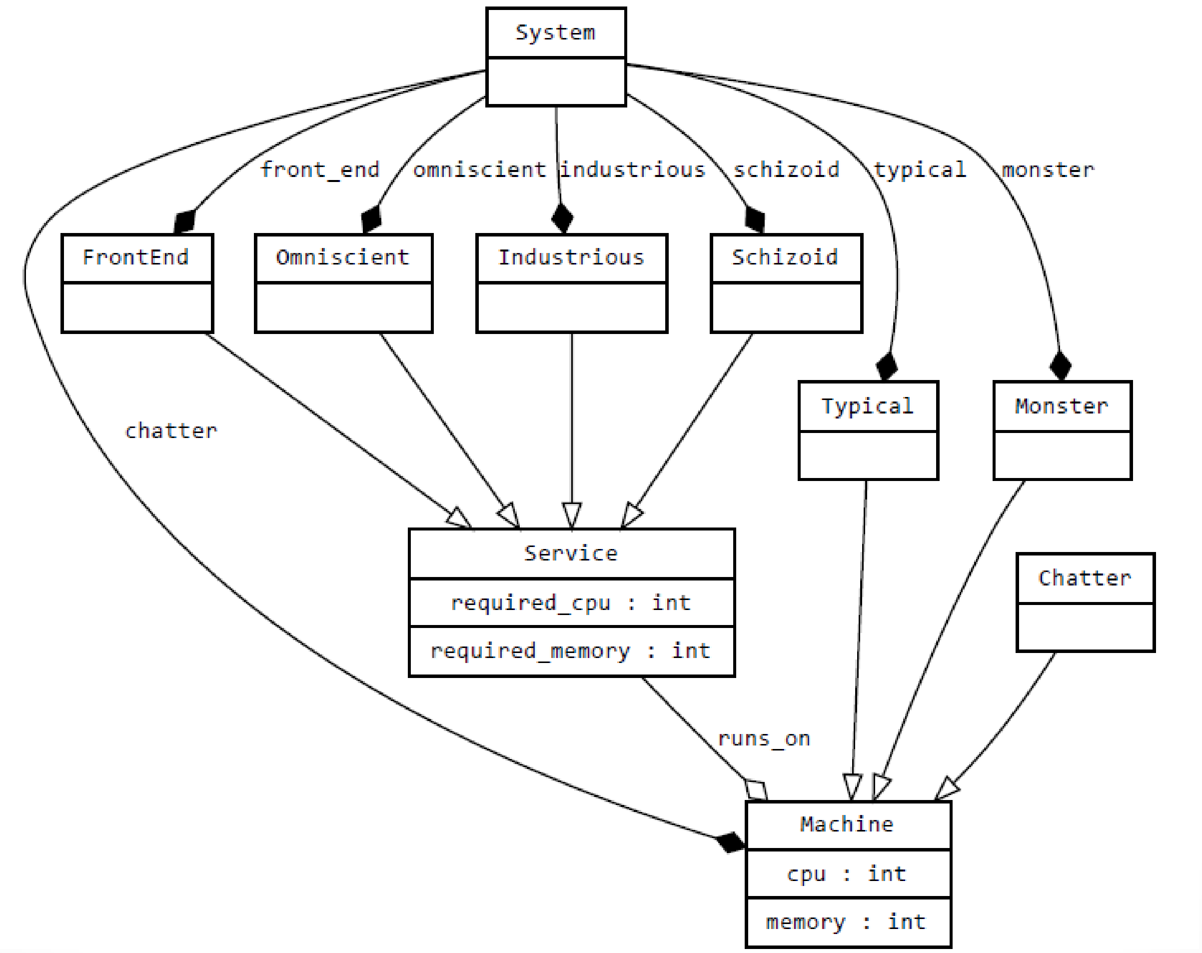
Service Minimum Requirement (Capabilities)

