FraSCAti, prenez le contrôle de vos applications

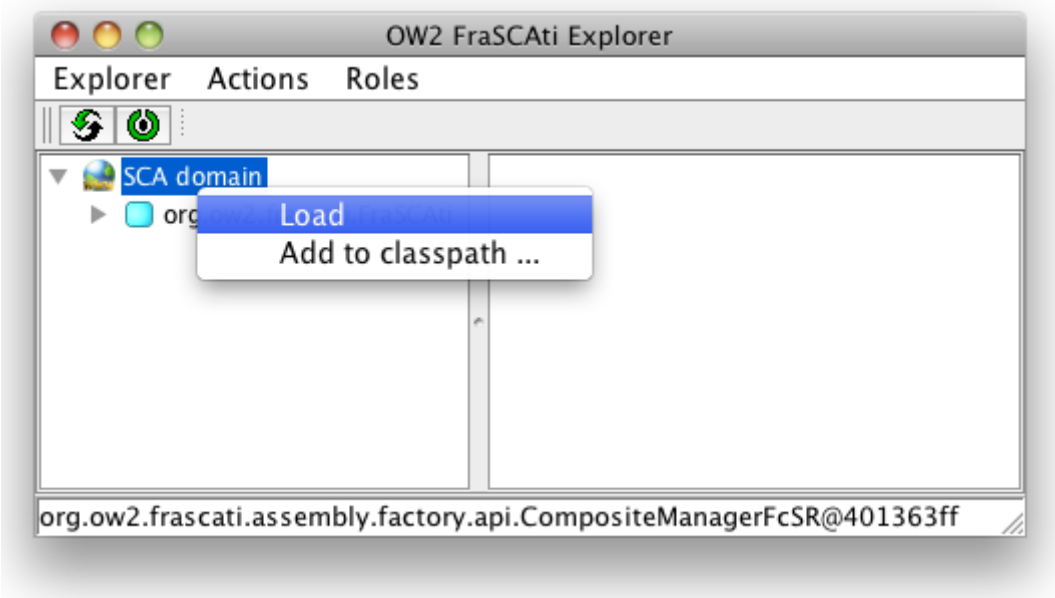
Contrôler les applications en cours d'exécution n'est pas chose aisée. Nous présentons dans cet article différents moyens permettant de reprendre la main sur vos applications grâce à FraSCAti.

Introduction

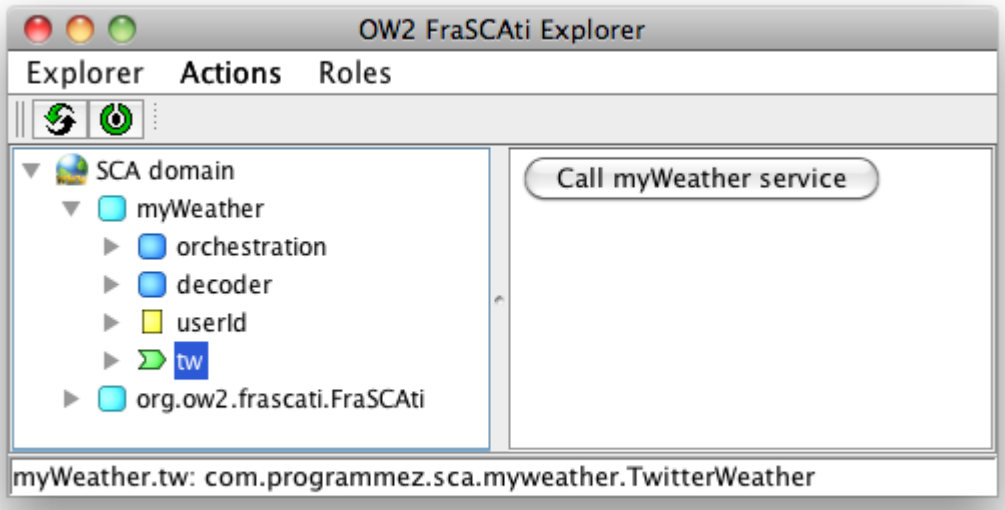
Dans le numéro précédent, nous avons vu comment SCA simplifie la réalisation d'applications orientées services tout en donnant un cadre architectural (SOA facile avec SCA). Nous allons ici nous intéresser à une autre préoccupation: comment observer une application en cours d'exécution, modifier sa configuration initiale, ou la faire évoluer sans la redéployer ? FraSCAti traite ces différentes problématiques en supportant la reconfiguration dynamique d'assemblages SCA. Nous les mettrons en pratique à l'aide de l'exemple introduit dans l'article précédent: `MyWeather`. Pour rappel, cet exemple permet d'interroger un compte Twitter afin de récupérer la localisation de l'utilisateur puis d'interroger un service météo pour connaître la météo à cette localisation. Nous compilerons cet exemple avec un script spécifique (`compile`, fourni avec les sources) afin de pouvoir développer un service technique (intent) intégré dans la plateforme FraSCAti.

