




SmartFrog for grid deployment and configuration management.

Xavier Gréhant
HP fellow - openlab



“Grid technologies make it feasible to access large numbers of resources securely, reliably, and uniformly.

However, the coordinated management of these resources requires new **abstractions, mechanisms, and standards** for the **quasi-automated management of the ensemble.**”

Foster, Jennings, Kesselman. *Brain Meets Brawn.*

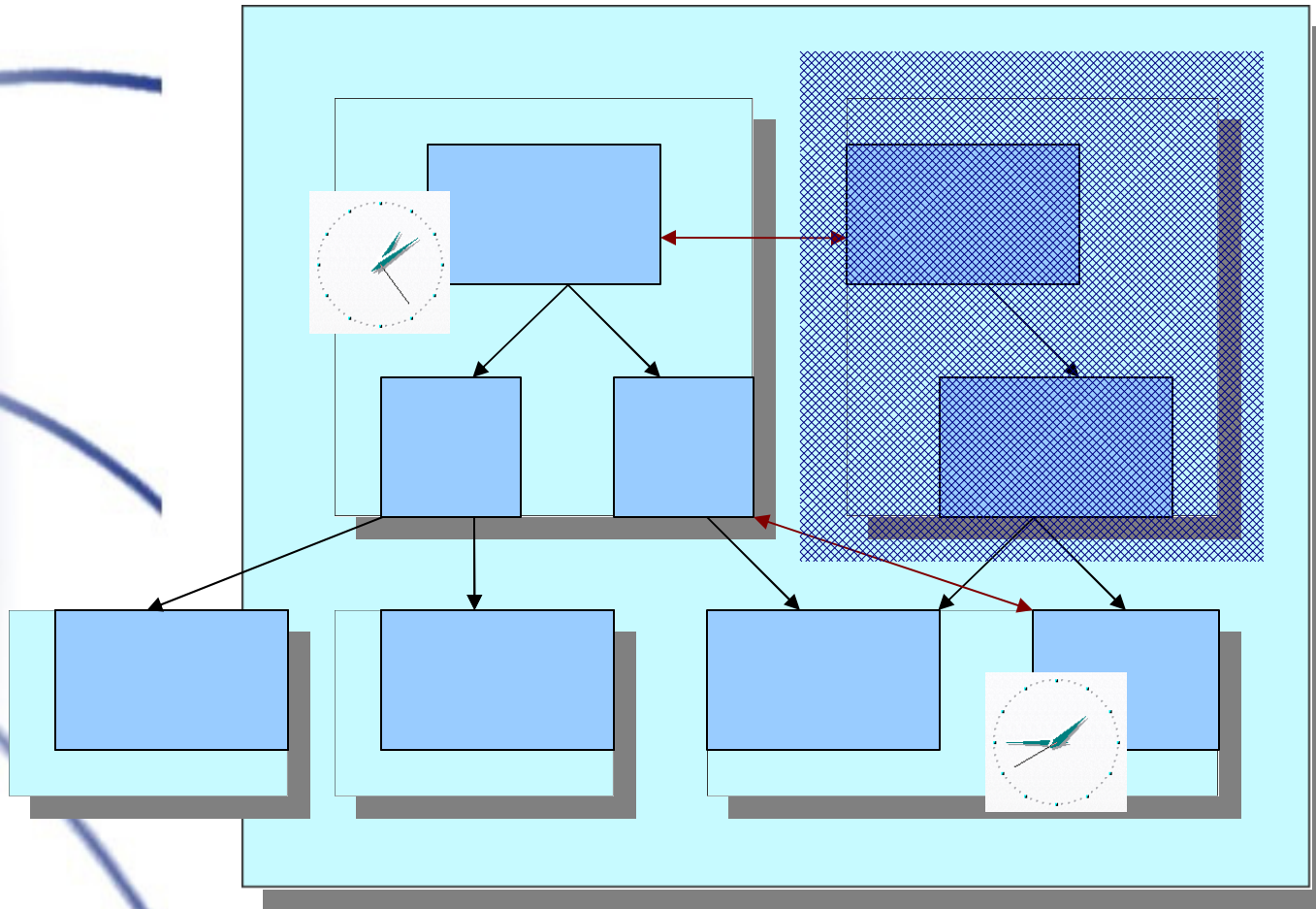
Contents

- Common patterns
- Two powerful frameworks
 - Fractal
 - SmartFrog
- Deployment & conf. mgt.
 - Different approaches
 - gLite deployment with SF

Contents

- **Common patterns**
- Two powerful frameworks
 - Fractal
 - SmartFrog
- Deployment & conf. mgt.
 - Different approaches
 - gLite deployment with SF

Common patterns: concepts



Common patterns: concepts

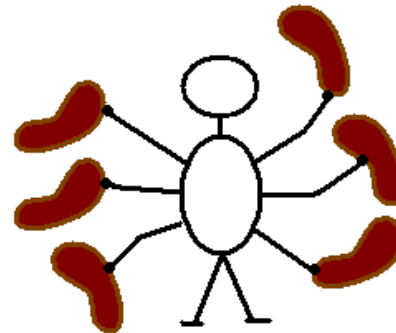
- a static conceptual structure
 - at different levels
 - objects hierarchies
 - components distributions/stacks
- + a dynamic (transverse) binding
 - to alleviate:
 - resources & code distribution
 - time locality
 - network unpredictability
 - environment changes/evolutions

