



LSR-Adèle

A Conceptual Comparison of Avalon, Fractal and Gravity

Humberto Cervantes
Richard S. Hall



Comparison

- Fractal

- “The Fractal composition **framework** is both a conceptual and a software framework that supports component-based programming -- including components definition, configuration, composition and administration -- according to the Fractal component model.”

- Avalon

- “Avalon is a **framework** that allows components of varying scale to be created, managed via a specific set of lifecycle methods, and used in an application. While Avalon is geared towards server-side applications, it is not limited to such, and is quite flexible.”

- Gravity

- “A **framework** that supports composing applications out of components that exhibit dynamic availability.”

- Comparison along two main axes

- Framework / Implementation



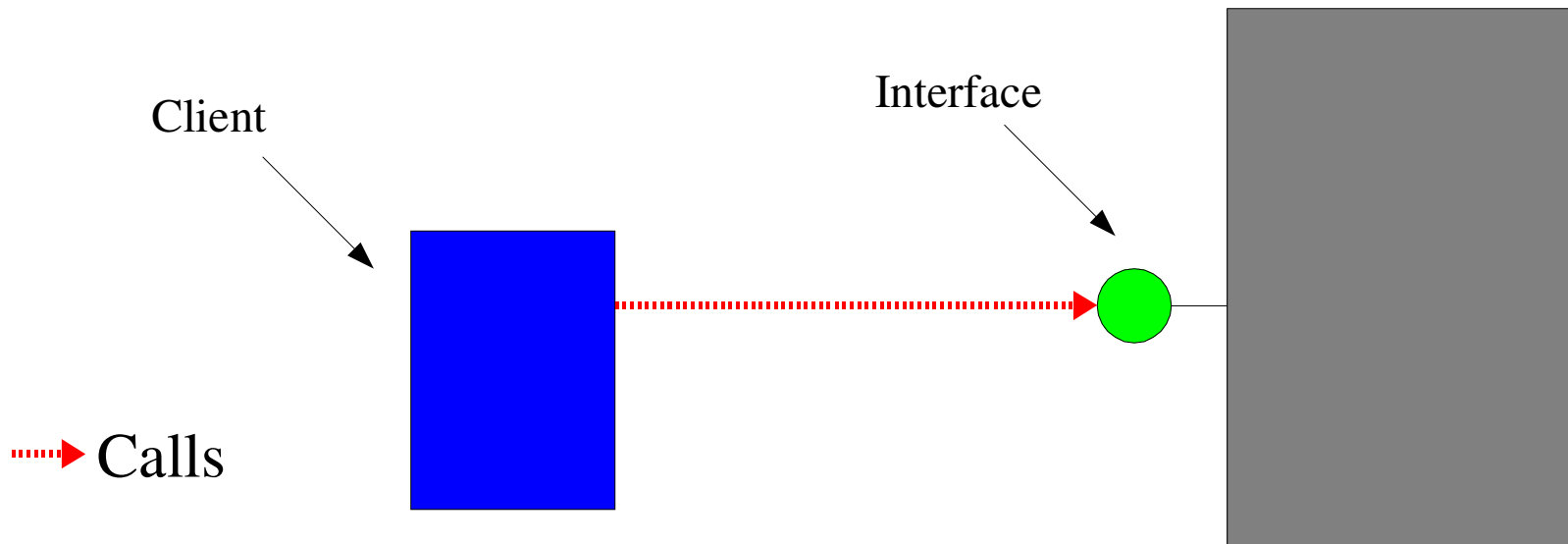
Framework

- Collection of abstract classes
 - Component model
 - Components
 - Management entities (controllers,...)
- Imposes a set of design patterns
 - Separation of interface and implementation
 - Factory
 - Inversion of control
 - Interceptor
 - Service registry