Examiner les applications en cours d'exécution

FraSCAti Explorer est un outil capable d'observer, dans le détail, les applications en cours d'exécution. Il permet de naviguer dans l'architecture d'un assemblage SCA (les composants, leurs services, références, bindings, propriétés métiers et aspects techniques). Ceci prend tout son intérêt lorsque l'on sait que FraSCAti permet de faire évoluer cette architecture dynamiquement. FraSCAti Explorer est donc un véritable microscope vous permettant de visiter et faire évoluer votre application. Pour lancer FraSCAti Explorer, tapez `frascati explorer`. Chargez vos composites à l'aide d'un clic droit sur le domaine SCA et du menu contextuel `Load`. Naviguez dans votre système de fichiers jusqu'à l'archive `myWeather.jar`. Double-cliquez dessus pour naviguer à l'intérieur du jar et chargez le composite `myWeather.composite`.



L'explorer permet aussi de définir des plugins pour nos applications métiers. Dans cet exemple, nous fournissons un panel pour le service `tw`. Il sera affiché dans la partie droite lorsque le service sera sélectionné. Il nous permet d'invoquer le service par un simple clic sur le bouton `Call myWeather service` et affiche le résultat de la requête.



Testons maintenant notre service en utilisant le panel. Vous devriez obtenir une réponse ressemblant à ceci:

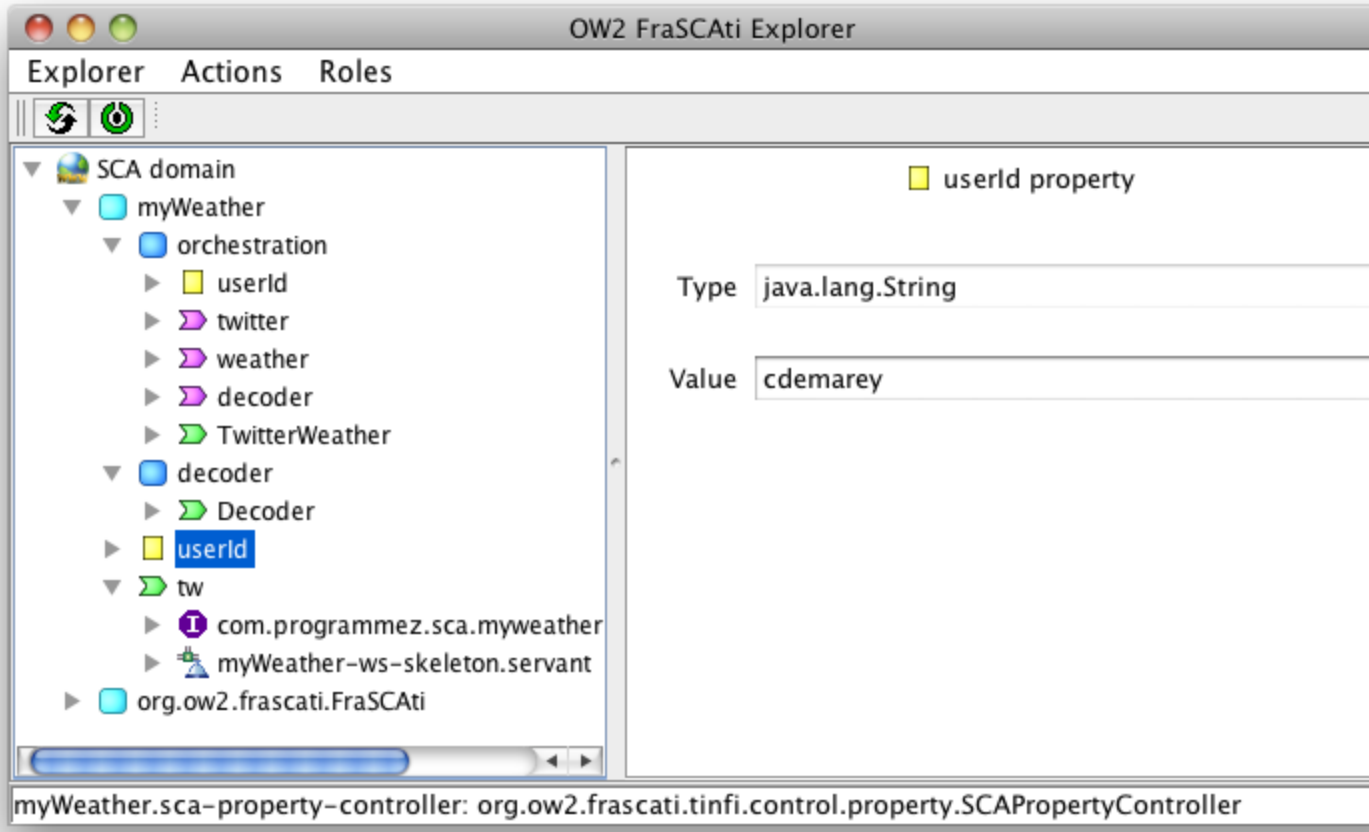
```
Current weather in Lille:
<?xml version="1.0" encoding="utf-16"?>
<CurrentWeather>
  <Location>Lille, France (LFQQ) 50-34N 003-06E 52M</Location>
  <Time>Sep 28, 2010 - 07:00 AM EDT / 2010.09.28 1100 UTC</Time>
  <Wind> Variable at 2 MPH (2 KT):0</Wind>
```

```
  <Visibility> 2 mile(s):0</Visibility>
  <SkyConditions> overcast</SkyConditions>
  <Temperature> 57 F (14 C)</Temperature>
  <DewPoint> 57 F (14 C)</DewPoint>
  <RelativeHumidity> 100%</RelativeHumidity>
  <Pressure> 30.00 in. Hg (1016 hPa)</Pressure>
  <Status>Success</Status>
</CurrentWeather>
```

