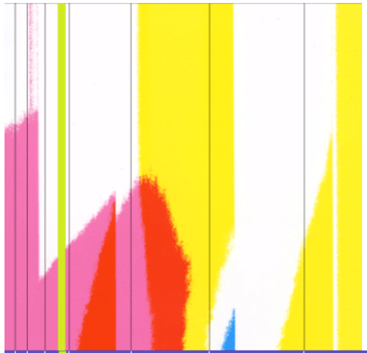


Client-Side Caching in Fractal RMI

Tuesday, November 29, 2005

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

INRIA

ObjectWeb Fractal Workshop – Grenoble, France

Philippe Merle

Email: *Philippe.Merle@inria.fr*
INRIA Futurs – Jacquard Project,
Laboratoire d'Informatique Fondamentale de Lille, France

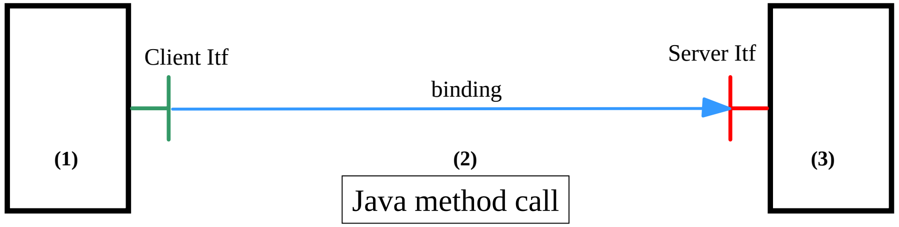
Agenda

- Motivation
- From Fractal to client-side caching in Fractal RMI
- Design issues and choices
- Examples
- Implementation status
- Evaluation
- Current limitations
- Conclusion
- Perspectives

Motivation

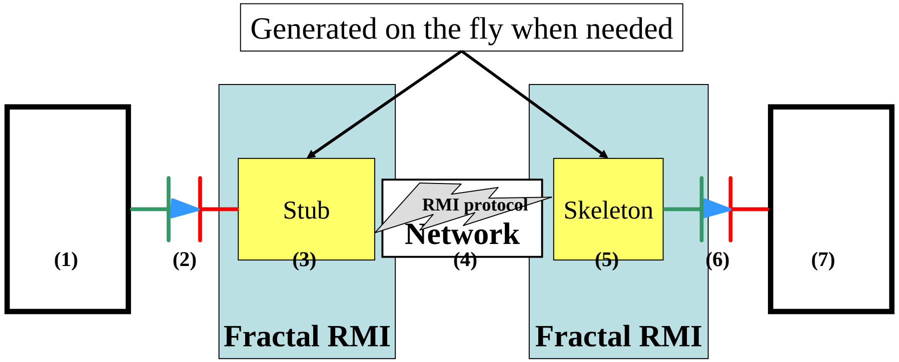
- Fractal for distributed fine-grain component-based middleware
  - Implementations: Julia, ProActive, Think, AOKell, etc.
  - Tools: Fractal ADL, Fractal Explorer, Fractal JMX, Fractal RMI, etc.
  - Middleware: DREAM, GoTM, Speedo, etc.
- Poor performance when distributed bindings between components!
  - Time(remote call) >> Time(local call)
- Our goal: Improving performance of distributed Fractal applications
- Well-known approaches to improve performance
  - Mobility: Move activities near used components (see ProActive)
  - Caching: Move data near using components
- Our approach: Client-Side Caching in Fractal RMI