Interfaces and Implementations

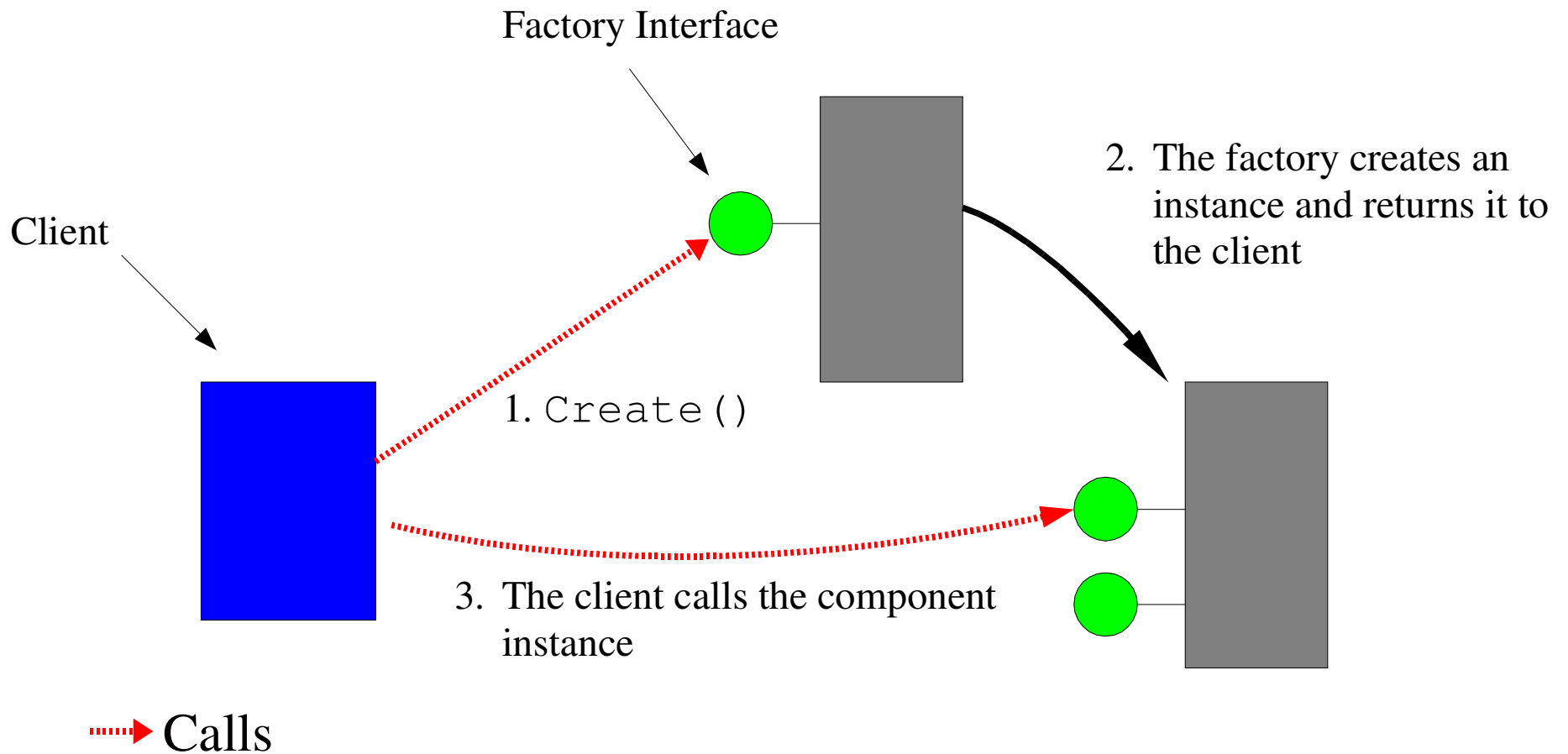
- Clients communicate with component instances only through well defined interfaces
- Different implementations can implement the same interface





