PDQ service module

February 4, 2014

Abstract

This document explains how the service module is designed. It is aims at whoever needs to use or extend it.

this document refers to Yahoo and Google services whose description file can be found respectively at:

- svn://svn.cs.ox.ac.uk/QDDA/code/trunk/demo/test/services/yahoo-services.xml
- svn://svn.cs.ox.ac.uk/QDDA/code/trunk/demo/test/services/google-services.xml

1 Service descriptions

The attached files named "google-services.xml" and "yahoo-services.xml" are examples our service descriptions.

1.1 Definitions

In the following, capitalized names correspond to actual classes in the code:

- a *`Service`* is an implementation or `Relation`, whose `access()` method performs the an access with input and output tuples, hiding all the details about how it is actually done.
- a *`ServiceRepository`* is a collection of `Service`s
- a *`UsagePolicy`*, is a rule that a service vendor imposes on users (e.g. max per-day request, max per-day results, paging, monetary cost, "backoff" policy, etc.)
- an *`InputMethod`* is a scheme imposed on the user to provide input value (e.g. URL params, URL path element, batch inputs, etc.)
- an *`OutputMethod`* is a schema to extract a piece of results (that usually comes as a tree) and put it into a tuple. Currently, this is expressed as a path.

1.2 Service descriptions structure

The service description files are structured as follows:

- each represent exactly one `ServiceRepository`
- a `ServiceRepository` may define a set of common usage policies (shared across all its services)
- a `ServiceRepository` may define a set of common input methods (shared across all its services)

Then follow a collection of Services, each one is structured as follows:

- a `Service` has a name (the relation name), a protocol, some base URI, link to the online documentation, and a path fragment defining how sequential results are delimited.
- a `Service` may define specific usage policies and input methods.
- then comes a list of static inputs and attributes. These may be interleaves, but the order is important.
- finally, comes a list of access methods, with the usual name, type, inputs positions, and (currently fixed) cost.

1.3 Inputs and attributes

Static inputs are typical inputs that are required for the service to function, but have little (or nothing) to do with the results content. They are specified at the level of a single Service as they may be different from one `Service` to another in a single `ServiceRepository`. They are used during an access to form the request, but they do not appear as part of the relation's attribute. Each static input is associated with an `InputMethod`, and possibly a static value.

`Attribute`s are the actual attributes of the `Service/Relation`. In addition to the usual type, that are associated with an `InputMethod` (if they can be used as inputs), and an output path. Some attributes may be used as both inputs and outputs, in which case, they required both input and output methods. However, sometimes an attribute can only be used as an input, in which case, it is not always possible to get a value for it from the service response. For instance, the YahooPlaces has a "keyword" attribute. If used, it is possible to place the input value in the output tuple. When it is not used, say the woeid was used as input instead, the attributes will have no value in the resulting tuples.

The limited access method's input positions corresponds to attributes only (i.e. static inputs are ignored).

1.4 Input and output methods

The rationale for introducing the input and output method was to avoid (or a least postpone) the need for defining complex types in Java class. It is easy to get caught into defining classes for every complex type we come across, and I am afraid we would have to maintain a lot of those if we follow this path.

What I came up with may be perfectible, but I think it will allows us to cover a lot of cases will sticking to simple type for our relation attributes. You will notice that some input methods feature a "template" attribute, with values of the form ".type('1')", such as (in the google file)

```
<input-method name="locations" type="url-param" template="{1},{2}"  />
```

When an attribute or static input is associated with such an input method, it must specify where in the template its value will be placed. For instance,

```
<attribute name="latitude"  input-method="locations.1" path="location/lat" ... />
<attribute name="longitude" input-method="locations.2" path="location/lng" ... />
```

An access with input tuple ("33.5", "-142.1") will form a URL param "locations=33.5,-142.1".

The outpath extract scheme is primitive for now, i.e. does now always very complex expression. But it can already let us "simplify" complex type, as in the latitude/longitude case above.

1.5 Usage policies

One can define usage policies for various kind of cases. Usage policies are class than implement either or both of the `PreAccessUsagePolicy` and `PostAccessUsagePolicy`.

The actual behaviour of this method is proper to each implementation. A usage policy may for instance, (i) put the current thread to sleep for a given period of time if the service requires the client to wait before sending a request, (ii) throw a `UsagePolicyViolationException` if the policy prevent the access complitaly (e.g., a quote was reached), (iii) modify the service request building process (e.g., if paging is required to obtained a complete results for a single access), etc.

The `UsagePolicy` classes follow an event based architecture. Pre- and post-access events are generate for each access. A policy may have to pre- and/or post-process access, which explains why they may implement the `processAccessRequest(RESTRequestEvent event)` and `processAccessResponse(RESTResponseEvent event)` methods. The `RESTRequestEvent` and `RESTResponseEvent` events given as parameters to these method, contains the information about the access, its inputs and outputs. The policy may use that information to determine whether there is a violation, or event modify them, e.g., a paging policy will adapter the request to add page select information.

Each `Service` may be initialized with a collection of `UsagePolicy`s or register/unregister policies at runtime. Policies register at the service repository level are shared by all services in the repository, i.e., any limit enforced by the policies must be globally satisified by all services. For instance, a shared policies of n requests per day, will generate an `AccessException` if the limit is reached through any combinations of accesses.

A service may or may not adhere to shared policy. For a service to adhere to a policy, the service defintion must referred to the policy by its name only. If a policy is fully defined in a single service definition, the scope of that policy will that service exclusively.

2 How-To. . .

2.1 . . . create my own service repository description.

TODO

2.2 . . . setup a schema for existing service descriptions.

Once the service description file is ready, one can refer to the service defined in it as relations in a schema.

For this, one needs to include the proper set of source descriptions at the beginning of the schema file. This is an optional block and was so far used for defining schemas whose tables could be discovered from a DB. In this case, "source" elements define an external file that contains description for services. Required attributes are "discoverer" which here takes the value "uk.ac.ox.cs.pdq.builder.io.xml.ServiceReader", and "file" which take the relation path to the service description file.

The schema is defined by listing the relations one wants to include from those defined in the external service description. Each service comes with a list of access methods. One can select in the schema which of those will be used/allowed. Note that this is why access-methods entries in the schema.xml only have a name and, optionally, a cost. The other details about the access-methods are defined in the service descriptions.

Note that dependencies are defined as before. Here, we have an inclusion dependency from the latitude and longitude of the Yahoo service to those of Google.