Fractal



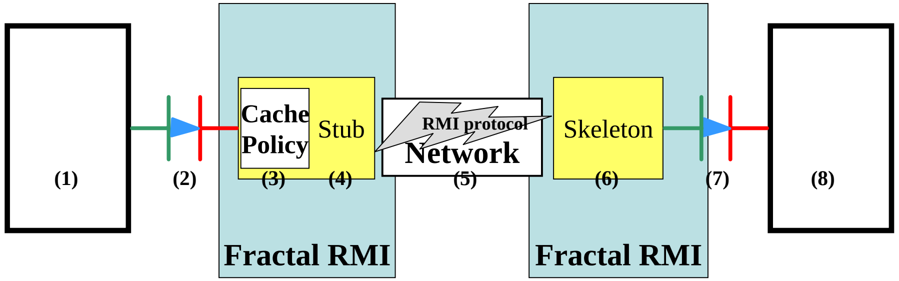
Same memory space

Fractal RMI



Two separate memory spaces

Client-Side Caching in Fractal RMI



Two separate memory spaces

Some Design Issues

- Which consistency policies?
  - None, local, or global
- Which caching granularity?
  - Operations, interfaces, components, composites, etc.
- Which level of transparency?
  - None, component participation, or full
- What kind of caching policies?
  - System or user defined
- How express caching policies?
  - Programmed as Java classes?
  - Described with Aspect Specific Language?
- How integrating client-side caching in Fractal RMI?
- Do we need to extend Fractal?

Our Current Design Choices

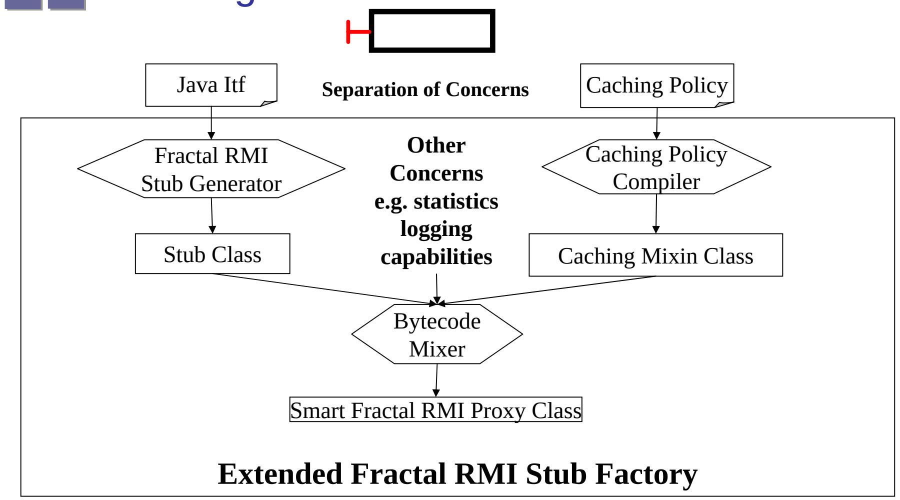
- Which consistency policies?
  - None, local, or global
- Which caching granularity?
  - Operations, interfaces, components, composites, etc.
- Which level of transparency?
  - None, component participation, or full
  - **Existing components directly reusable!**
- What kind of caching policies?
  - System or user defined
- How express caching policies? BOTH
  - Programmed as Java classes
  - Described with Aspect Specific Language
- How integrating client-side caching in Fractal RMI?
  - Caching as an aspect weaved into Fractal RMI
- Do we need to extend Fractal?

  NO

Client-Side Caching in Fractal RMI

- An Aspect Specific Language to abstract caching policies
  - At operation, interface, inter-interface levels
- A set of caching policies for
  - All Fractal controller interfaces
  - Fractal RMI Registry interface
  - Specific Julia controller interfaces
- Caching policies are compiled to caching mixins
- Caching mixins are mixed with Fractal stubs
  - → Smart proxies with fine grain cache
- Fractal Stub Factory is updated to use the bytecode mixer
- Still Work In-Progress!

The Big Picture



Java Itf

Separation of Concerns

Caching Policy

Fractal RMI Stub Generator

Other Concerns e.g. statistics logging capabilities

Caching Policy Compiler

Stub Class

Caching Mixin Class

Bytecode Mixer

Smart Fractal RMI Proxy Class

Extended Fractal RMI Stub Factory

A Simple Example: The *NameController* Interface

```
interface NameController
{
  public String  getFcName();



  public void setFcName(String name);



}
```

Caching Policy for the *NameController* Interface

```
interface NameController
{
  public String  getFcName();
          If already cached then return it
          Else delegate to stub
          Keep result in cache
  public void setFcName(String name);
          If cached value == name then return // OPTIMISATION
          Else delegate to stub
          Update cache
}
```

