Interfaces Introduce Extension

John Chilton

February 29, 2008

Interfaces Extension (1 / 3)

Operation Creation

Allows developers to annotate fields of the service implementation class as implementing plain old Java interfaces on behalf of the service. The extension will automatically create matching grid operations and implement these via delegation to the annotated object.

Interfaces Extension (2 / 3)

Operation Management

As interfaces are added, removed, and modified the extension continues to ensure the caGrid service is synchronized with the given interfaces.

Note: The developer may still manually add, implements, and manage other operations.

Interfaces Extension (3 / 3)

Client

Creates a client class that implements all of these interfaces via delegating method calls to the normal Introduce generated caGrid client. This enables users of the service to program againist the Java interfaces.

The Interfaces

Adder.java

```
package calc;
public interface Adder {
  int add(int x, int y);
  int add3(int x, int y, int z);
}
```

Subtracter.java

```
package calc;
public interface Subtracter {
  int subtract(int x, int y);
}
```

The Implementations

AdderImpl.java

```
package calc;
public class AdderImpl implements Adder
```

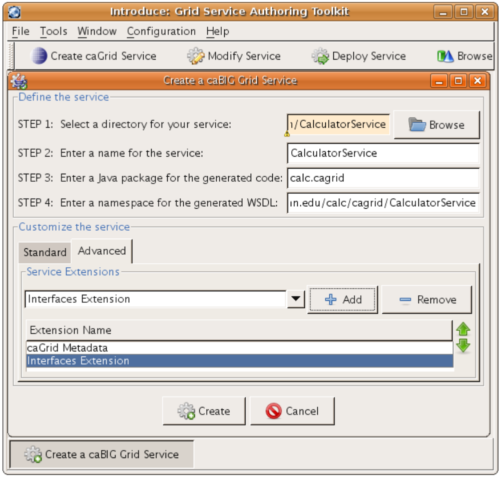
SubtracterImpl.java

```
package calc;
public class SubtracterImpl implements Subtracter
```

Calculator.java

```
package calc;
public class Calculator implements Adder, Subtracter
```

Create a calculator service for this example.



STEP 1: Select a directory for your service: ı/CalculatorService Browse

STEP 2: Enter a name for the service: CalculatorService

STEP 3: Enter a Java package for the generated code: calc.cagrid

STEP 4: Enter a namespace for the generated WSDL: ın.edu/calc/cagrid/CalculatorService

Interfaces Extension

Extension Name
caGrid Metadata
Interfaces Extension

CalculatorServiceImpl.java

```
package calc.cagrid.service;

import edu.umn.msi.cagrid..ImplementsForService;
import java.rmi.RemoteException;

public class CalculatorServiceImpl extends ... {
  public CalculatorServiceImpl() throws ... {
    super();
  }
}
```

The Interfaces extension added an import to the service implementation file for the annotation `ImplementsForService`.

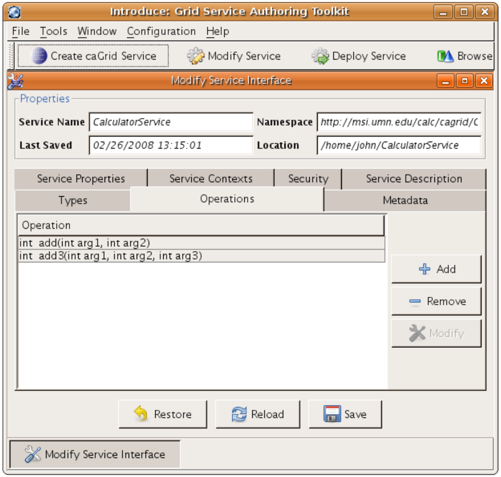
Annotating the Adder Interface

CalculatorServiceImpl.java

```
import calc.Adder;
import calc.AdderImpl;
...
public class CalculatorServiceImpl ... {
  @ImplementsForService(interfaces={"calc.Adder"})
  Adder adder;

  public CalculatorServiceImpl() ... {
    super();
    adder = new AdderImpl();
  }
}
```

Upon adding the jar file containing the calculator example to the project and saving the service in Introduce, the `Adder` operations are automatically created and implemented.



Service Name	CalculatorService	Namespace	http://msi.umn.edu/calc/cagrid/C
Last Saved	02/26/2008 13:15:01	Location	/home/john/CalculatorService

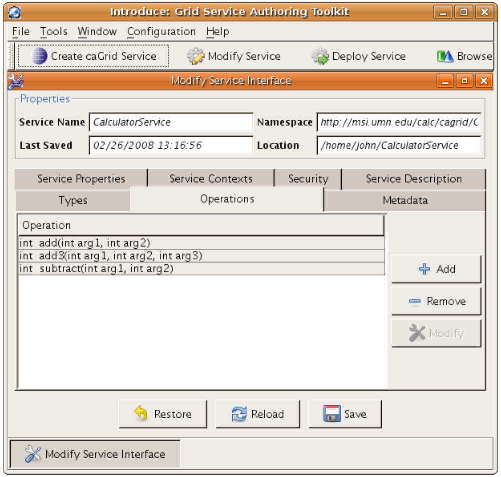
Operation
int add(int arg1, int arg2)
int add3(int arg1, int arg2, int arg3)

