Taming Deployment with SmartFrog

Steve Loughran
Julio Guijarro
HP Laboratories, Bristol, UK

steve.loughran at hpl.hp.com
julio.guijarro at hpl.hp.com

O'REILLY EUROPEAN OPENSOURCE CONVENTION

17-20 OCT., 2005 • NH GRAND HOTEL KRASNAPOLSKY • AMSTERDAM, THE NETHERLANDS

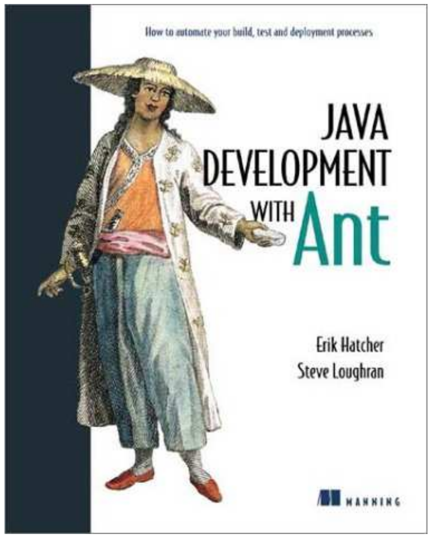
About Us

Steve Loughran

- Research scientist at HP Laboratories on *Grid-Scale Deployment*
- Apache Ant & Axis committer
- Co-author of *Java Development with Ant*
- Writing the 2nd “Ant1.7” edition;

Julio Guijarro

- Research scientist at HP Laboratories on *Grid-Scale Deployment*
- *Leads the SmartFrog open source effort*


How to automate your build, test and deployment processes
JAVA DEVELOPMENT WITH Ant
Erik Hatcher
Steve Loughran
MANNING

The goal of our HPLabs research

- How to host big applications across distributed resources
  - Automatically / Repeatably
  - Dynamically
  - Correctly
  - Securely
- How to manage them from installation to removal
- How to make grid fabrics useful for classic server-side apps

Deployment: why does it always go wrong?

Because

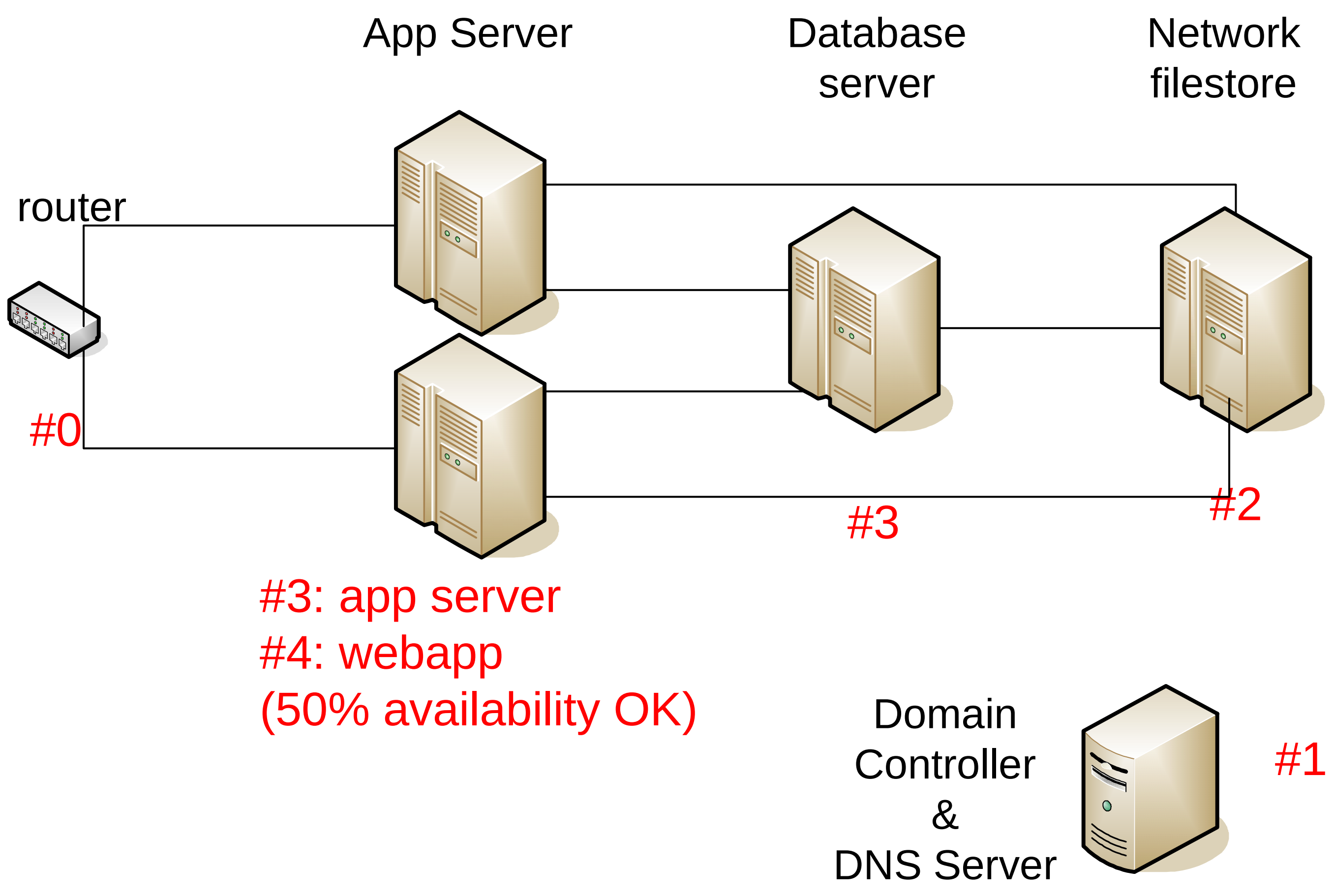
- it gets ignored
- configuration is half the problem
- nobody ever automates it
- the tools are inadequate
- it always goes wrong just before you go live