Service	Capability	ServiceCapabilities
Omniscient	IsIISEnabled	0
Omniscient	IsSQLEnabled	1
Omniscient	HasDualProc	0
Omniscient	HasQuadProc	0
Omniscient	HasRAID5	1
Omniscient	HasGigEther	0
FrontEnd	IsIISEnabled	1
FrontEnd	IsSQLEnabled	0
FrontEnd	HasDualProc	0
FrontEnd	HasQuadProc	0
FrontEnd	HasRAID5	0
FrontEnd	HasGigEther	1
Industrious	IsIISEnabled	0
Industrious	IsSQLEnabled	0
Industrious	HasDualProc	1
Industrious	HasQuadProc	0
Industrious	HasRAID5	0
Industrious	HasGigEther	0
Schizoid	IsIISEnabled	0
Schizoid	IsSQLEnabled	0
Schizoid	HasDualProc	1
Schizoid	HasQuadProc	0
Schizoid	HasRAID5	0
Schizoid	HasGigEther	0

Service Minimum Requirement (Metric)

Service	Metric	ServiceMetric
Omniscient	Memory	4096
Omniscient	CPU	6
FrontEnd	Memory	512
FrontEnd	CPU	1
Industrious	Memory	512
Industrious	CPU	1
Schizoid	Memory	1024
Schizoid	CPU	2

Example: Specification (Classes)



Example: Specification

```
component Machine {
   var cpu as int;
   var memory as int;
}

component Service {
   var required_cpu as int;
   var required_memory as int;
   var runs_on as ref Machine;
}

...
```

THE UNIVERSITY of EDINBURGH informatics

Example: Specification

```
component FrontEnd extends Service {
   where required_cpu == 1;
   where required_memory == 512;
   where required_capabilities == {IsIISEnabled, HasGigEther};
}

component Monster extends Machine {
   where required_cpu == 1;
   where required_memory == 512;
   where required_capabilities == {IsIISEnabled, HasDualProc,
                                    HasQuadProc, HasRAID5};
}

...
```

Example: Specification

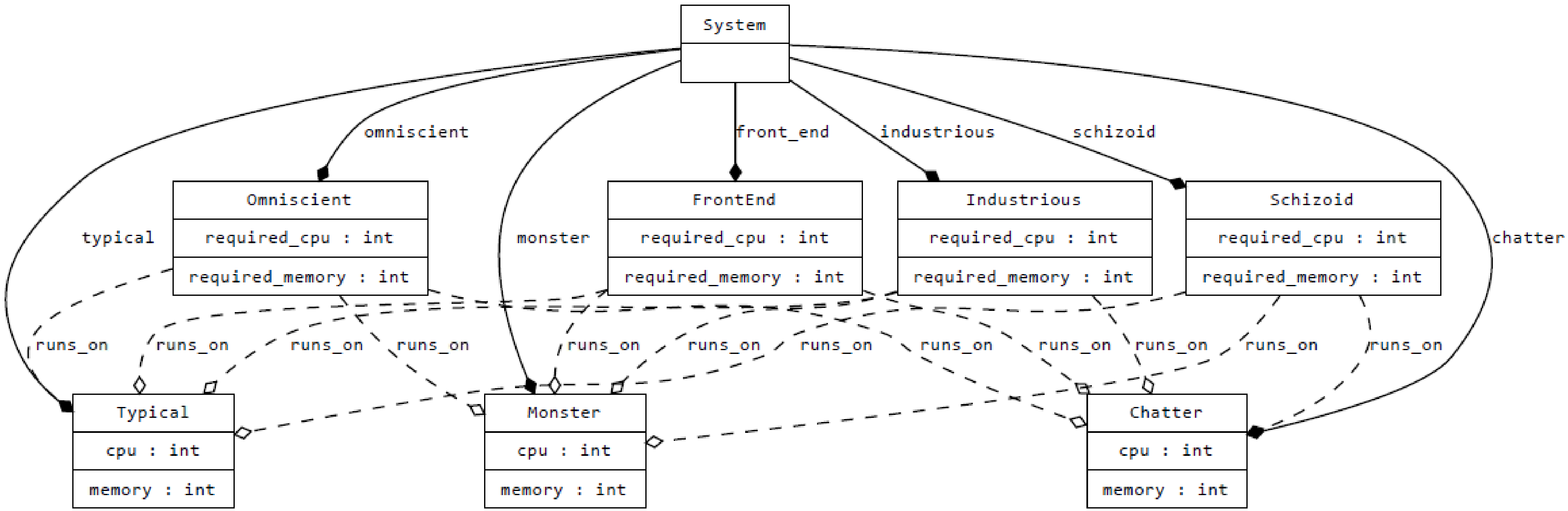
```
root component System {
    var typical as Typical;
    var monster as Monster;
    var chatter as Chatter;

    var front_end as FrontEnd;
    var omniscient as Omniscient;
    var industrious as Industrious;
    var schizoid as Schizoid;

    ...
}
```