Caching Mixin for the *NameController* Interface

```
public class NameController_CachingMixin
implements CachingMixin,
                NameController
{
  // Reference to the delegate stub
  private NameController _stub_;

  // Cache for FcName
  protected StringHolder cachedFcName_;

   …

}
```

Caching Mixin for the *NameController* Interface

```
public String  getFcName()  {
  // Check if the result is already cached.
  if(cachedFcName_ != null) return cachedFcName_.value;
  // Is not already cached invoke remote controller.
  String result = _stub_.getFcName();
  // Update the cache.
  cachedFcName_ = new StringHolder(result);
  return result;
}
public void setFcName(String name)   {
  if(cachedFcName_ != null && name.equals(cacheFcName_.value)) return;
  // Invoke the remote NameController via the Fractal RMI stub.
  _super_.setFcName(name);
  // Keeps the name in the local cache.
  cachedFcName_ = new StringHolder(name);
}
```

Overview of Other Caching Policies

From	Interface	Operation	Policy
Fractal	**Interface**	getFcItfOwner, getFcItfName, getFcItfType, isFcInternalItf	Result cached
	Component	getFcType getFcInterface, getFcInterfaces	Result cached + Init cache of returned stubs
	Generic Factory	newFcInstance	Init cache type of returned component
	Content Controller	getFcInternalInterfaces getFcInternalInterface getFcSubComponents addFcSubComponent removeFcSubComponent	Result cached + init cache of returned stubs Update cache
	Binding Controller	listFc, lookupFc bindFc, unbindFc	Result cached Update cache
Julia	**LifeCycle Coordinator**	getFcState, startFc, stopFc, setFcStarted	Result cached Update cache
	SuperControllerNotifier	getFcSuperComponents addedToFc, removedFromFc	Result cached Update cache
Fractal RMI	**Naming Service**	list, lookup bind, rebind, unbind	Result cached Update cache

Main Issue with Fractal Specification

- No pre defined formal behavior specification for Fractal controller interfaces
  - To allow various implementations for various application contexts
- However, caching policies are based on observable behaviors of controller interfaces
- Examples
  - Are sub-components stopped when the super component is stopped?
  - Is a null name authorized?
    - Some components have a NameController which returns null value
- Proposal for Fractal V3:
  - Continue to define standard Fractal controller interfaces
  - But also define some standard possible behaviors
  - `Controller` and `ControllerBehavior1`, ... `ControllerBehaviorN`

Implementation Status

- Caching mixins already available for
  - All Fractal controller interfaces
  - Fractal RMI Registry interface
  - Some specific Julia controller interfaces
- Mixer of caching mixins and Fractal RMI stubs
  - Written with ASM 2.1
  - Based on the Julia controller mixer
- New Fractal Stub Factory using the bytecode mixer
- Added Statistics as another concern
  - Useful for evaluating method calls / methods cached

Evaluation

- Done on Fractal Explorer and Fractal ADL
- No modification of these Fractal applications!
- All remote Fractal introspection calls are cached!
  - Fractal Explorer: Drastically improve performance
  - Fractal ADL: Between 30%-50% of remote calls removed
    Only keep strict necessary remote calls

Current Limitations

- No consistency between distributed caches!
- Caching ASL must be defined!
- ASL compiler must be written!
- Mixer does not support inheritance between caching mixins
  - e.g., CachingMixin(IB) extends CachingMixin(IA) when IB inherits IA
- From a prototype to a stable release

Conclusion and perspectives

- Improving performance of distributed Fractal applications
- Client-Side Caching in Fractal RMI
  - ASL for abstracting caching policies
  - Generate (write) caching mixins
  - Mixing caching and stub concerns transparently and efficiently
- No modification of existing Fractal applications
  - Effective separation of concerns
- Perspectives
  - Resolve the current limitations
  - Generalize the approach to Java RMI, CORBA, Web Services
    - ASL, caching policies/mixins, and mixer
    - Specific caching policies

Thank you for your attention…

If you have any questions?