Deployment is unreliable, unrepeatable and doesn't scale

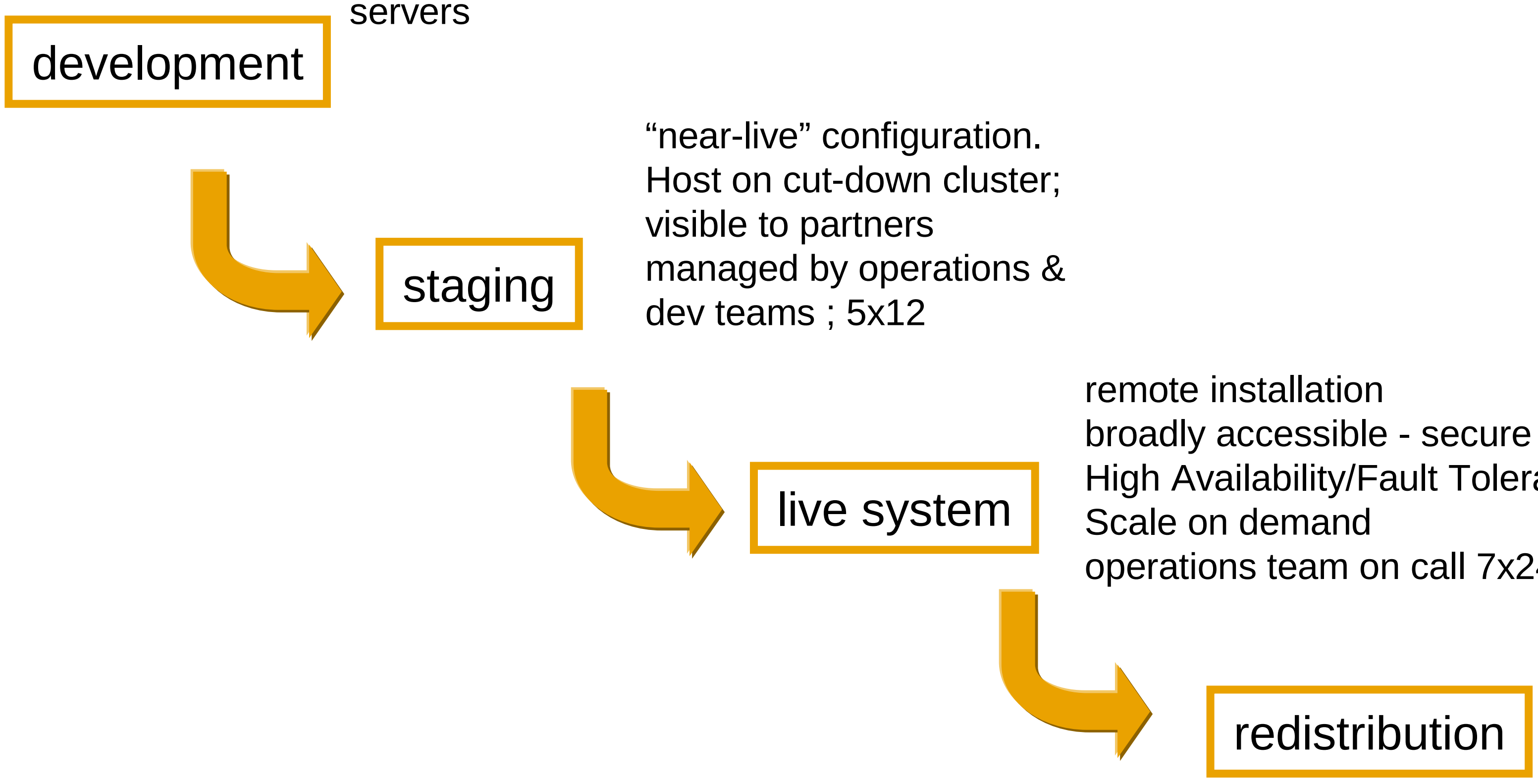
Configuration causes the problems

- It's the difference between configurations that hurt
- All those things that need to be consistent
  - configuration files
  - registry settings
  - router bindings
  - firewall
  - database
  - run-time values
- Trying to track down mismatches is hard

Choreography is "tricky"



Deployment through development



configure for diagnostics & testing
host on developers' boxes or local servers

"near-live" configuration.
Host on cut-down cluster;
visible to partners
managed by operations & dev teams ; 5x12

remote installation
broadly accessible - secure
High Availability/Fault Tolerant
Scale on demand
operations team on call 7x24

location/user specific configuration
e.g. IP addresses, passwords, ...
Self-diagnosis