Common patterns: concepts

- To be defined
 - structure
 - lookup
 - scope
 - metadata
- 2 opposite strategies
 - runtime-sets
 - component-gets

Common patterns: examples

- JMX, MBeans
 - introspection
 - static interface
 - dynamic interface: runtime exposure
 - agent: remote management
 - distributed service layer



Common patterns: examples

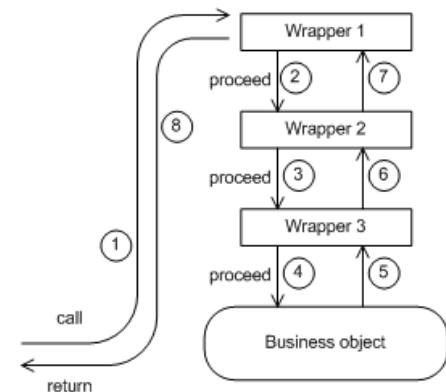
- OSGi
 - originally for embedded devices
 - *component = Bundle*
 - *jar file*
 - *manifest: metadata for framework*
 - *BundleActivator, BundleContext, Events and Listeners.*
- *Oscar: OSGi-compliant*
 - *define skeleton applications*
 - *invoke bundles at runtime*
 - *activate services and patches.*

Common patterns: examples

- Aspect-oriented programming
 - *Plain Old Java Object (POJOs)*
 - *main requirement*
 - *Aspects*
 - *cross-cutting concerns (persistence, distribution, transaction, fault-tolerance, logging...)*
 - *advice: plugin code*
 - *point-cut: regexp*
 - *Framework*
 - *wrapper*
 - *metadata: up to the implementation*

Common patterns: examples

- *JAC (Java Aspect Components)*
 - *Compile-time*
 - *poor OO*
 - *Run-time*
 - *introspection/reflection*
 - *Run-Time Type Introspection (set/query)*
 - *Thread local attributes*
 - *Wrappers*
 - *host*
 - *class*
 - *method endpoints.*



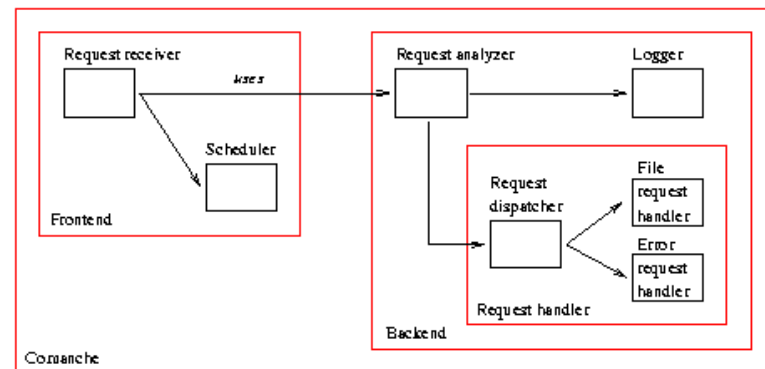
from jac.objectweb.com

Contents

- Common patterns
- **Two powerful frameworks**
 - Fractal
 - SmartFrog
- Deployment & conf. mgt.
 - Different approaches
 - gLite deployment with SF

Fractal

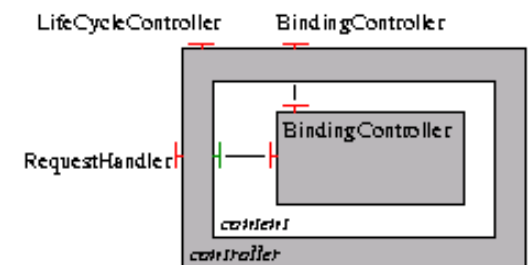
- The model
 - separation of concerns
 - separation of interface and implementation
 - component oriented programming
 - inversion of control
 - recursive components identification
 - contracts definitions between components



from fractal.objectweb.com

Fractal

- Component description
 - java API
 - ADL (architecture description language)
 - GUI
- Management tools
 - controler methods
 - introspection methods