A noter que ce service n'est pas toujours disponible (saturation du serveur). Dans ce cas, l'application ne sera pas chargée et vous aurez le message d'erreur suivant: `WSDLException (at /html): faultCode=INVALID_WSDL`.

Reconfiguration dynamique

La visualisation est une chose mais l'interaction en est une autre. FraSCAti Explorer permet donc aussi d'interagir avec les applications. Il est possible, par exemple, de modifier la valeur d'une propriété sur un composant en cours d'exécution. Avec notre exemple, nous pouvons changer l'identifiant Twitter de l'utilisateur pour lequel nous souhaitons récupérer les informations. Pour vérifier que le changement a bien été pris en compte, il nous suffit d'invoquer à nouveau notre service grâce au panel disponible sur le service `tw`.



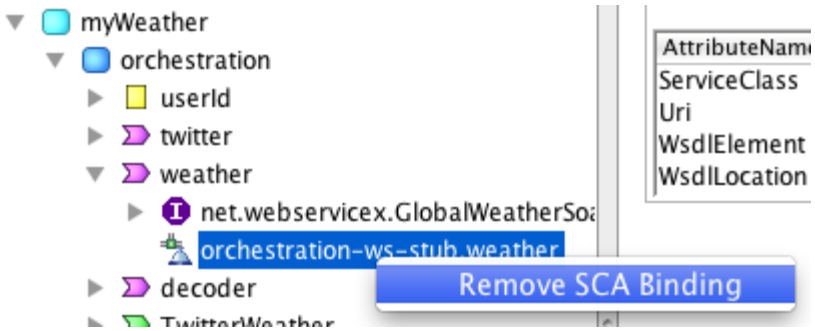
Avec FraSCAti Explorer, nous pouvons également ajouter (ou supprimer) des composants, mettre à jour une liaison (wire) ou un binding. Nous allons mettre en oeuvre ces techniques sur notre exemple.