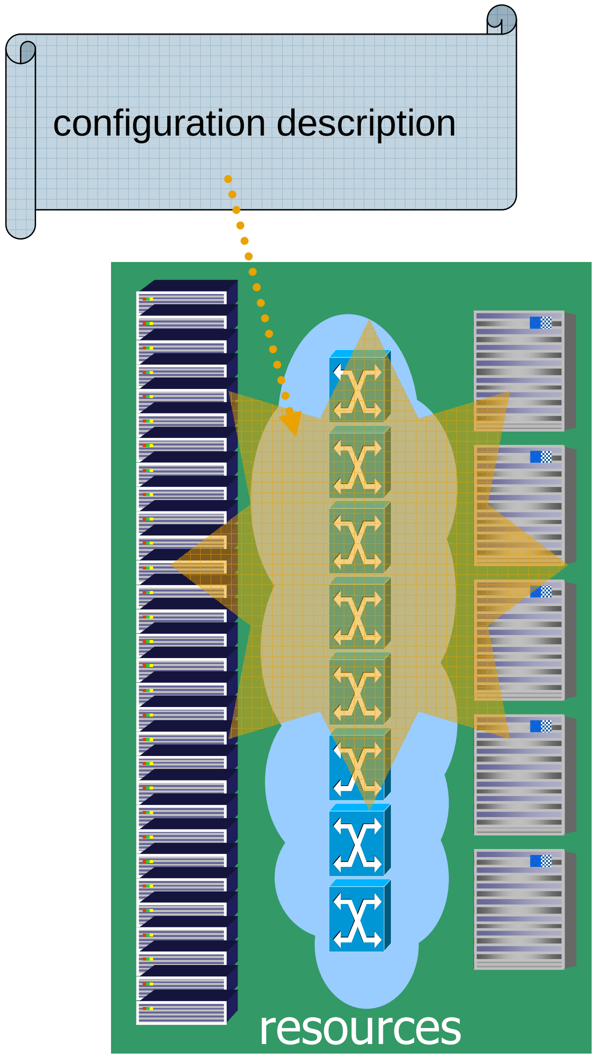
Configuration *is* deployment

Imagine a file that could declare the desired configuration state of a distributed system

- Define templates and extend them to describe different configurations
- Cross-referencing to eliminate duplication errors
- Composition for bigger systems

Create reality to match

- configure the declared items
- start/stop them
- adapt to failure or changing load



Imagine: SmartFrog

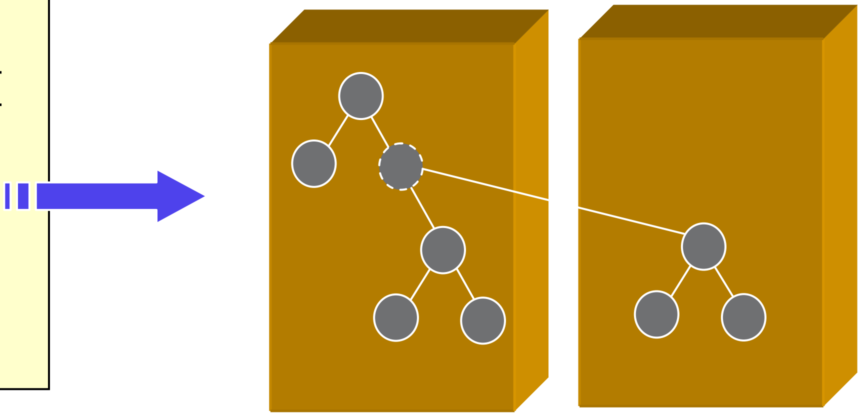
- Distributed Deployment System
- LGPL licensed
- Written in Java
- SourceForge hosted
- http://smartfrog.org/

SmartFrog
(Smart Framework for Object Groups)

A framework for describing, deploying and managing distributed service components.

- A description **language** for specifying configuration
- A **runtime** for realising the descriptions
- A **component model** for managing service lifecycle
- **Components** to deploy specific things

```
sfConfig extends WebService {

    WebServer extends LAZY Apache {
         port 8080;
    }

    AppServer extends Jboss;

}
```



SmartFrog Description Language

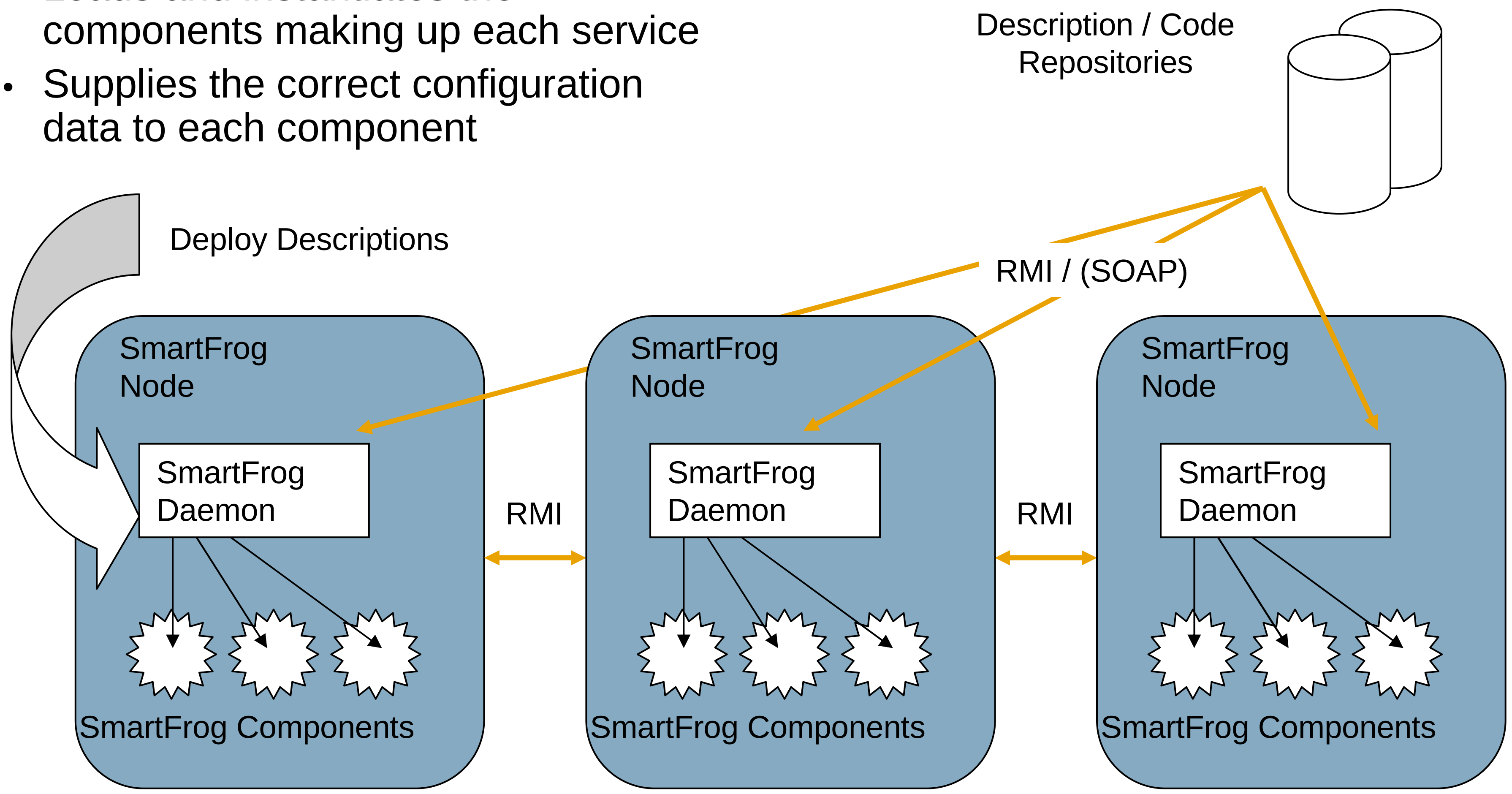
- A declarative, data description language
  - Describes the configuration of a system

