Documentation for the libsf1r library

Paolo D'Apice

March 20, 2012

Abstract

This document contains the detailed description of the libsf1r library included in the izenelib project. The library provides two client drivers for the SF1 search engine.

Contents

1 Introduction

The libsf1r provides two C++ client drivers for the SF1, namely:

single for connecting to a single SF1 instance (Figure 1a on the following page)

distributed for connecting to a cluster of SF1 instances registered with a ZooKeeper server (Figure 1b on the next page)

2 The SF1 communication protocol

The SF1 search engine accepts connections using the custom socket protocol described in Table 1 on page 4.

The custom protocol requires the following information:

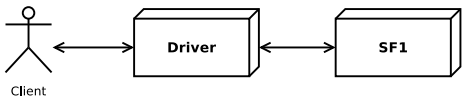
sequence number non-zero unsigned integer associating a request and a response

message length unsigned integer

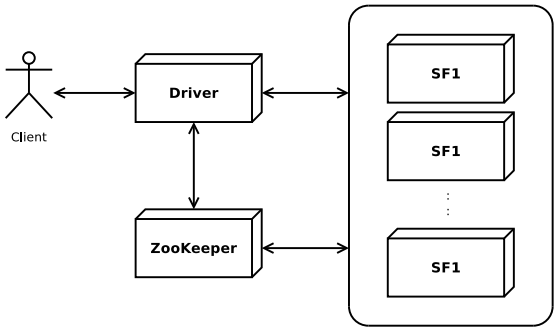
message body sequence of characters

The actual message body can be in one of the following formats:

- JSON



(a) Single driver



(b) Distributed driver

Figure 1: SF1 driver scheme.

3 Implementation details

3.1 Architecture

The main software architecture is depicted in Figure 2 on page 3. The following modules are defined:

RawClient sends requests to and receives responses from an SF1 instance via socket

Writer interface for read and write operations on the message body for pre/post processing operations

PoolFactory factory for the connection pools to an SF1 instance

ConnectionPool of re-usable RawClient instances; the pool can be configured to be fixed-sized or to automatically grow its size up to an upper limit

ZooKeeperRouter implements a topology monitor for the nodes actually registered with the ZooKeeper server; it also realizes an active router to the running SF1 instances

Sf1Node represents a running SF1 instance registered with ZooKeeper

Sf1Topology provides several views on the actual topology registered with ZooKeeper, allowing accessing a node directly or by through its hosted collections

Sf1Watcher event handler for changes in the ZooKeeper registered nodes

RoutingPolicy interface for the routing policies used for request forwarding

The classes implementing the aforementioned components are shown in Figure 3 on page 4.

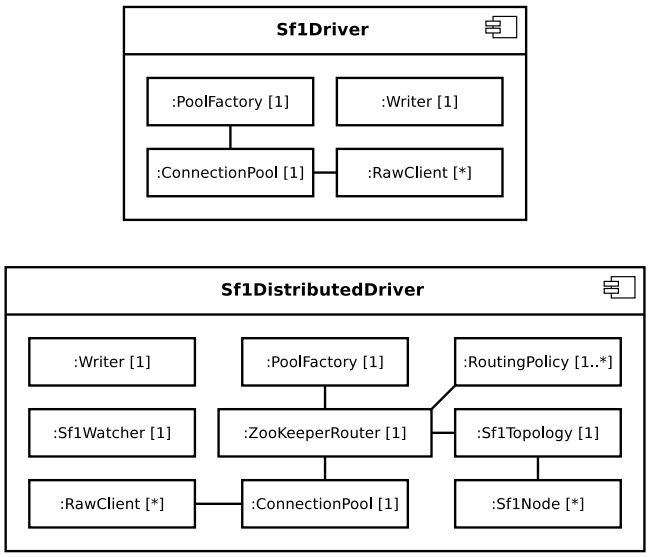


Figure 2: Composite structure diagram.

3.2 Provided interface

The SF1 driver exposes the following interface:

```
std::string
call(const std::string& uri,
     const std::string& tokens,
           std::string& request)
throw(std::runtime_error);
```

where:

uri is the URI of the request, in the format /controller/action

tokens optional tokens to be sent to the SF1

request the message body

3.3 Behavior

The general behavior of the driver is depicted in Figure 4 on the next page. During the preprocessing stage, the request is modified in such a way that it contains both the controller and the action specified in the URI. Then a connection is obtained either directly from the connection pool (single driver, Fig. 5 on page 5) or through ZooKeeper (distributed driver, Fig. 6 on page 5). In the latter case the ZooKeeperRouter acts as a proxy to the ConnectionPool.

For the distributed driver, the information contained in the request are used in order to retrieve a proper SF1 instance from the actual topology. For example, the collection parameter can be used to route a request only to the SF1 instances actually hosting such collection.