Supposons que le service météo que nous utilisons soit en panne. Nous allons reconfigurer notre application afin qu'elle utilise un autre service météo. En général, les APIs des services ne sont pas les mêmes. Pour contourner ce problème, nous ajouterons un composant adaptateur qui respectera l'interface du service météo utilisé par défaut, et traduira les appels vers notre nouveau service météo. Voyons l'implantation de ce composant:

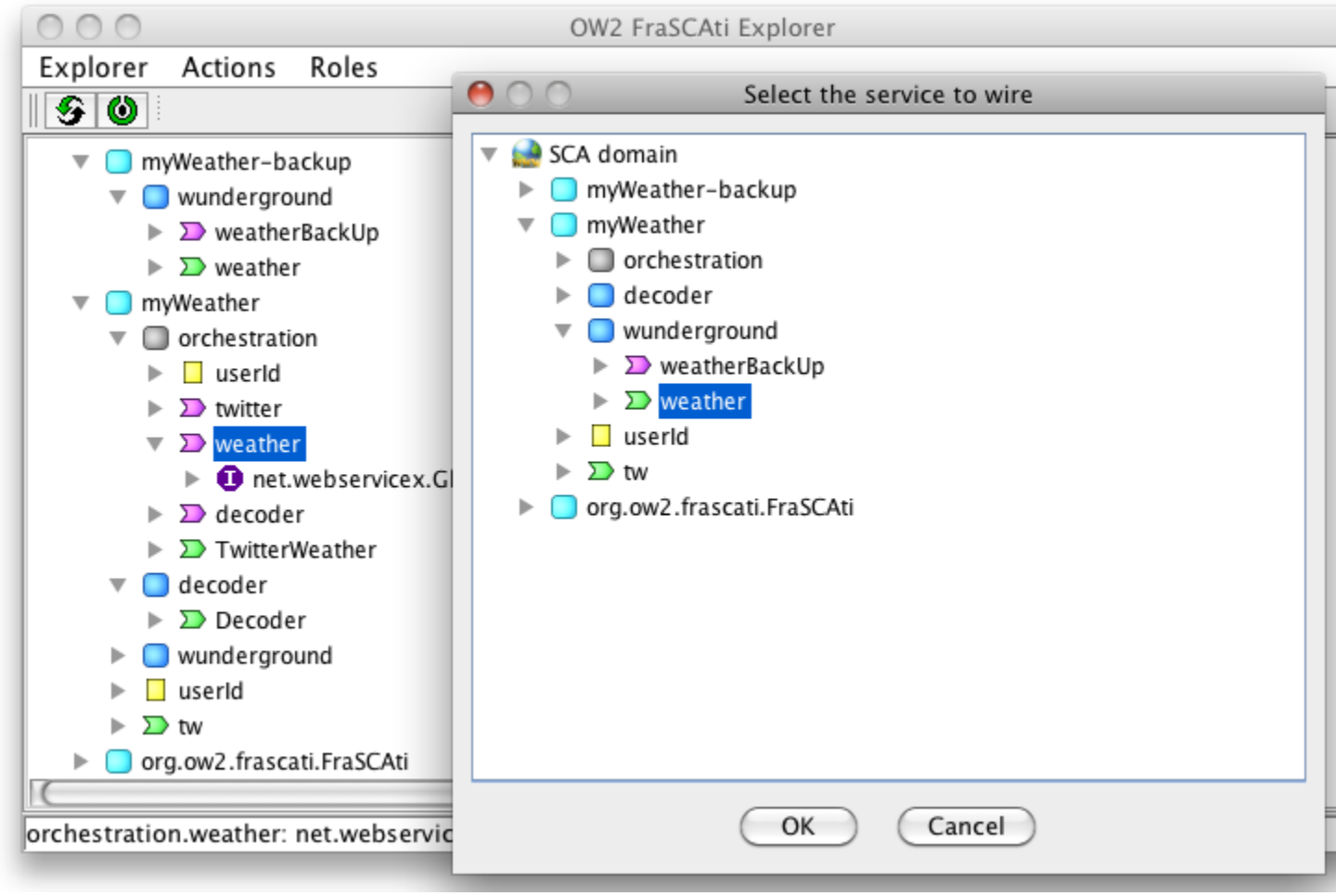
```
public class WundergroundProxy implements GlobalWeatherSoap {
        /** Reference to the Weather Underground service. */
        @Reference private Wunderground weatherBackUp;

        @Override
        public String getCitiesByCountry(String country) {
                // Not implemented
                return null;
        }

        @Override
        public String getWeather(String city, String country) {
                StringBuilder sb = new StringBuilder().append(city).append(',').append(country);
                Forecast forecast = weatherBackUp.getWeather( sb.toString() );
```

```
                return forecast.getTxt_forecast().getForecastdays().get(0).getFcttext();
        }
}
```

