The Fractal Project
Status & Perspectives

Jean-Bernard Stefani -- INRIA

Outline
- An “exo-kernel” view of middleware architecture
- Fractal: model
- Fractal: support
- Ongoing work around Fractal
- Perspectives

Exo-kernel middleware
- A view of middleware architecture
  - exo-kernel philosophy
    - no fixed set of infrastructure services
    - applications should be allowed to choose their supporting abstractions
  - 4 strata towards a middleware exo-kernel
    - not software layers
    - middleware as extensible and configurable component library
  - underlies ObjectWeb vision of middleware

Exo-kernel middleware
- Stratum #1: Components
  - lightweight, reflective component model
  - common set of tools for static and dynamic code generation and adaptation
  - basis for on-line system adaptation and evolution
- Stratum #2: Architectural Frameworks
  - pattern-based architectural frameworks for hard recurring issues
  - naming, types and meta-data
  - communications
  - monitoring and failure detection
  - resource management
  - distributed configuration management
Exo-kernel middleware

- Stratum #3: System services
  - P2P indexing and routing
  - Asynchronous communication services
  - Transactions & Orchestration
  - Configuration and Resource Management
  - High-availability support
  - Persistency support
  - Distributed queries
  - etc

- Stratum #4: Personalities
  - J2EE
  - Web services
  - OGSI
  - CORBA
  - etc

Fractal: model

- A component model
  - for building dynamically reconfigurable distributed systems
  - programming language-independent
  - reflective: components have their own meta-level
  - open: no fixed meta-level, no fixed connector types
  - with lightweight implementations (C, C++, Java)
  - comes with an extensible Architecture Description Language (ADL)

- Used in particular for building distributed systems infrastructures
  - operating systems
  - middleware (application servers, grids, etc)

Fractal: classical concepts

- Components are encapsulated data & behavior
  - runtime entities, not only design time or load time

- Interfaces are the access points to components
  - aka ports
  - interfaces emit and receive operation invocations

- Bindings mediate interactions between components
  - aka connectors
  - can be primitive (in the same address space) or composite
  - composite bindings are components + primitive bindings.
  - Bindings may span address spaces and networks
  - No fixed semantics for bindings

Fractal: original concepts

- A component comprises a membrane and a content
  - A membrane is made of controllers.
    - It can export control interfaces for some of these controllers.
    - The membrane exercises an arbitrary control over its content.
    - Components can export arbitrary details of their implementation.
    - No fixed set of controllers for component introspection & intercession
  - A content is made of other components.
  - A component has state.

- Components can be shared by multiple enclosing components.
  - Shared components: crucial for modeling software architectures with resources and cross-cutting aspects
Fractal: useful controllers

- Minimal introspection:
  - Component interface
  - Interface interface
    - cf COM, IUnknown
- Component introspection (I)
  - Content controller
    - to add/remove sub-components
  - Attribute controller
    - to set/get component attributes

Fractal: useful controllers

- Component introspection (II)
  - Binding controller
    - to set up/remove communication paths to/from component
    - a “binding” between components:
      - a component
      - can have arbitrary communication semantics
    - connecting components via a binding involves:
      - creating a binding (component)
      - using binding controllers on components to bind to set up ‘primitive bindings’
        (e.g. language references) with binding (component)
  - Lifecycle controller
    - to start/stop a component
Fractal: additional elements

- Instantiation
  - Factories
    - esp. binding factories
  - Templates: "homomorphic" factories
  - Bootstrap: "well-known" generic factory
- Packaging
  - software packages (e.g. Jpackages or OSGI bundles) as components
- Simple type system
  - Interface
  - Component

Fractal: forms of components

- Components without introspection: Objects
  - Java object (POJO)
- Component with minimal introspection & simple aggregation
  - COM component
- Component with binding controller and (fixed) lifecycle controller
  - OSGI bundle
- Composite with transaction, persistency & content controllers for POJOs: EJB container
  - POJOs with lifecycle interface: EJB2
  - POJOs: EJB3

Fractal: support

- Several Fractal implementations
  - Java: Julia (FTR&D - INRIA Sardes)
  - Java + AspectJ: AOKeLL (FTR&D - INRIA Jacquard)
  - C: Think (FTR&D - INRIA Sardes)
  - C++: (INRIA Sardes)
  - Smalltalk: (ENSM Douai)

Supporting the Fractal model

- General component structure
  - membrane = set of controllers
  - content = set of components
- No pre-determined control => support must facilitate the definition of membranes
  - library of controllers
    - default ones from Fractal specification
    - interceptors
  - ability to define new controllers
    - e.g. using mixins (Julia), AspectJ advices (AOKeLL)
Supporting the Fractal model: Julia

- Supporting Fractal in Java
  - primitive components defined by Java classes
  - primitive bindings are Java references
  - controllers are Java objects
  - interceptors are Java objects
  - membranes are lists of controllers and interceptors
  - controller (mixin) classes can be combined at load-time using a byte-code generator

Julia: component structure

- merge of control objects
- merge of control objects & interceptors
- merge of control objects, interceptors, & content
- list of control objects
- list of interface references
- control object
- interceptor
- interface reference

Ongoing work around Fractal

- Using Fractal
  - building infrastructure software (OS, middleware)
    - INRIA (Sardes, Oasis, Jacquard), IMAG-LSR, FTR&D, STMicroelectronics
  - building frameworks for adaptive applications (multimedia, mobile, context-aware)
    - INRIA (Sardes, Obasco), ENSM Douai, Nokia
Ongoing work around Fractal

- Extending & refining Fractal technology
  - formal basis
    - INRIA Sardes
  - component behavior
    - U. Prague, INRIA (Sardes)
  - formal verification
    - INRIA (Oasis, Vasy)
  - combining Fractal & AOP
    - INRIA (Jacquard, Sardes), FTR&D, U. Prague
  - ADL support & other tools
    - INRIA (Jacquard, Obasco, Sardes)

Fractal: Sample uses

- Operating system kernels
  - Think (FTR&D, STMicroelectronics & INRIA Sardes)
- Asynchronous middleware & communication subsystems
  - DREAM (INRIA Sardes)
- Transaction management
  - GOTM, Jironde (LIFL-INRIA Jacquard, INRIA Sardes)
- Persistency services
  - JORM, Speedo, Perseus (FTR&D, IMAG-LSR)
- Software architecture for Grid applications
  - Proactive (INRIA Oasis)
- Self-adaptive structures
  - (EMN-INRIA Obasco, Nokia)
- Distributed systems management
  - Jade (INRIA Sardes)

Fractal: perspectives

- Fractal v3
  - mid 2006
  - refined specifications (sharing, controller library, language mappings)
  - multiple mature implementations (Julia v3, AOKell v1, Think v3)
- Developing Fractal technology
  - extensible & retargettable ADL compiler
  - formal semantics for full Fractal
  - strongly typed, dynamic ADL
  - verification and validation tools
  - additional language mappings and implementations
  - model refinement: failures, transactions, aspects

Fractal perspectives

- Using Fractal in ObjectWeb projects
  - Software deployment & configuration management
  - Autonomic system management
  - Asynchronous middleware & Enterprise Service Bus (JBI implementation)
  - Jonas 5
    - at least service deployment and configuration
References to Fractal-related work