Factory

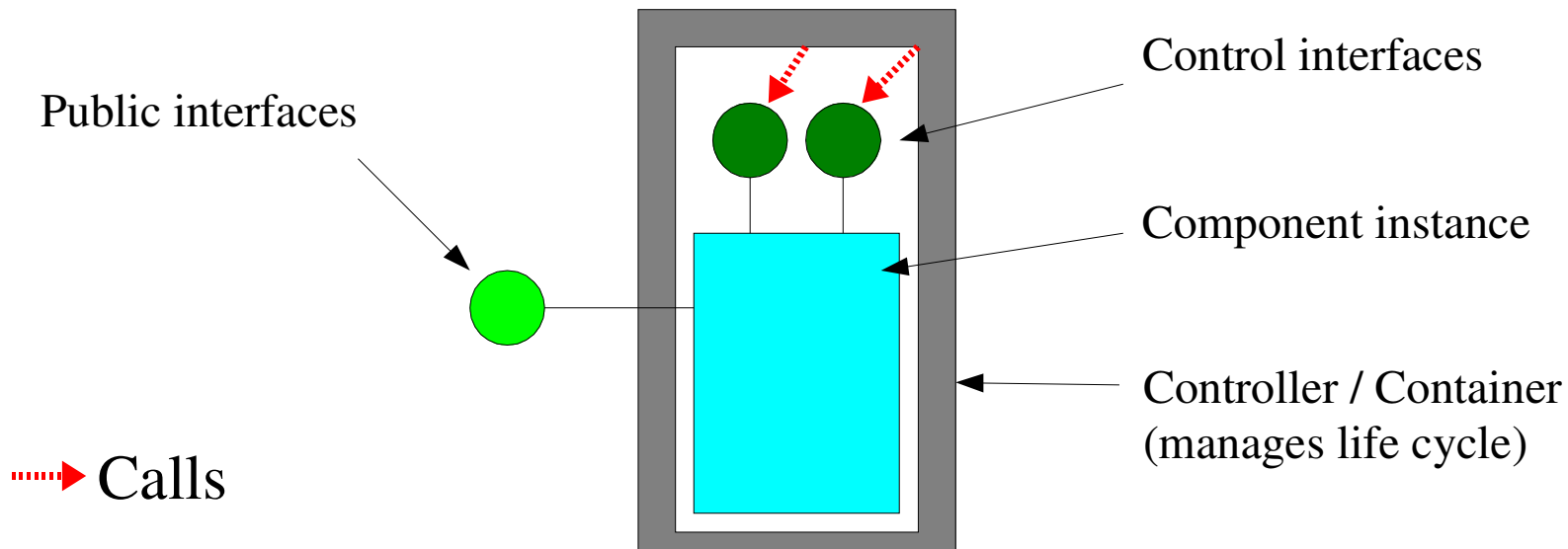
- Provide and indirect instantiation mechanism





Inversion of Control (IOC)

- A component instance is always externally managed through a set of control interfaces that provide life cycle methods

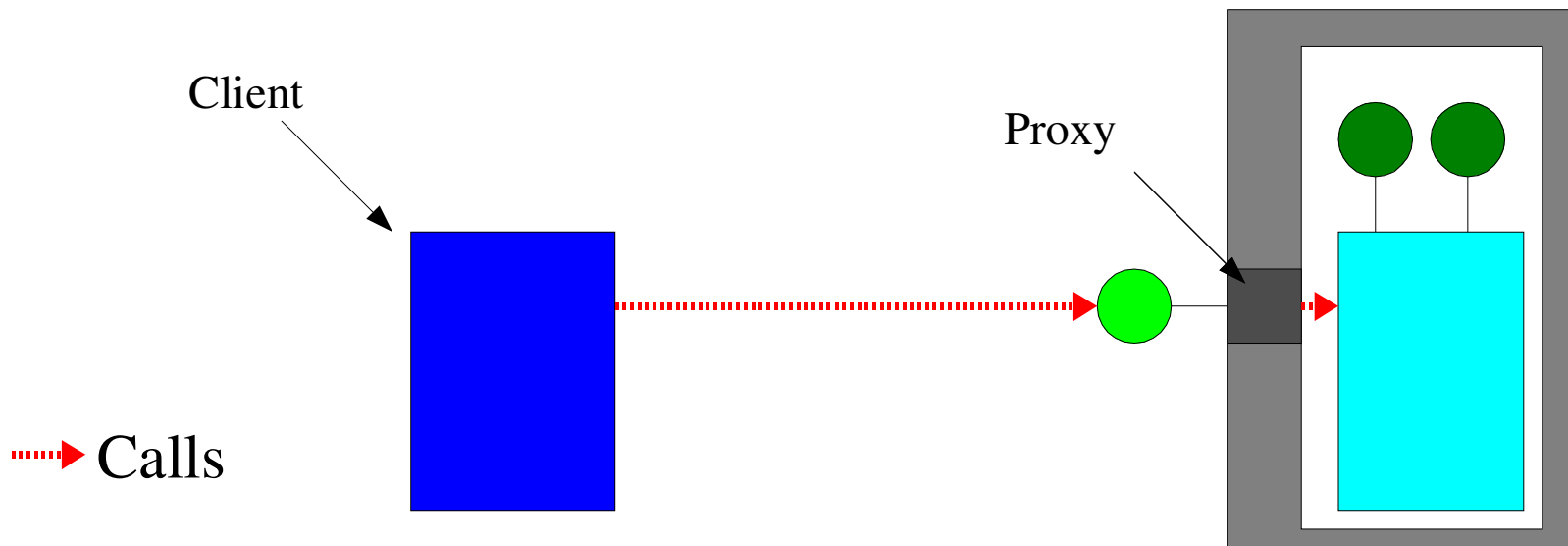


- Control interfaces can be discovered at run time



Interceptor (Proxy)

