An API for Autonomous and Client-Side Service Substitution

H. Tchinda\textsuperscript{1,2}, J. Ponge\textsuperscript{2}, Y. Dan\textsuperscript{2,3}, and N. Stouls\textsuperscript{2}

\textsuperscript{1}UMMISCO, LIRIMA, Université de Yaoundé I, BP 812 Yaoundé Cameroun
\textsuperscript{2}Université de Lyon, INRIA, INSA-Lyon, CITI, F-69621, France – Email: first.second@insa-lyon.fr
\textsuperscript{3}College of Computer Science Chongqing University, Chongqing, China
Objective

**Client side solution to deal with dynamicity in OSGi**

- **Client side:** *No assumption on services*
- **Dynamics:** *Used services can be unregistered*
  - *Knowing it*
  - *Silently substitute it*
- **Consider also state-full services**
  - How to keep their internal state?
Context: OSGi

Service Oriented Architectures
- Dynamic load/unload of service
- Loosely coupled client-server through interfaces

Stale references: it’s a feature
- Unloading a service does not garbage collect it
- References to the service still usable
Stale Reference: example

- **Stale reference**: `reference to a no more registered service`
- **Service stopped**: `does it still work?`
- **OSGi specification**: `never use a stale reference`
Stale Reference: example

- **Stale reference:** *reference to a no more registered service*
- **Service stopped:** *does it still work?*
- **OSGi specification:** *never use a stale reference*
Stale Reference: example

- **Stale reference:** *reference to a no more registered service*
- **Service stopped:** *does it still work?*
- **OSGi specification:** *never use a stale reference*
Stale Reference: example

- **Stale reference**: reference to a no more registered service
- **Service stopped**: does it still work?
- **OSGi specification**: never use a stale reference
**Stale Reference: example**

- **Stale reference:** *reference to a no more registered service*
- **Service stopped:** *does it still work?*
- **OSGi specification:** *never use a stale reference*
Some Related Works

Server side approaches

- CORBA service reconfiguration [BISZ98]
  - based on the capabilities to passivate a links
  - substitution of state-full services by setting internal state
- SIROCO framework [FGIZ08]
  - each service provide get/set internal state

OSGi solutions

- Service Coroner [GD08]
  - Aspect Oriented solution to observe stale references
  - Tool for test
- Using a proxy [AOH07]
  - Current works only support state-less services
Contribution

Inspired from self healing software techniques

- Usually 3 families of treatment to recover an error:
  - to mask the error: *Redundant information*
  - to roll-forward: *Jump forward until coherent state*
  - to roll-back: *Jump backward to a previous coherent state*

Our proposition

- Introduce a proxy between service and client
- Add a registry listener
- Implement two treatments:
  - Roll-forward policy: *to throw an exception*
  - Roll-back policy: *to make a silent substitution*
Roll-forward policy: Exception

```java
EchoService echoService = 
    serviceProxyBuilder.getService(EchoService.class,
    DISABLED_AFTER_UNREGISTERED);

try {
    echoService.echo("Fail explicitly if the echo service is no more available");
} catch (UnregisteredServiceException ignored) {
    // Executed if the service is unregistered
    ignored.printStackTrace();
}
```

- The code of the client is kept simple (only a try catch)
- If multiple state-full services:
  - `catch` can be used to reset the state of each service
Roll-back policy: Service Substitution

Two cases

1. Use of state-less services
   - No constraints.
   - The same call can be replayed

2. Use of state-full services
   - Need to define a transaction
   - If multiple services: *need to provide a roll-back for other services*
Case 1: state-less service with API and substitution

// Let some echo services implementation be in memory
EchoService echoService =
    serviceProxyBuilder.getService(EchoService.class,
    RELOAD_AFTER_UNREGISTERED);

try {
    echoService.echo("You never fail while an echo service\n    is available");
} catch (UnregisteredServiceException ignored) {
    // Executed if no more services are available
    ignored.printStackTrace();
}
Case 2: Roll-back using state-full services

Transaction mechanism

- **Transacted Block**: code manipulating one or some state-full services
- If a used service disappears during TB execution:
  - If another service is available
    - (Undo the transaction effects on other services)
    - Make substitution
    - Start again the transaction
  - Else (no other service available): throw an exception

Constraints

- Define a pure transacted block *(from the client point of view)*
- *(Define a service roll-back code for other services)*
Case 2: Roll-back using state-full services

Transaction diagram for multiple services

prepare → execute → finish

substitute service → roll-back

Inspired from transactional memory

Provided by the developer of the client:
- prepare
- execute
- finish
- roll-back
Case 2: using 2 services with API and substitution

```java
ServiceReference reference = bundleContext.getServiceReference(TransactedServiceExecutor.class.getName());
transactedServiceExecutor = (TransactedServiceExecutor) bundleContext.getService(reference);

TransactedExecution<String> myExecution =
    new TransactedExecution<String>() {
        @ServiceInjection FooService service1;
        @ServiceInjection BarService service2;

        public void prepare() { /* Initialization */ }
        public <...> execute() { /* Transaction body */ }
        public void finish() { /* Closing accesses */ }
        public void rollback() { /* In case of subst. */ }
    };

String echo = transactedServiceExecutor.executeInTransaction(myExecution, new RetryForeverPolicy());
```
Proof of Concept

- API for client developer
- 2 main functionalities:
  - single service use: ServiceBuilderProxy
  - multiple services use: TransactedServiceExecutor
- Freely downloadable

https://bitbucket.org/jponge/osgi-substitution
Conclusion

Contributions

- Approach and tool to develop services affected by stale references
  - Be actively notified in case of service unload
  - Have another service in case of service unload

- Main properties of this contribution are:
  - Client side
  - No any assumption on used services
  - Can be used even if used services are state-full
  - Low development cost
  - Not expressiveness restriction of services

- Constraints
  - Client designer need to provide a roll-back method
  - Transacted block without side effects on client state
Future Work

- Propose more autonomy:
  - Use of a service logger
  - Use of a behavioral specification
  - Generate a replay behavior to not roll-back other services
Questions?