Nous allons charger ce nouveau composite SCA `myWeather-backup` via le menu `Load` de FraSCAti Explorer. Une fois chargé, nous ajoutons le composant `wunderground` dans notre composite `myWeather`. Ceci se réalise en faisant un glisser/déposer du composant `wunderground` vers le composite `myWeather`.
Nous supprimons la liaison entre le composant `myWeather` et le service météo défaillant, en supprimant le “binding” web service sur la référence `weather` via un clic droit sur le binding (voir figure ci-dessous)



Ensuite, nous stoppons le composant `orchestration` en le sélectionnant, puis avec un clic droit nous choisissons l’action `stop`. Nous pouvons alors réaliser la liaison (wire) entre sa référence `weather` et notre service météo de secours `wunderground`. Notez que lorsque le composant `orchestration` est stoppé, tous les appels émis vers ce composant sont mis en attente et seront traités une fois le composant redémarré. Ainsi, le service d’orchestration est toujours disponible vis-à-vis du client.
Toujours à l’aide du clic droit, mais cette fois sur la référence, nous sélectionnons le menu “`Wire ...` “. Une boîte de dialogue s’ouvre, nous sélectionnons le service `weather` du composant `wunderground` qui se trouve maintenant dans le même composite que le composant `orchestration`. Nous validons en cliquant sur `OK` et nous redémarrons le composant `orchestration`, notre reconfiguration est terminée !



Appelons de nouveau notre service grâce au panel disponible sur le service `tw`. Nous voyons maintenant que nous obtenons toujours les informations météo mais légèrement différentes aussi bien au niveau des données que du format. C'est logique car nous avons changé de fournisseur de service météo.

FraSCAti FScript, langage d'interrogation et de reconfiguration d'architecture

Nous avons, pendant l'exécution, modifié l'architecture de notre application avec l'outil graphique fourni par FraSCAti. Ceci est très pratique pour le test ou réaliser de petites modifications d'architecture, mais cela est beaucoup moins envisageable pour des applications en production.
FraSCAti propose d'autres moyens de reconfigurer une application autrement que par son outil de visualisation: soit via une API Java spécifique, soit via un langage dédié particulièrement adapté aux interrogations d'architecture : FraSCAti FScript (extension SCA de FScript, http://fractal.ow2.org/fscript).

Ce langage possède une syntaxe similaire à XPath. Nous ne détaillerons pas toutes ses possibilités ici (plus d'informations sur http://frascati.ow2.org/doc/current/ch08.html). Nous ne retiendrons que quelques éléments essentiels de FraSCAti FScript : l'utilisation d'axes de navigation pour parcourir une architecture (`scachild`, `scaservice`, `scaproperty`, `scawire`, `scabinding`, `scaintent`, etc.), la présence de noeuds sources et cibles de chaque côté d'un axe, et la possibilité de définir des variables et des procédures. Pour mieux comprendre, prenons un exemple:

```
$domain/scachild::*
```

L'instruction ci-dessus demande au domaine SCA (variable prédéfinie) la liste de ses fils (sans restriction sur le nom des fils). Le résultat est le suivant:

```
[#<scacomponent: myWeather>, #<scacomponent: myWeather-backup>,
#<scacomponent: org.ow2.frascati.FraSCAti>]
```

Maintenant, si nous désirons obtenir un fils particulier, nous utilisons l'expression suivante:

```
orch = $domain/scachild::myWeather/scachild::orchestration;
```

Nous voyons ci-dessus que nous pouvons enchaîner les axes de navigation et stocker le résultat (des noeuds) dans une variable. Il existe aussi quelques raccourcis. L'expression ci-dessus est équivalente à:

```
orch = $domain/scadescendant::orchestration;
```

Pour obtenir toutes les références du composant orchestration, nous utilisons:

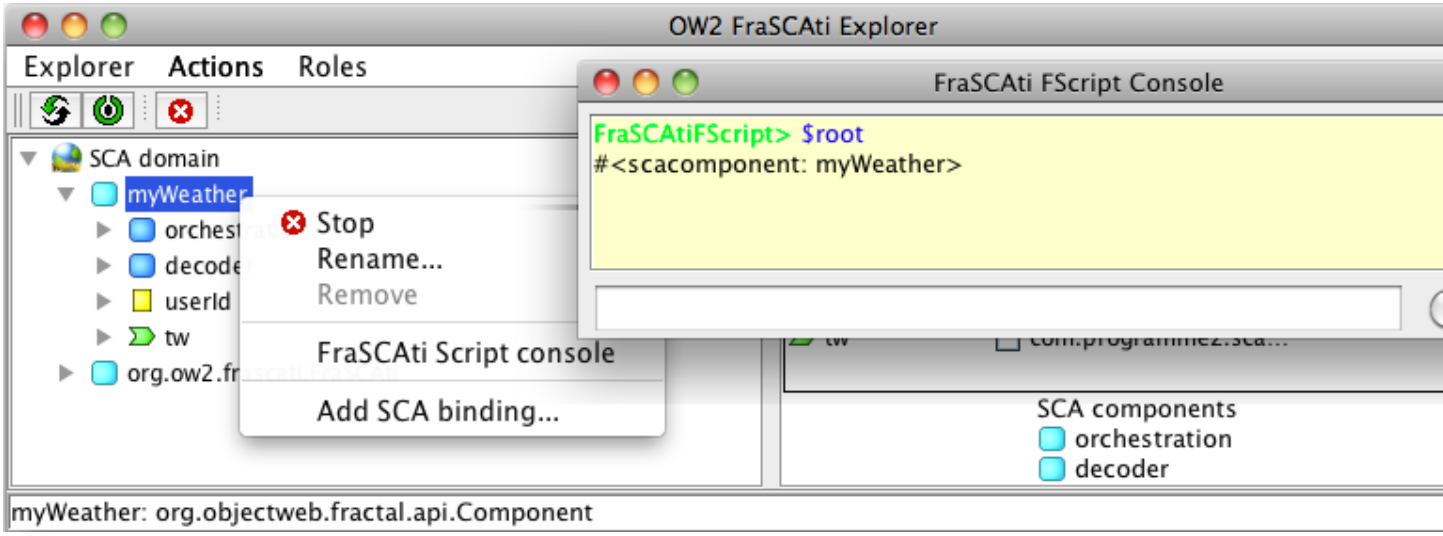
```
$orch/scareference::*
```

Après un bref aperçu de ce langage, intéressons-nous à un cas d'utilisation. Si nous reprenons la reconfiguration dynamique effectuée avec FraSCAti Explorer, nous pouvons la traduire avec la procédure FraSCAti Script suivante:

```
action switchToWundergroundService() {
    orchestration = $domain/scadescendant::orchestration;
    weatherRef = $orchestration/scareference::weather;
    wsBinding = $weatherRef/scabinding::*;
    remove-scabinding($weatherRef, $wsBinding);

    wunderground = $domain/scadescendant::wunderground;
    myWeather = $domain/scachild::myWeather;
    add-scachild($myWeather, $wunderground);
```

```
    set-state($orchestration, "STOPPED");
    wundergroundService = $wunderground/scaservice::*;
    add-scawire($weatherRef, $wundergroundService);
    set-state($orchestration, "STARTED");
}
```

Cette procédure peut être enregistrée et réutilisée par le moteur de FraSCAti Script. Voyons comment. Tout d'abord, nous démarrons FraSCAti Explorer avec l'option `-s` afin d'activer le plugin FScript: `frascati explorer -s`.



Ensuite, nous chargeons les composites `myWeather` et `myWeather-backup`. A l'aide d'un clic droit sur le composite `myWeather`, choisissons `FraSCAti Script console`. Une console s'ouvre. Vous pourrez taper vos instructions FScript de manière interactive ou utiliser l'éditeur pour écrire des procédures, ouvrir des fichiers et charger des scripts dans le moteur. Nous allons cliquer sur `Editor`, taper la procédure décrite ci-dessus puis cliquer sur `Register procedures`. La procédure `switchToWundergroundService` est maintenant disponible. Cliquons à nouveau sur editor pour masquer la fenêtre et exécutons l'instruction suivante : `switchToWundergroundService()`

La reconfiguration a été réalisée. Vous pouvez vous en assurer en rafraîchissant l'affichage de FraSCAti Explorer.