CalculatorServiceImpl.java

```
...
public class CalculatorServiceImpl ... {
  @ImplementsForService(interfaces={"calc.Adder"})
  Adder adder;

  @ImplementsForService(interfaces={"calc.Subtracter"})
  Subtracter subtracter;

  public CalculatorServiceImpl() ... {
    super();
    adder = new AdderImpl();
    subtracter = new SubtracterImpl();
  }
...
```



Service Name: CalculatorService
Namespace: http://msi.umn.edu/calc/cagrid/C
Last Saved: 02/26/2008 13:16:56
Location: /home/john/CalculatorService

Operation
int add(int arg1, int arg2)
int add3(int arg1, int arg2, int arg3)
int subtract(int arg1, int arg2)

Resaving causes the new operation corresponding to `Subtracter` to be created.

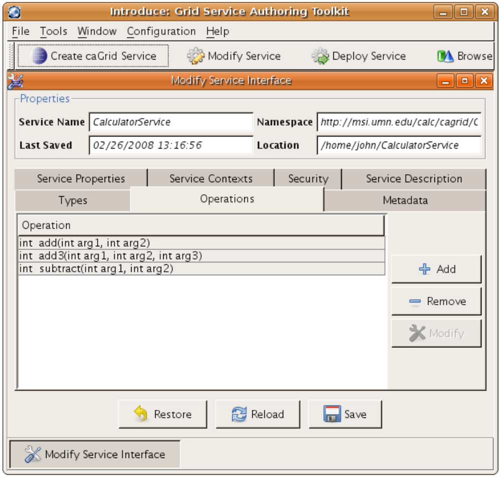
The developer can add interfaces, remove interfaces, and change the field that implements an interface and the caGrid service operations and implementations will be modified accordingly next time the service is saved in Introduce.

CalculatorServiceImpl.java

```
public class CalculatorServiceImpl ... {
  @ImplementsForService(interfaces={
                     "calc.Adder","calc.Subtracter"})
  Calculator calculator;

  public CalculatorServiceImpl() ... {
    super();
    calculator = new Calculator();
  }
...
```

After saving, all of the operations are still there though their implementations have changed.



Service Name: CalculatorService
Namespace: http://msi.umn.edu/calc/cagrid/C
Last Saved: 02/26/2008 13:16:56
Location: /home/john/CalculatorService

```
int add(int arg1, int arg2)
int add3(int arg1, int arg2, int arg3)
int subtract(int arg1, int arg2)
```

The developer can make changes to the interfaces themselves and replace the corresponding jars. After restarting Introduce and saving, the service operations and implmentations will again adapt to the interface changes.

Warning

After replacing jars in a service's `lib` directory, Introduce **must** be closed and reopened before saving the service.

Make Some Changes

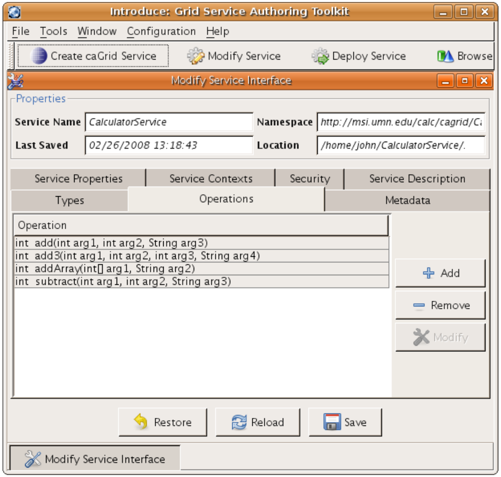
Adder.java

```
package calc;
public interface Adder {
  int add(int x, int y, String ticket);
  int add3(int x, int y, int z, String ticket);
  int addArray(int[] xs, String ticket);
}
```

Subtracter.java

```
package calc;
public interface Subtracter {
  int subtract(int x, int y, String ticket);
}
```

After replacing the jar, restarting Introduce, and saving the service, the new operations will appear.



Service Name: CalculatorService
Namespace: http://msi.umn.edu/calc/cagrid/Ca
Last Saved: 02/26/2008 13:18:43
Location: /home/john/CalculatorService/.

Operation
int add(int arg1, int arg2, String arg3)
int add3(int arg1, int arg2, int arg3, String arg4)
int addArray(int[] arg1, String arg2)
int subtract(int arg1, int arg2, String arg3)

The Interfaces Client

The extension creates a client class that implements all of these interfaces via delegating method calls to the Introduce generated caGrid client.