- templates for deployment
  - Prototypes to fill in with real values
  - Extend, override, combine

- Service descriptions are interpreted by components hosted by the runtime
  - Semantics are not implemented in the language
  - Can accommodate wide range of services and models

SmartFrog Deployment Engine

- Distributed, decentralized, secure deployment engine
- Loads and instantiates the components making up each service
- Supplies the correct configuration data to each component



A complex template can cover everything

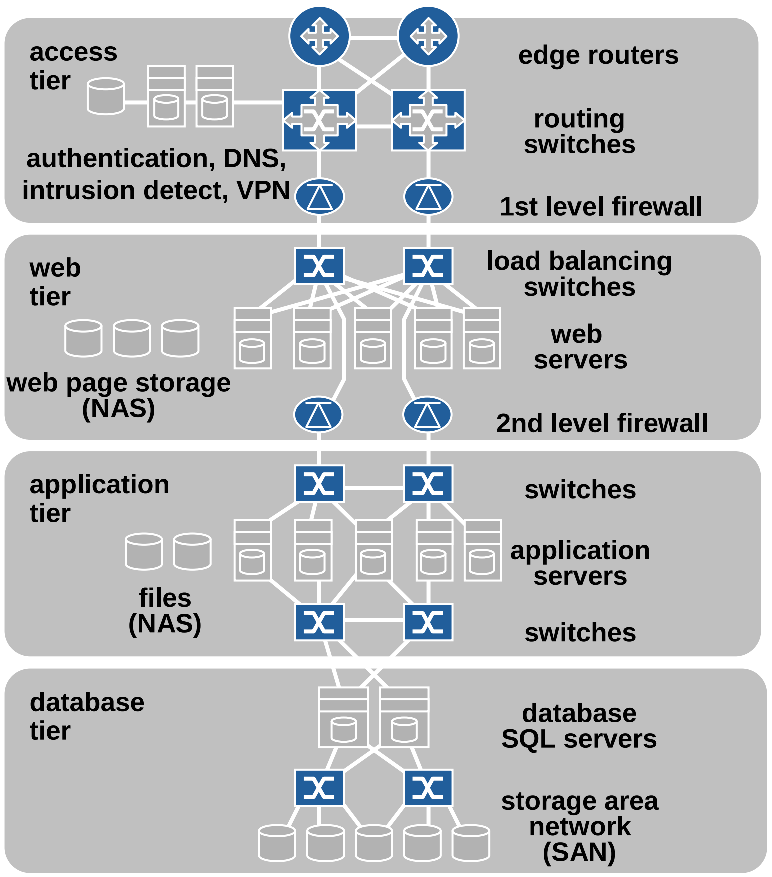
Template parameters

- transaction rate
- response times

- min/max no. of web servers

- min no. of app servers
- specific EJB's

- size of data,
- no. of tables



- constructed from templates for
  - web server
  - application server
  - …
- example of multiple domains
- (sub-)system templates require strong notion of validation
- collections of sub-templates are a common feature