THE UNIVERSITY of EDINBURGH informatics

Example: Specification

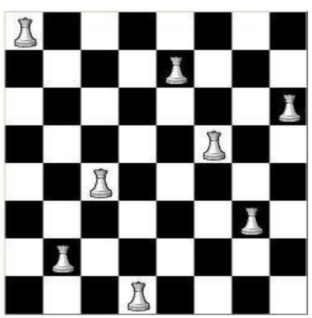
```
var machines as (ref Machine)[3];
var services as (ref Service)[4];

foreach(m in machines, s in services where s.runs_on == m)
{
          sum(s.required_cpu) <= m.cpu &&
          sum(s.required_memory) <= m.memory &&
          s.required_capabilities in m.capabilities;
}
```

Example: Specification (Instances)



Constraint-Satisfaction Problem (CSP)

- Closely related to SAT and SMT solvers.
- Problem is described as a sets of variables, domains, and constraints.
- Everything is finite – complete, decidable. *Very desirable properties.*
- Modern solvers also support optimisation, local search, and soft constraints.
- *N*-queens problem: or Sudoku:



The Code Project

8	6				9	4	7	
			5	3	4			
1			8	7				9
		1	9	8		7		
			7		2			
		7		4	5	3		
6				5	1			8
			4	9	8			
	9	8	2				4	5