- Clients do not communicate directly with the component instance, calls are received by a proxy

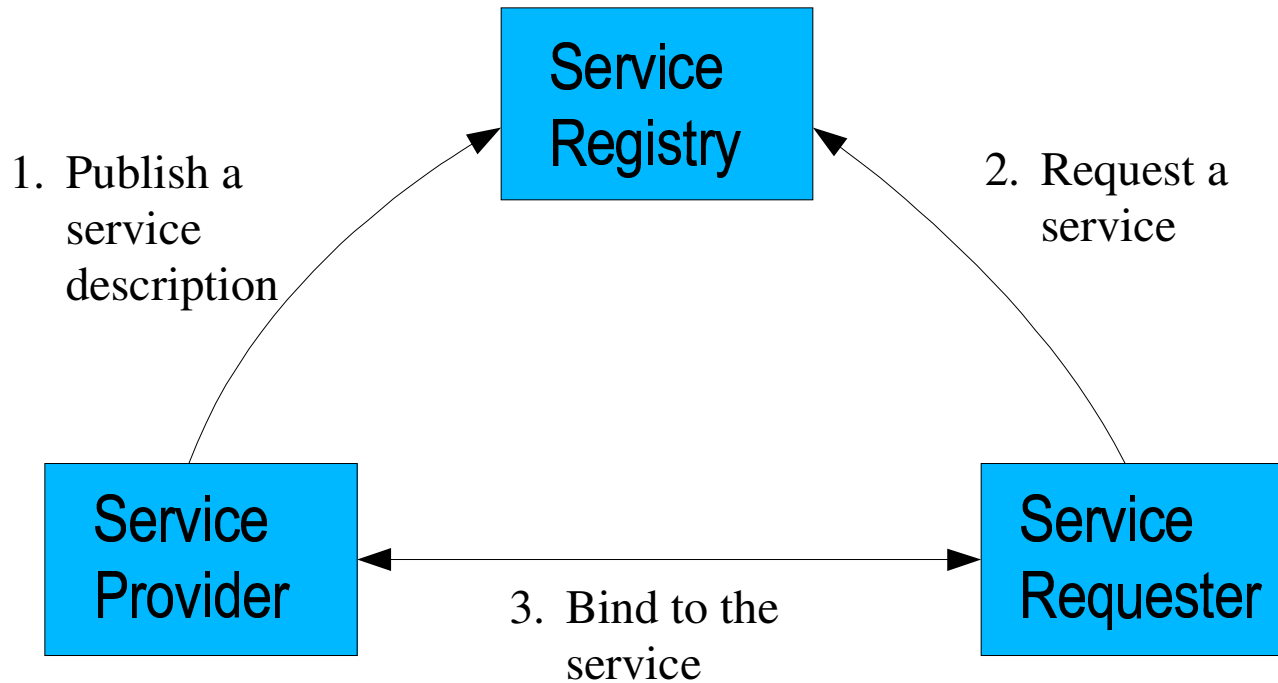


- The interceptor may insert, block or replace the component's functionalities



Service Registry