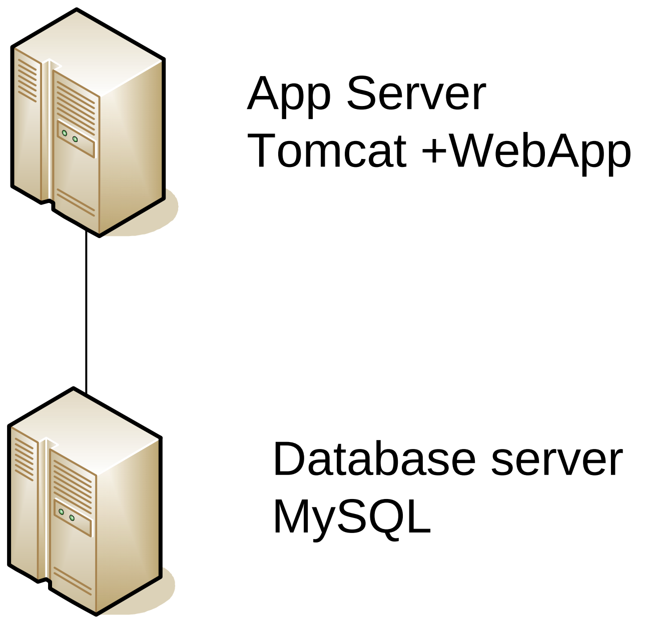
Goal: two tier app

MySQL database

Tomcat server

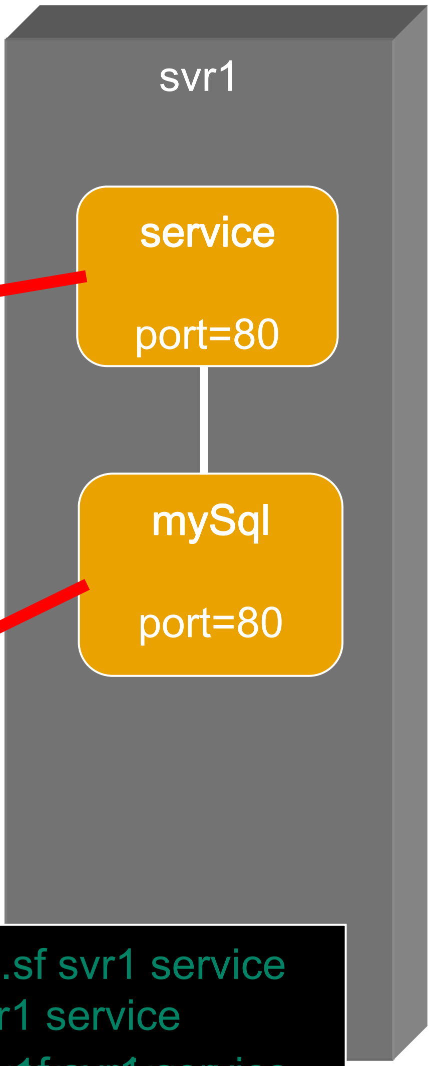
WAR application

Two hosts



MySQL

```
MySQLTemplate extends Prim {
  sfClass  "org.sf.mysql";
  port     TBD;
}

sfConfig extends Compound {
  port 80;


  mySql extends MySQLTemplate {
    sfProcessHost  "svr1";
    port           ATTRIB:port;
    db             "myDB";
    username       "user";
    password LAZY securePassW;
  }



}
```



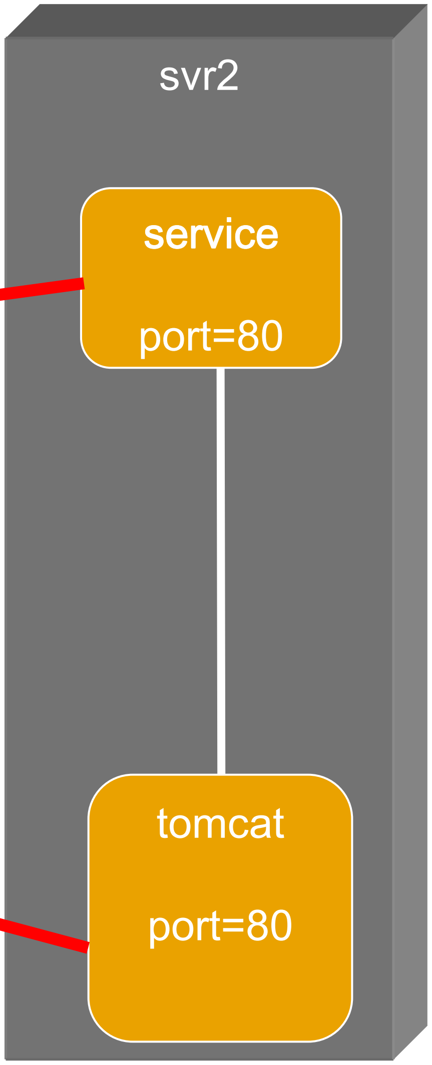
```
$ sfstart mySQL.sf svr1 service
$ sfterminate svr1 service
$ sfstart mySQL.sf svr1 service
```

MySQL

Demo

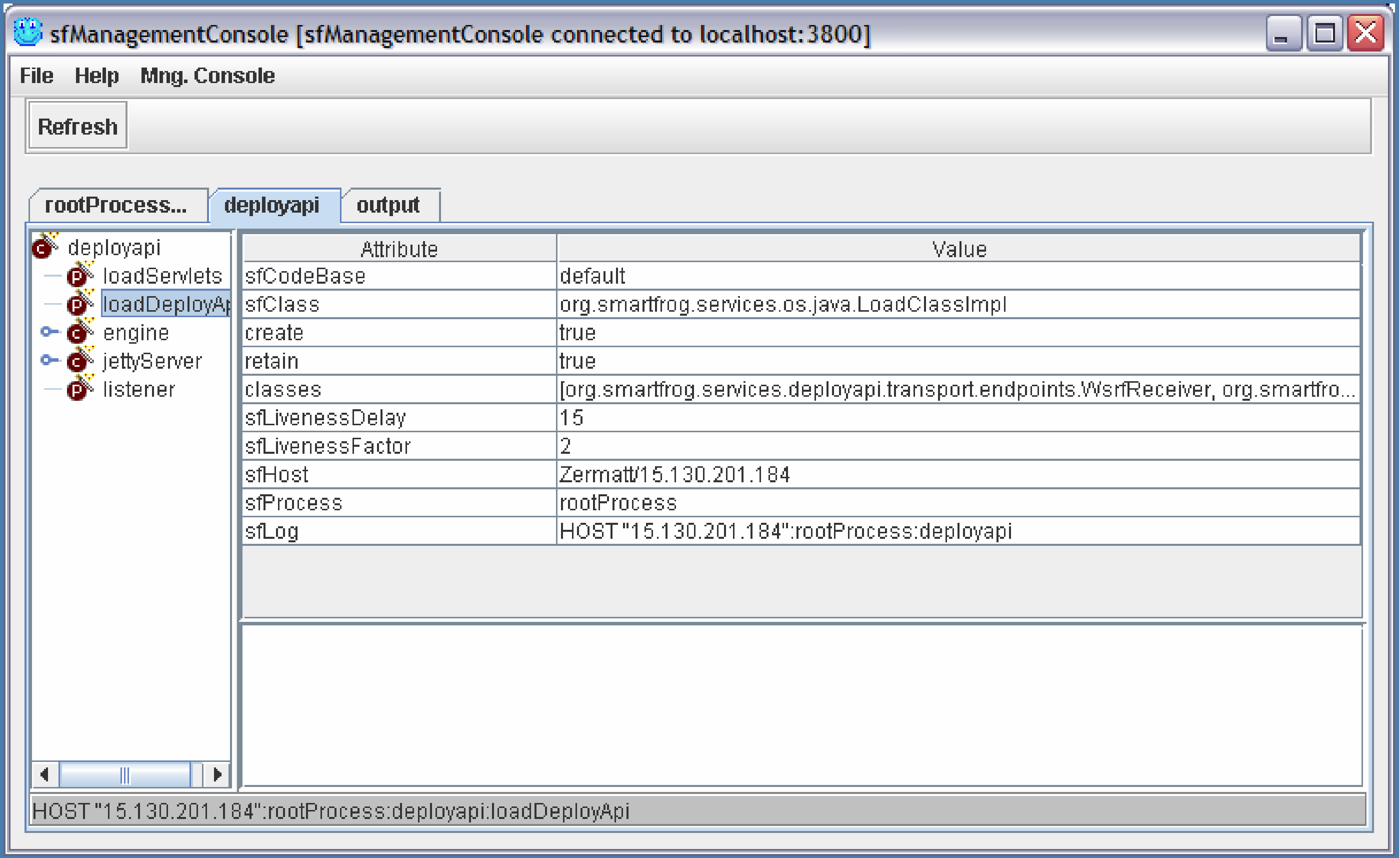
Tomcat

```
TomcatTemplate extends Prim {
  sfClass  "org.sf.tomcat";
  port     TBD;
  peer     TBD;
}

sfConfig extends Compound {

  port 80;




   tomcat extends TomcatTemplate {
     sfProcessHost  "svr2";
     port           ATTRIB:port;
     peer           LAZY svr1;
   }

}
```



```
$ sfstart tomcat.sf svr2 service
$ sfstart service.sf svr2 service
$ sfterminate svr2 service
$ sfstart tomcat.sf svr2 service
$ sfterminate svr2 service
```