SmartFrog

- Management fundamentals
 - java code to define the scope:
 - in the component or inherited
 - structure for transverse binding:
 - component attribution
 - late binding provided by *LAZY*

```
#include "org/smartfrog/components.sf"

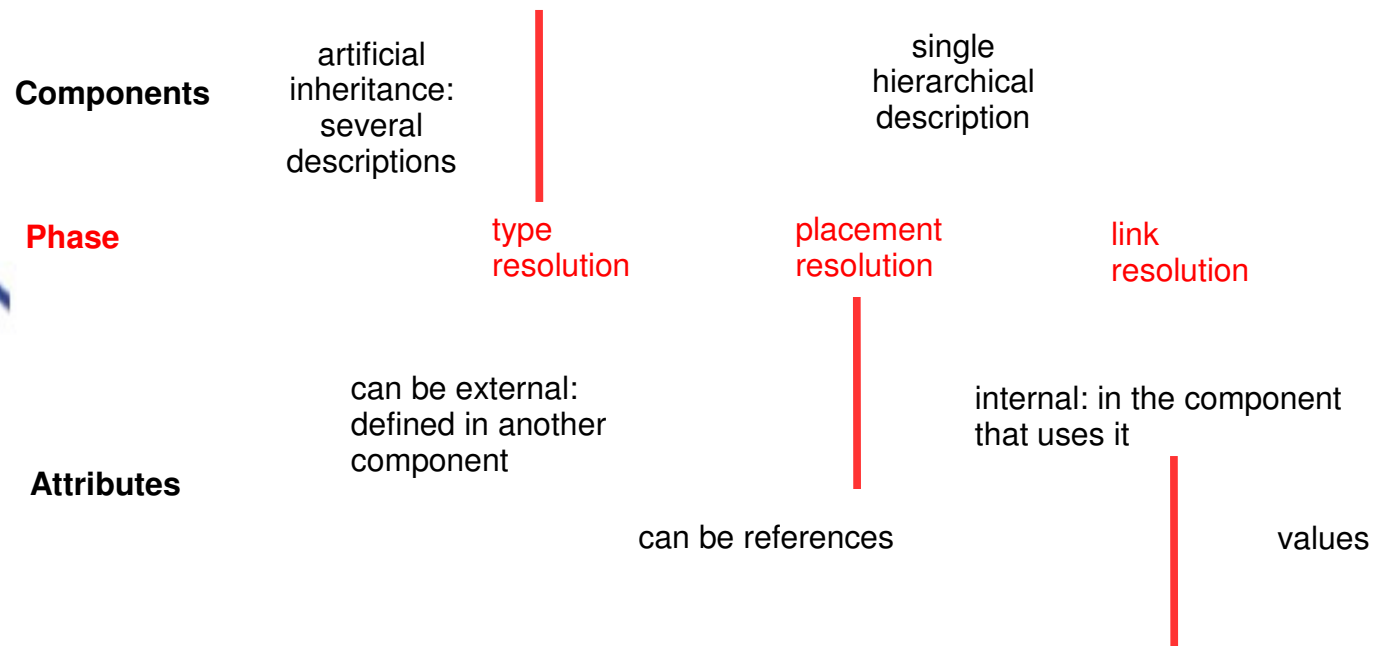
MyPrim extends Prim {
  sfClass "com.hp.myexamples.MyPrim";
  debug true;
  retryCount 10;
  databaseRef LAZY ATTRIB DB;
}
```

SmartFrog

- Components representation
 - Two component hierarchies
 - extension (flattened at parse time) to inherit attributes handling
 - attribution: a component configures and manages another.
 - Extreme tunability/flexibility
 - parser level (phases)
 - component description level
 - assertions up to user (schemas)
 - java code
 - standard/TBD general methods

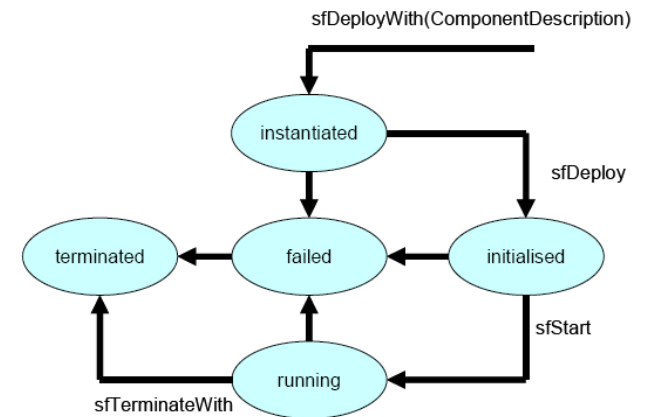
SmartFrog

- Parser



SmartFrog

- Java code
 - User adds interfaces for lookup
 - defining context variables
= component attributes
 - User overrides methods
 - for lifecycle management
 - Framework provides API
 - context reflection
 - useful components



from SmartFrog tutorial

Contents

- Common patterns
- Two powerful frameworks
 - Fractal
 - SmartFrog
- **Deployment & conf. mgt.**
 - Different approaches
 - gLite deployment with SF

Different approaches

- local machine
 - Automake / Autoconf
 - Ant
- Linux package managers
 - RPMs, APT, YUM
- parallel commands
- nothing common to unify distributed installation, configuration, management.

gLite deployment with SF

- SF components
 - to handle gLite installation/configuration methods
 - yet leveraging only SF ubiquity
- Status and issues:
 - debugging stage
 - large component granularity
 - security (worm container)
 - sfDaemons deployment
 - security/(c) procedures


gLite deployment with SF

- Expectations
 - link to client interface
 - fault tolerance
 - autonomy?

“New components integrate as effortlessly as a new cell establishes itself in the human body.

These ideas are not science fiction, but elements of the grand challenge to create self-managing computing systems.”

Jeffrey O. Kephart, David M.Chess, *The Vision of Autonomic Computing.*



Thank you!
Questions?