http://radialmind.blogspot.com

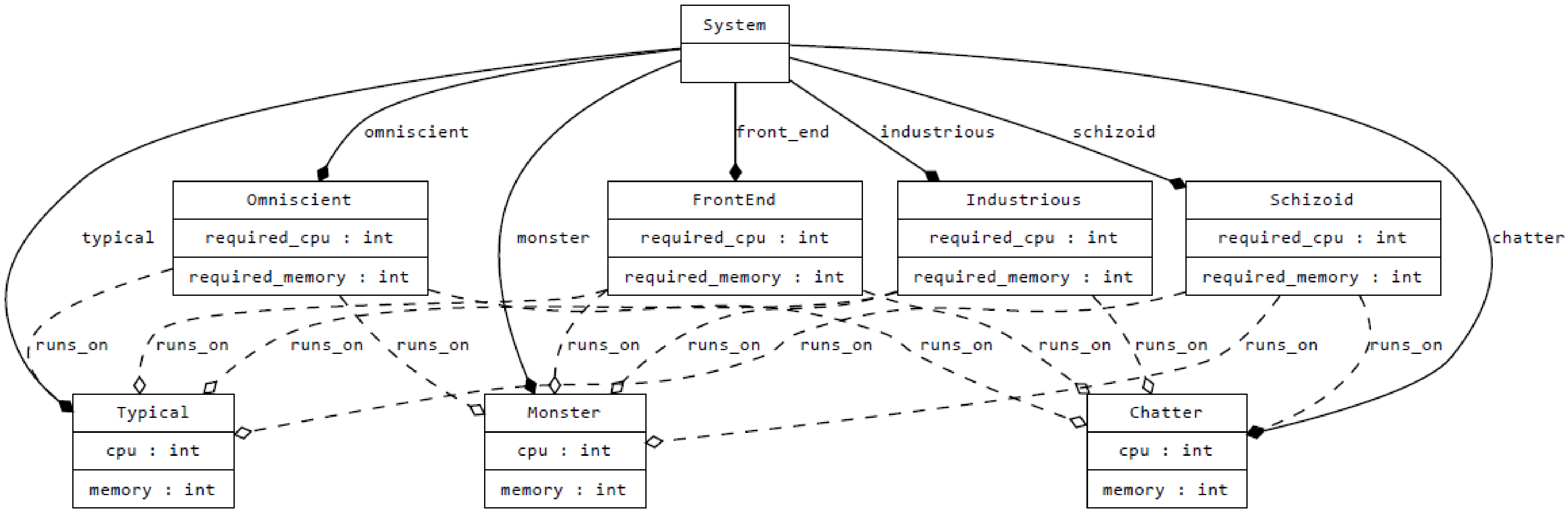
Auto-Generated CSP Code (MiniZinc)

```
/* variables */
var int : root_typical_cpu;
var int : root_typical_memory;
var int : root_monster_cpu;
var int : root_monster_memory;
var int : root_chatter_cpu;
var int : root_chatter_memory;
var int : root_front__end_required__cpu;
var int : root_front__end_required__memory;
var int : root_omniscient_required__cpu;
var int : root_omniscient_required__memory;
var int : root_industrious_required__cpu;
var int : root_industrious_required__memory;
var int : root_schizoid_required__cpu;
var int : root_schizoid_required__memory;
var {1, 2, 3} : root_front__end_runs__on;
var {1, 2, 3} : root_omniscient_runs__on;
var {1, 2, 3} : root_industrious_runs__on;
var {1, 2, 3} : root_schizoid_runs__on;

/* constraints */
/* System */ constraint (((((bool2int((root_front__end_runs__on = 2)) * root_front__end_required__cpu) + (bool2int((root_omniscient_runs__on = 2)) *
root_omniscient_required__cpu)) + (bool2int((root_industrious_runs__on = 2)) * root_industrious_required__cpu)) + (bool2int((root_schizoid_runs__on = 2)) *
root_schizoid_required__cpu)) <= root_monster_cpu);
/* System */ constraint (((((bool2int((root_front__end_runs__on = 2)) * root_front__end_required__memory) + (bool2int((root_omniscient_runs__on = 2)) *
root_omniscient_required__memory)) + (bool2int((root_industrious_runs__on = 2)) * root_industrious_required__memory)) + (bool2int((root_schizoid_runs__on = 2)) *
root_schizoid_required__memory)) <= root_monster_memory);
/* System */ constraint (((((bool2int((root_front__end_runs__on = 1)) * root_front__end_required__cpu) + (bool2int((root_omniscient_runs__on = 1)) *
root_omniscient_required__cpu)) + (bool2int((root_industrious_runs__on = 1)) * root_industrious_required__cpu)) + (bool2int((root_schizoid_runs__on = 1)) *
root_schizoid_required__cpu)) <= root_typical_cpu);
/* System */ constraint (((((bool2int((root_front__end_runs__on = 1)) * root_front__end_required__memory) + (bool2int((root_omniscient_runs__on = 1)) *
root_omniscient_required__memory)) + (bool2int((root_industrious_runs__on = 1)) * root_industrious_required__memory)) + (bool2int((root_schizoid_runs__on = 1)) *
root_schizoid_required__memory)) <= root_typical_memory);
/* System */ constraint (((((bool2int((root_front__end_runs__on = 3)) * root_front__end_required__cpu) + (bool2int((root_omniscient_runs__on = 3)) *
root_omniscient_required__cpu)) + (bool2int((root_industrious_runs__on = 3)) * root_industrious_required__cpu)) + (bool2int((root_schizoid_runs__on = 3)) *
root_schizoid_required__cpu)) <= root_chatter_cpu);
/* System */ constraint (((((bool2int((root_front__end_runs__on = 3)) * root_front__end_required__memory) + (bool2int((root_omniscient_runs__on = 3)) *
root_omniscient_required__memory)) + (bool2int((root_industrious_runs__on = 3)) * root_industrious_required__memory)) + (bool2int((root_schizoid_runs__on = 3)) *
root_schizoid_required__memory)) <= root_chatter_memory);
/* Typical */ constraint ((root_typical_cpu = 3) /\ (root_typical_memory = 2048));
/* Monster */ constraint ((root_monster_cpu = 12) /\ (root_monster_memory = 16384));
/* Chatter */ constraint ((root_chatter_cpu = 2) /\ (root_chatter_memory = 1024));
/* FrontEnd */ constraint ((root_front__end_required__cpu = 1) /\ (root_front__end_required__memory = 512));
/* Omniscient */ constraint ((root_omniscient_required__cpu = 6) /\ (root_omniscient_required__memory = 4096));
/* Industrious */ constraint ((root_industrious_required__cpu = 1) /\ (root_industrious_required__memory = 512));
/* Schizoid */ constraint ((root_schizoid_required__cpu = 2) /\ (root_schizoid_required__memory = 1024));

solve satisfy;
```