Demo: Tomcat + Web Application

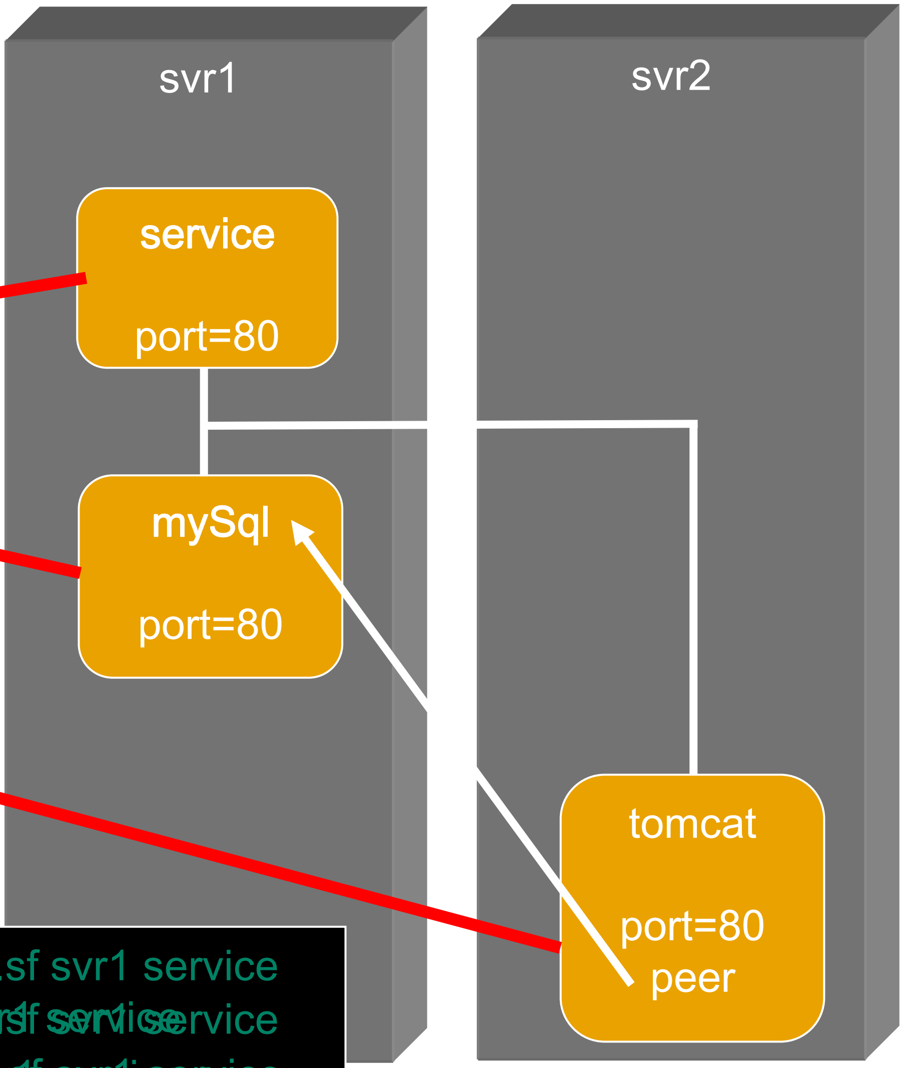


Attribute	Value
sfCodeBase	default
sfClass	org.smartfrog.services.os.java.LoadClassImpl
create	true
retain	true
classes	[org.smartfrog.services.deployapi.transport.endpoints.WsrfReceiver, org.smartfro...
sfLivenessDelay	15
sfLivenessFactor	2
sfHost	Zermatt/15.130.201.184
sfProcess	rootProcess
sfLog	HOST "15.130.201.184":rootProcess:deployapi