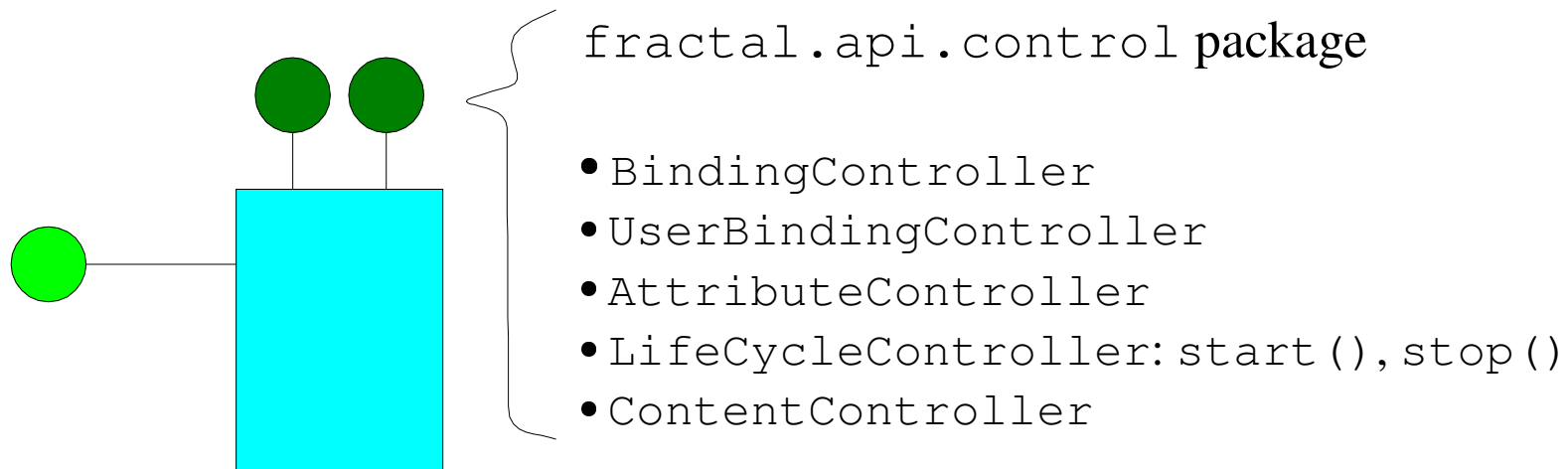
- Services are well defined interfaces





Design Patterns in Fractal

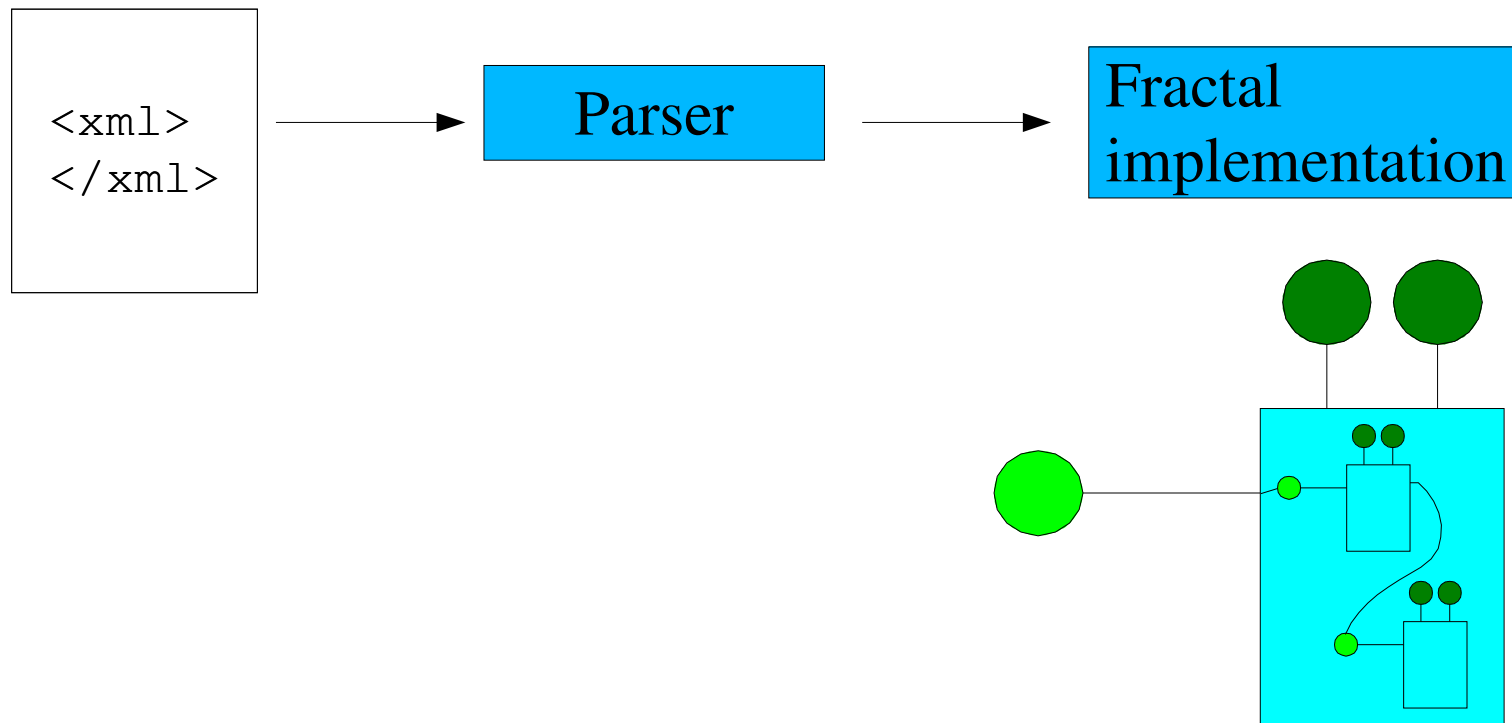
- Separation of interface and implementation
- Factory
- Interceptor
- Inversion of control