CalculatorServiceInterfacesClient.java

```
public class CalculatorServiceInterfacesClient
      implements calc.Adder, calc.Subtracter {
  ...
}
```

The Interfaces Client Constructors

The Interfaces client can be constructed using any of the argument combinations permitted by the Introduce generated client class or using an Introduce generated client instance directly.

Programming Againist an Interface

SomeClass.java (Pseudocode)

```
// A class that uses an Adder to do work
public class SomeClass {
  Adder adder;
  public static SomeClass create() {
    if(runningOnHost())
      // Set adder to a calculator
    else
      // Set adder to a interfaces service client
  }
  ...
```

Effect is even cleaner using a dependency injection framework, such as Spring.

Testing Againist an Interface

AdderTestCase.java

```
public class AdderTestCase {
  public void testAdder(Adder adder) {
    assert 2 == adder.add(1,1);
  }
}
```

Can use (mostly) the same code to test deployment as server side implementation.

Refactoring

As of Eclipse 3.2, you may create “scripts” out of your refactorings and distribute them. This has the potential to reduce the amount of work your users have to do if you happen to refactor the underlying interfaces and your users are programming againist these interfaces.

If you happen to be a consumer of your own services and your client project references your service project, Eclipse should do all of the refactoring of your client code automatically.

Customizing The Operations

A developer can customize certain aspects of generated operations via annotations on the underlying Java interface.

In most cases interfaces should work out of the box and these annotations are entirely optional as long as Introduce knows about the referenced types and the method names are unique and valid operation names.

Method Overloading (1 / 3)

Adder.java (Invalid)

```
package calc;

public interface Adder {
  int add(int x, int y);

  int add(int x, int y, int z);

  int add(int[] xs);
}
```

Invalid interface for annotating a caGrid service with, resulting operation names must be unique!

Method Overloading (2 / 3)

Adder.java

```
package calc;
import edu.umn.msi.cagrid.introduce.interfaces.client.GridMethod;
public interface Adder {
  int add(int x, int y);

  // Use add3 as operation name instead
  @GridMethod(operationName="add3")
  int add(int x, int y, int z);

  @GridMethod(operationName="addArray")
  int add(int[] xs);
}
```

Method Overloading (3 / 3)

Some client code

```
void foo() {
  CalculatorServiceInterfacesClient client;
  ⋮
  // Client has overloaded methods, even though
  // underlying grid service cannot.
  int sumOf2 = client.add(1,2);
  int sumOf3 = client.add(1,2,3);
  int sumOfArray = client.add(new int[]{1,2});
  ⋮
}
```

XML Type Disambiguation (1 / 4)

Adder.java

```
public interface Adder {
  int add(int x, int y, String ticket);
}
```

The extension must guess what XML type the developer wants `ticket` to map to, both of the types `string` or `anySimpleType` from the W3C XML Schema namespace map to `java.lang.String`.

XML Type Disambiguation (2 / 4)

Other out of the box ambigous types:

- `java.lang.String` could map to `string`* or `anySimpleType`.
- `byte[]` could map to an array of bytes, a `hexBinary`, or a `base64Binary`*.
- `byte[][]` could map to an array of `hexBinarys` or `base64Binarys`*.

* indicates the Interfaces extension default choice.

XML Type Disambiguation (3 / 4)

You can override the default XML mapping on parameters as follows.

Adder.java

```
import edu.umn.msi.cagrid.introduce.interfaces.client.GridParam;
public interface Adder {
  int add(int x, int y,
      @GridParam(namespaceURI="http://www.w3.org/2001/XMLSchema",
                 localPart="anySimpleType")
      String ticket);
}
```

XML Type Disambiguation (4 / 4)

Foo.java

```
import edu.umn.msi.cagrid.introduce.interfaces.client.GridResult;
public interface Foo {
  @GridResult(namespaceURI="http://www.w3.org/2001/XMLSchema",
              localPart="hexBinary")
  byte[] foo(int x);

  @GridResult(namespaceURI="http://www.w3.org/2001/XMLSchema",
              localPart="byte")
  byte[] foo2(int x);
}
```

Note: The extension determines if the mapped type should (as in `foo2`) or should not (as in `foo`) be an array automically.

Additional Annotation Options

`GridParam` can also be used to set the name of the parameter with the `name` field.

All three annotations have a `description` field which can be used to set a description for the method, parameter, or result respectively.

`GridMethod` has a `boolean` field `exclude` that can be used to prevent the creation of operation for that method. If the corresponding method is then called on the Interfaces client, a `java.lang.UnsupportOperationException` is thrown.

Exception Handling

The Introduce generated client code only throws `RemoteExceptions`. These could be because the delegated method threw an exception or because of some other issue such as a connection problem. If the interface method permits a `RemoteException` to be thrown, this exception will be thrown as is, if not the exception is wrapped in a runtime exception first.

Essentially, the fact an exception is thrown by the underlying implementation survives and reaches the client, but the type and content of the exception will likely not.