HOST "15.130.201.184":rootProcess:deployapi:loadDeployApi

Integration: Deploying a Service

```
Service extends Compound {
  sfClass  "org.sf.service";
  port     TBD;
}

sfConfig extends Service {

  port 80;

  mySql extends MySQLTemplate {
    sfProcessHost  "svr1";
    port           ATTRIB:port;
  }

  tomcat extends TomcatTemplate {
    sfProcessHost  "svr2";
    port           ATTRIB:port;
    peer           LAZY mySql;
  }

}
```



```
$ sfstart service.sf svr1 service
$ sfterminate svr1 service
$ sfstart service.sf svr1 service
```

Integration: Deploying everything

Demo

Components are like Ant tasks: they do the heavy lifting

	Ant	SmartFrog
Runtime	Ant	SmartFrog Daemon
Unit of execution	Project	System
Unit of work	Task	Component
Binding	`IntrospectionHelper`	`sfResolve()`
Lifespan	`execute()`	Lifecycle methods
Failure	Halt the build or ignore	Report to container/ping

Implementing a component

```
import com.hp.smartfrog.Prim.*;
import java.rmi.*;

public class Example extends PrimImpl implements Remote {
    private String hostname;

    public Example() throws RemoteException {
    }

    public void sfDeploy() throws Exception {
        super.sfDeploy();
        hostname=sfResolve("hostname","",true);
    }

    public void sfStart() throws Exception {
       super.sfStart();
       sfReplaceAttribute("Started",new java.util.Date());
    }

    public void sfTerminateWith(TerminationRecored tr) {
         /* any component specific termination code */
         super.sfTerminateWith(tr);
    }
}
```

extend base class
implement a Remote interface

lifecycle methods
called by the runtime

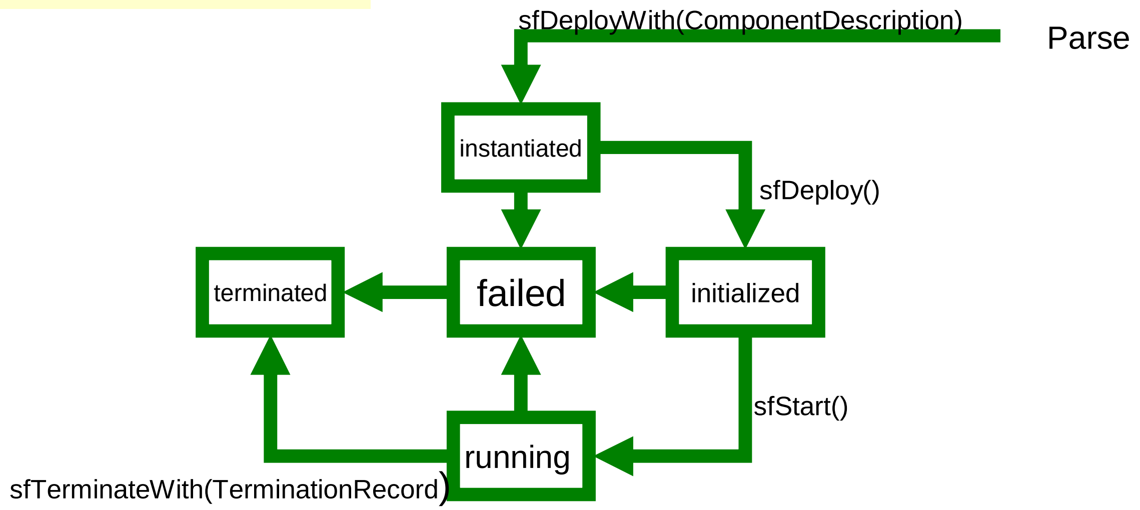
How to write a new one? Describing components

```
MyExample extends {
    sfClass "Example";
    hostname "localhost";
}


something extends MyExample {
    sfProcessHost "192.168.2.1";
    sfProcessName "subproc-2";
    hostname "laptop";
    timestamp LAZY:Started;
}
```

initial template

component location

other configuration data



Composition

Systems are composed of ***applications*** that are composed of ***components***

Applications: are deployed and managed as a group

Built in components that manage other components

- **shared lifecycle** (`Compound`): start and end components together
- **sequential**: when one component stops, the next starts, …
- **parallel**: start components together, but end separately
- **failure handling**: start one component if another fails

```
mySystem extends Compound {
    appServer extends JBoss {}
    database extends Oracle {}
    apps extends Compound { ... }
}
```

What ones do we have?

Filesystem	tempfiles, directories, text & XML files
Execution	shell scripts, Java, maven2 library download
Workflow	sequential, conditional, retry operations
Logging	remote forwarding/control of logs
Networking	telnet, scp, ftp, email
WWW:	HTTPD, jetty, tomcat, web page liveness check
SLP, *Anubis*	dynamic node discovery
JMX integration	configure JMX objects
JUnit	distributed unit testing

Where is SmartFrog being used?

SE3D: HP/Alias Film Rendering:

http://se3d.co.uk/

CERN Openlab

- Install, configure and uninstall a PBS/Torque cluster
- SmartFrog RPMs (it also installs SF as a service)
- http://openlab-mu-internal.web.cern.ch/

University UFCG, Brasil

- JBOSS http://www.lsd.ufcg.edu.br/~gustavo/smartfrog/jboss.tgz

PlanetLab: distributed application research

http://www.planet-lab.org/

Key points

- Deployment and configuration is a serious problem
- Large Scale Deployment is fun research
- With SmartFrog you can
  - describe deployments
  - instantiate them across a network
  - host components that form the application

Get involved!

- Download and play with the tool!
- Join the mailing list and send us any questions!
- Check out and build the code from CVS. Start with small projects, work up to big clusters...
- Look at http://se3d.co.uk/ to see what you can do with 500+ servers

For more information and downloads:

www.smartfrog.org

Questions?

invent

LGPL?