Le script de reconfiguration est bien plus rapide que les glisser/déposer réalisés précédemment avec FraSCAti Explorer. Mieux encore, nous aurions pu prévoir la panne du service météo et intégrer la gestion de cette panne dans notre application. Pour cela, deux possibilités s'offrent à nous : écrire un composant en Java et utiliser le moteur FraSCAti FScript depuis celui-ci (utile pour réaliser des traitements en plus de la reconfiguration) ou écrire un composant dont l'implantation sera un script FraSCAti FScript. Pour réaliser ce composant, il suffit de copier/coller l'action `switchToWundergroundService` dans un fichier que nous nommons `myreconfig.fscript`. Ensuite, dans le descripteur d'assemblage de `myWeather-backup`, ajoutons les lignes suivantes:

```
<component name="switch-to-wunderground-service">
```

```
  <frascati:implementation.script script="myreconfig.fscript"/>
  <service name="myReconfig">
    <interface.java interface="com.programmez.sca.myweather.MyReconfig"/>
  </service>
 </component>
```

Vous noterez que nous devons écrire une interface Java reflétant les procédure disponibles dans le script. Voici son implantation:

```
public interface MyReconfig
{
        void switchToWundergroundService();
}
```

Comme pour tout composant SCA, il est possible d'exposer l'action `switchToWundergroundService` via un protocole d'accès comme SOAP, RMI etc. Ainsi, l'architecture de notre application pourra être reconfigurée avec un simple appel distant.
En ajoutant un mécanisme qui contrôle la disponibilité du service, on obtiendra alors une application autonome, capable de s'adapter en cas de panne du service météo dont elle dépend.

Ajouter des aspects non-fonctionnels

La spécification SCA propose aux développeurs de composants de définir des intentions (intents SCA) sur les services et références des composants. Les intentions traduisent en général des préoccupations non fonctionnelles telles que authentification, confidentialité, etc. que l'on décrit à haut niveau. Celles-ci sont généralement implantées par la plateforme SCA et appliquées au déploiement des composants. Ici, FraSCAti innove sur deux aspects: libre au développeur de fournir ces propres intents. Pour ne pas le perturber, il développera ces intents sous forme de composants SCA, même formalisme pour le métier et le non fonctionnel. Second point, vous l'aurez peut-être deviné, ces intents peuvent être appliqués au déploiement mais aussi dynamiquement, lorsque les applications s'exécutent, grâce à un mécanisme dérivé de la programmation orientée aspects (AOP). Lorsque l'on place un intent sur un service ou une référence, un intercepteur est généré et déroute l'appel vers le composant réalisant l'intent. Il est alors possible de réaliser des pré et post-traitements, de reprendre le cours d'exécution normal de l'application.
Écrire un intent dans FraSCAti est très facile. Il suffit d'écrire un composant SCA proposant un service dont l'interface est `org.ow2.frascati.tinfi.control.intent.IntentHandler` pour laquelle il n'y a qu'une seule méthode à implanter: `invoke`. Voyons ceci sur un exemple simple de journalisation des appels sur un composant. Nous allons écrire cet intent `log`. Voici sa description d'architecture:

```
<component name="log">
  <implementation.java class="com.programmez.sca.intent.Log" />
```

```
  <service name="intent">
    <interface.java interface="org.ow2.frascati.tinfi.control.intent.IntentHandler"/>
  </service>
  <property name="header">[FRASCATI-BASIC-LOG] </property>
</component>
```