Composition in Fractal

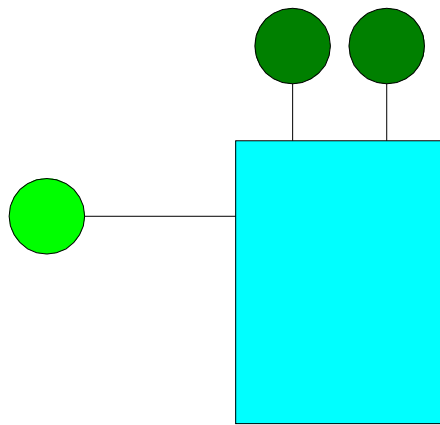
- Fractal provides an API to define and instantiate templates and their components
- Hierarchical composition is supported





Design Patterns in Avalon

- Separation of interface and implementation
- Service registry
- Inversion of control



- **LogEnabled:** `enableLogging(Logger)`
- **Contextualizable:** `contextualize(Context)`
- **Serviceable:** `service(ServiceManager)`
- **Configurable:** `configure(Configuration)`
- **Parametrizable:** `parametrize(Parameters)`
- **Initializable:** `initialize()`
- **Startable:** `start()`, `stop()`

- **Suspendable:** `suspend()`, `resume()`
- **Recontextualizable:** `recontextualize()`
- **Reconfigurable:** `reconfigure()`
- **Reparametrizable:** `reparametrize()`

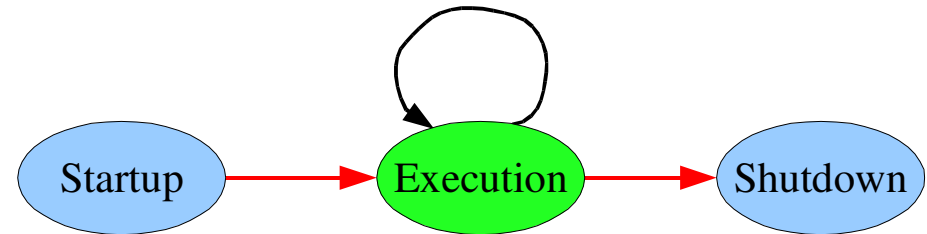
- **Disposable:** `dispose()`



Control Method-Call Sequence

Startup Phase:

1. `constructor` (as a consequence of instantiation)
2. `contextualize`
3. `service`
4. `configure`
5. `parameterize`
6. `initialize`
7. `start`



At various or repeated times after startup:

1. `suspend`
2. `recontextualize`
3. `recompose`
4. `reconfigure`
5. `reparameterize`
6. `resume`

Shutdown Phase:

1. `stop`
2. `dispose`
3. `finalize` (at some indeterminate moment by the garbage collector)

Legend:

Called only once during life cycle

Called at least once but possibly more

Optionally called



Instance Creation in Avalon

- The container instantiates the components and register its roles with a `ServiceManager`
- The same role can be registered multiple times through a *selector*
 - The client selects the service by passing a key that identifies a service
- Different kinds of “lifestyles”
 - Single threaded: only serves 1 client
 - Thread safe: serves multiple clients
 - Poolable: serve 1 client, reusable



Implementation of Fractal

- Julia is France Telecom's implementation of the Fractal specification
- Julia code organized in 3 parts
 - Core Runtime Classes: this part contains the minimum classes that are needed at runtime
 - Code Generator Classes: this part contains the classes that are used to generate application specific classes that are needed by Julia
 - Tools: this part contains classes that are not needed at runtime



Controller Generation

- Controllers and interceptors are generated by following a *mixin* technique
- Julia is configured through files with a Lisp syntax
 - These configuration files allow for the addition of interceptors at different levels in the component hierarchy