- Better that than inventing a new one.
- Apache stance is currently “you can depend on, but not redistribute LGPL libraries”
- So use it, don’t be scared. LGPL only means you must provide the source of any changes to SmartFrog or its bundled components, not any components/descriptors you write.

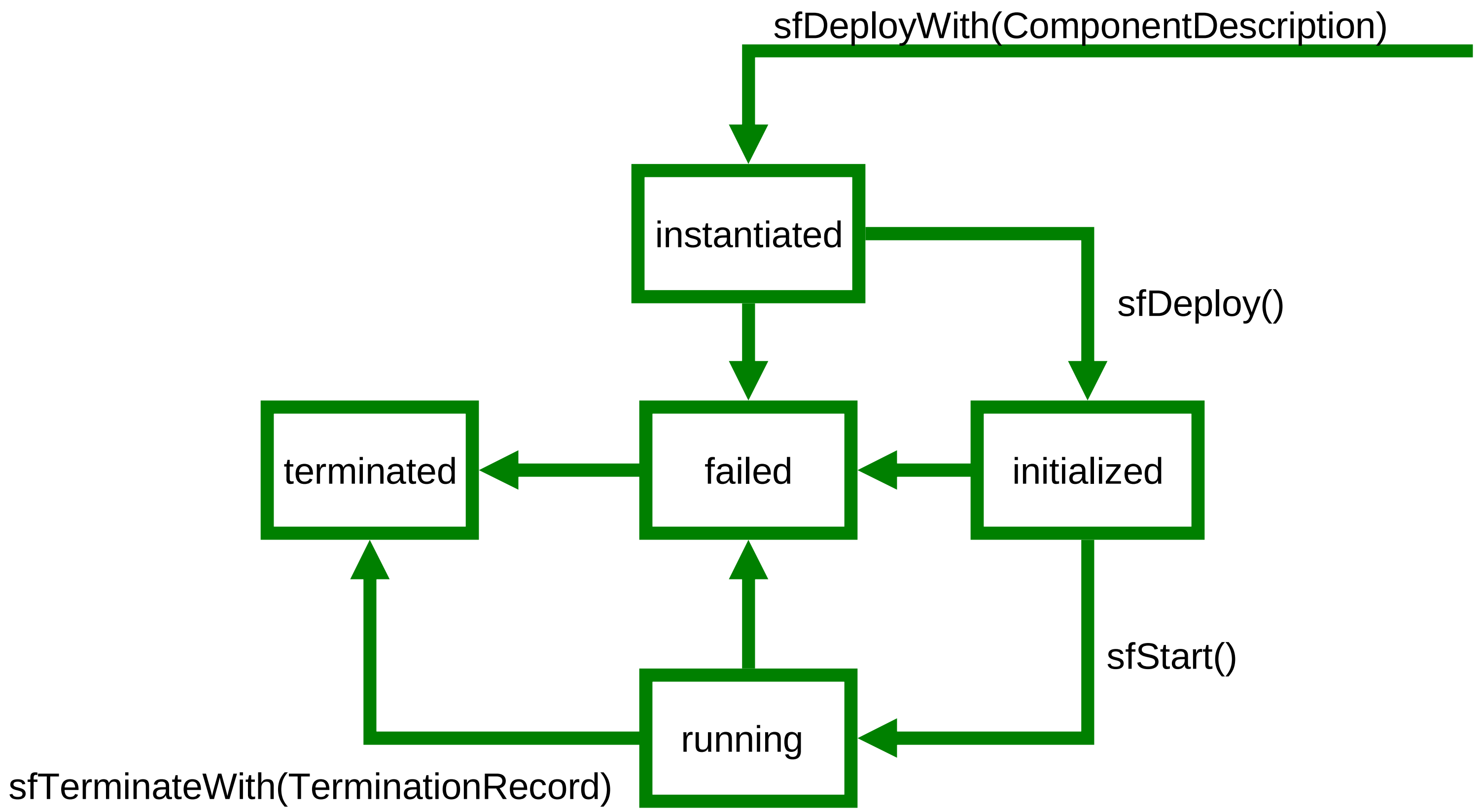
Security

- SmartFrog needs to protect against deployment or other management actions from rogue entities
- Cannot rely purely on SSH/user accounts/etc as SmartFrog has active communicating agents
- As SmartFrog downloads configuration descriptions and code, we need to protect against introduction of rogue code
- Communications over SSL
- Signed JARs to contain everything
- Private CA for each deployment.

Not XML?

- There is an XML derivative language being standardised at the Global Grid Foundation
- Join the CDDLM working group to get involved
  - https://forge.gridforum.org/projects/cddlm-wg
  - http://xml.coverpages.org/computingResourceManagement.html#cddlm
- We have found that an XML language is harder for humans to work with, but it has value in XML/XSL pipelines, e.g. Cocoon, inside Ant, XDoclet...
- XSD is particularly troublesome, as are bits of XPath
- Maybe RDF would be work better :)

The component lifecycle is that of a system


sfDeployWith(ComponentDescription)
instantiated
sfDeploy()
terminated
failed
initialized
sfStart()
running
sfTerminateWith(TerminationRecord)

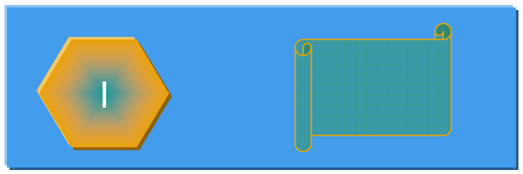
Components: Interpreters of Descriptions

- Each configuration domain is associated with a configuration interpreter, programmed to reify the configurations associated with that domain

- Each description from a domain is matched with one of these interpreters to reify the description

- The full semantics of a description is defined by

  interpreter + description



- The description is in effect a parameter to the interpreter defining the configuration state of the sub-system involved

- Can freely define new interpreters and new "languages" as required