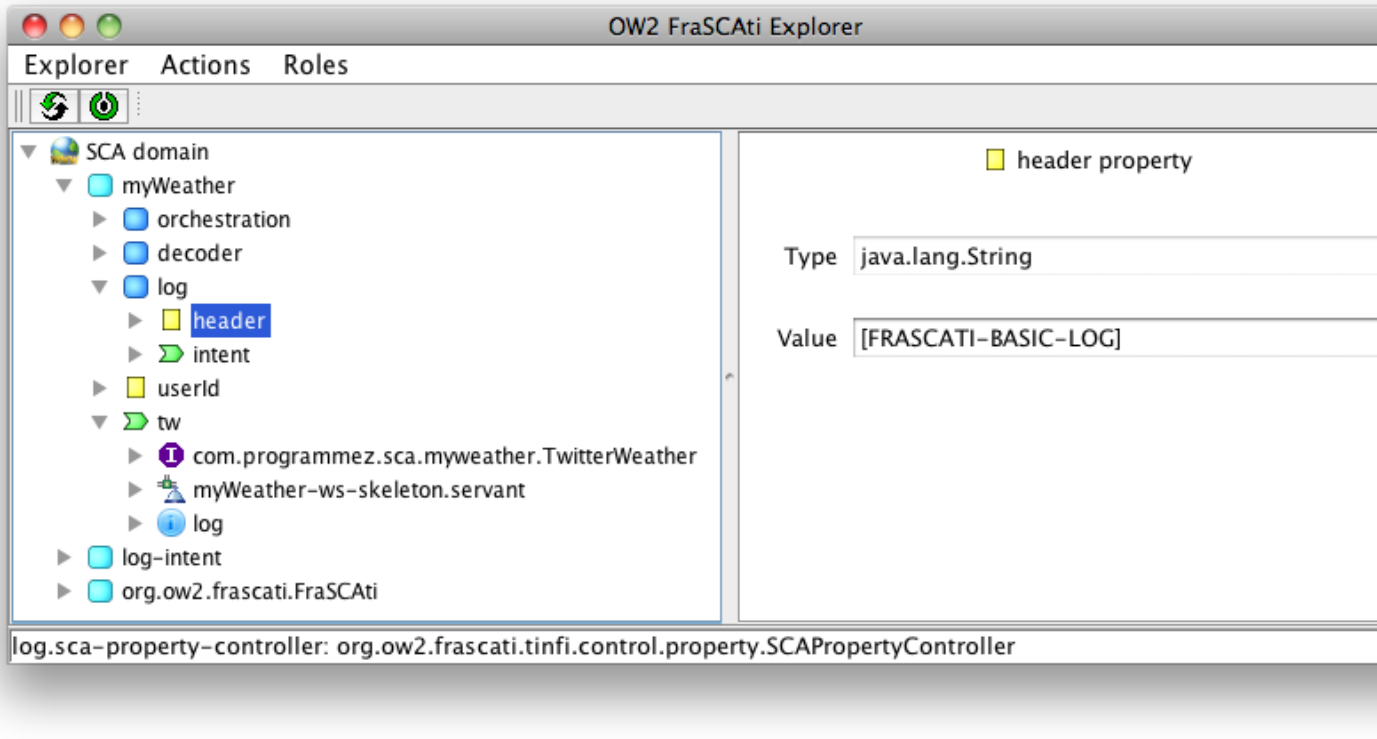
Et maintenant l'implantation de la classe `Log` :

```
public class Log implements IntentHandler {
        /** A configurable header to add to log traces. */
        @Property
        protected String header;

        @Override
        public Object invoke(IntentJoinPoint ijp) throws Throwable {
                // Before the current invocation.
                System.err.println(header + " Before proceed");
                // Proceed the current invocation.
                Object ret = ijp.proceed();
                System.err.println(header + " result of proceed:" + (ret==null ? "null" :
ret.toString()));
                // After the current invocation.
                System.err.println(header + " After proceed");
                return ret;
        }
}
```

Nous pouvons maintenant le charger dans FraSCAti Explorer comme un composant classique. Pour utiliser cet intent, nous allons le tisser dynamiquement sur les services et références sur lesquels il doit s'appliquer. Par exemple, faisons un glisser/déposer du composant `log` vers le service `tw` pour tracer les appels à ce service. Nous voyons les traces suivantes s'afficher:

```
[FRASCATI-BASIC-LOG] Before proceed
[FRASCATI-BASIC-LOG] result of proceed: Current weather in Lille: Chance of
Rain. High 18°C (64°F). Winds 10 kph NNE
[FRASCATI-BASIC-LOG] After proceed
```

Comme notre intent de journalisation est un composant SCA, il peut lui aussi être reconfiguré.



Si nous avions voulu positionner cet intent au démarrage de l'application, nous aurions modifié la déclaration du service `tw` en ajoutant le mot-clé `requires`:

```
<service name="tw" promote="orchestration/TwitterWeather"
requires="log-intent">
```

De cette façon, nous indiquons la liste des intents à appliquer à un service ou une référence.

Conclusion

Nous avons pu voir tout au long de cet article qu'il est possible de garder le contrôle sur des applications en cours d'exécution: observation de l'architecture, reconfiguration de celle-ci, ajout/suppression d'aspects non fonctionnels, etc. Il est donc possible de faire évoluer des applications sans devoir les redéployer, permettant ainsi de pouvoir répondre à différentes préoccupations: gestion des pannes, corrections de bugs, évolutions à chaud.

Christophe Demarey et Damien Fournier,
Ingénieurs de Recherche - INRIA.