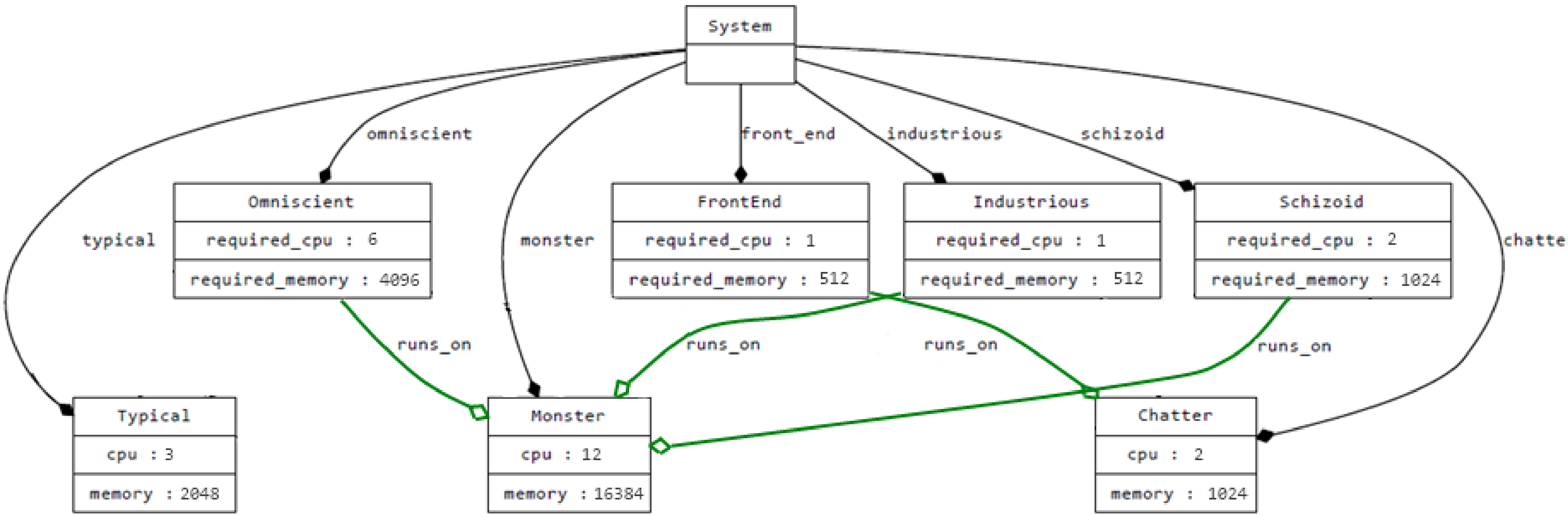
CSP Solution

- Used the *Gecode* CSP Solver, which supports:
  - Backtracking search
  - Local search
  - Optimisation functions
  - Decision heuristics
- Takes < 400ms (hard to benchmark tiny problems)
- *Lets show the solution visually…*

Example: Problem (Instances)



Example: Solution (Instances)



On-Going & Future Work

- Formally defined semantics for the configuration language, including:
  - Refinement Types (*e.g.* x:int where x > 4)
  - Optimisation Functions
  - Soft Constraints (Preferences)
- Minimum-change goal (for Re-Configuration)
- Usability
- Generate *Puppet* code using templates

Summary

- Cloud and IaaS configuration
  - Cloud = PaaS
  - Enterprise = IaaS
- State-of-the-art: Declarative languages
  - LCFG, Cfengine, Puppet
- Modelling an IaaS problem
  - New declarative language
- Solving with CSP
  - Using the *Gecode* solver
- Future work
  - Semantics, usability, advanced features