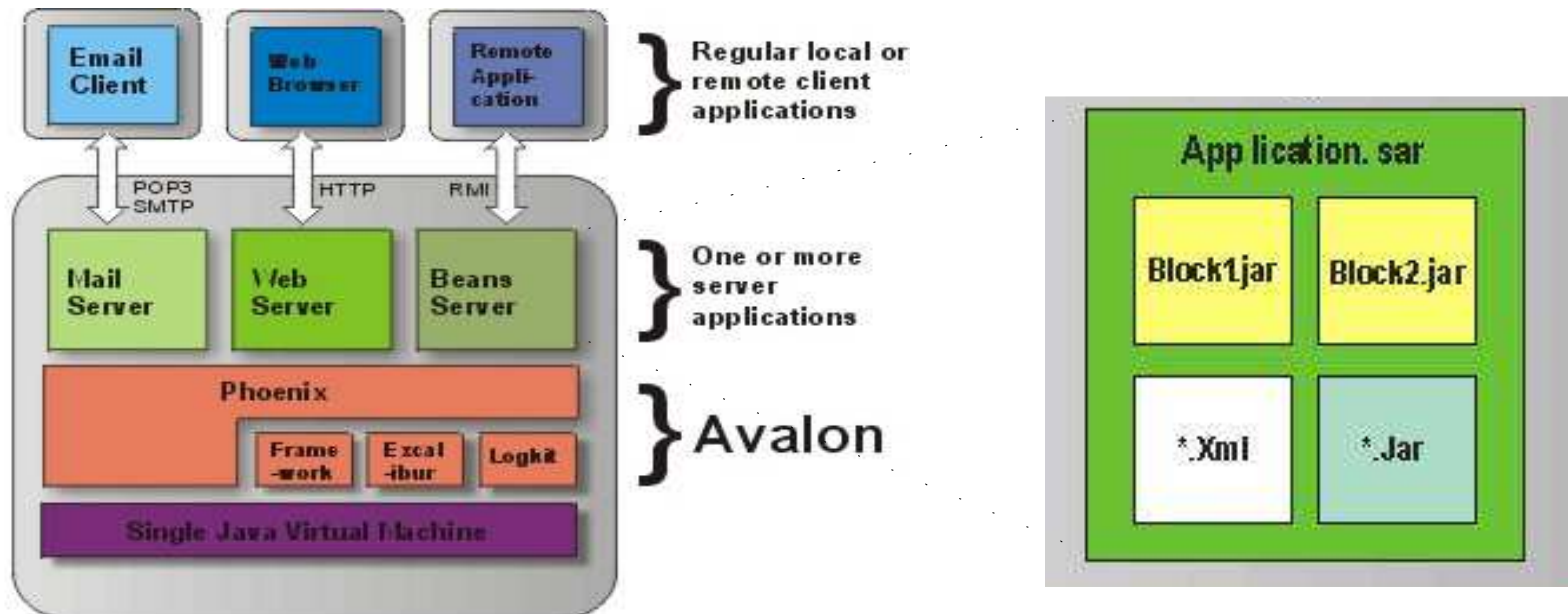
Implementation in Avalon

- Several subprojects
 - Framework: defines the component model
 - Excalibur: implements framework
 - Phoenix: micro-kernel to manage apps
 - Cornerstone: component repository
 - LogKit: logging toolkit
- Excalibur offers different types of containers
 - Fortress: extensible life cycle
 - Merlin: hierarchical container, automated assembly, candidate resolution
 - Tweety: learning purposes



Phoenix

- “Phoenix is a micro-kernel designed and implemented on top of the Avalon framework. It provides a number of facilities to manage the environment of server applications. Such facilities include log management, class loading, thread management and security.”





Conclusions

- Fractal and Avalon have similar goals
 - Infrastructure for “bigger” projects
 - Apache HTTP server, ...
 - OpenCCM, JOnAS, ...
- Fractal is a hierarchical component model
 - API to define types, compositions...
- Avalon is a service-oriented component model
 - Server-side orientation



Conclusions (2)

Avalon	Fractal	Gravity
Framework	Framework	Framework
Excalibur	Julia	Gravity
Phoenix	?	OSGi

- OSGi could be used as an equivalent to Phoenix in Fractal
- OSGi has successfully been used as an infrastructure in Gravity