Table 1: SF1 communication protocol.

field	sequence number	message length	message body
size (bytes)	4	4	variable

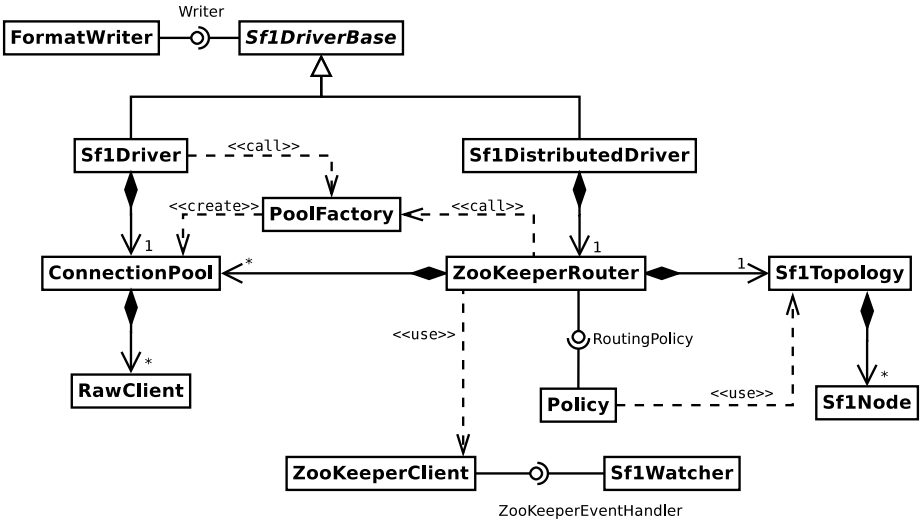


Figure 3: Class diagram.

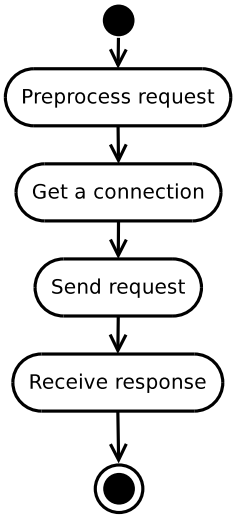
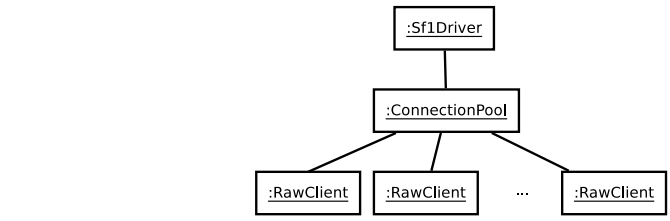
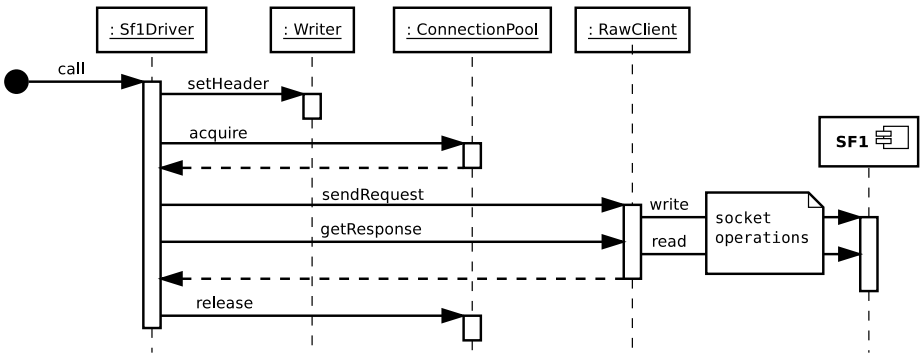


Figure 4: Activity diagram.

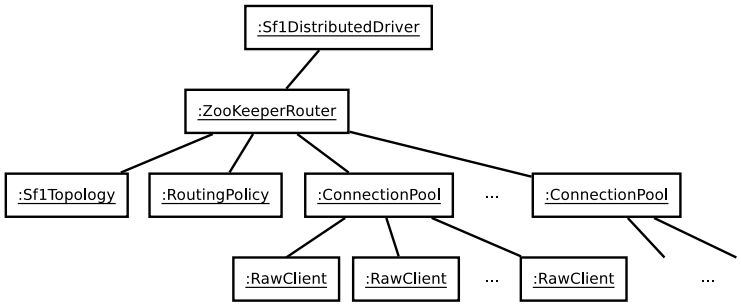


(a) Object diagram

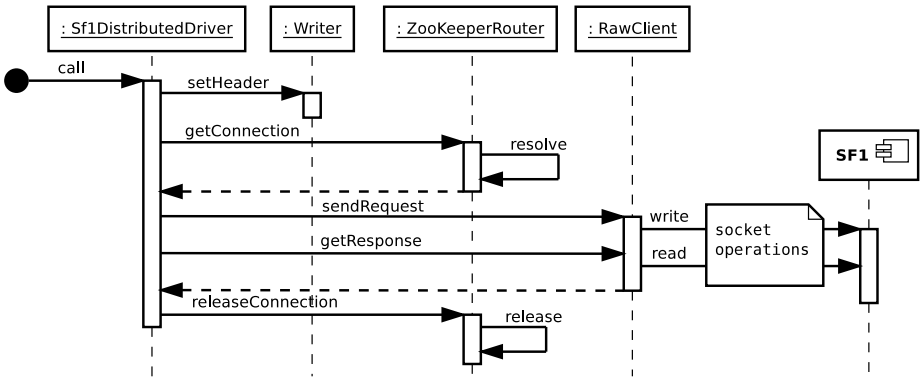


(b) Sequence diagram

Figure 5: Single driver details.



(a) Object diagram



(b) Sequence diagram

Figure 6: Distributed driver details.

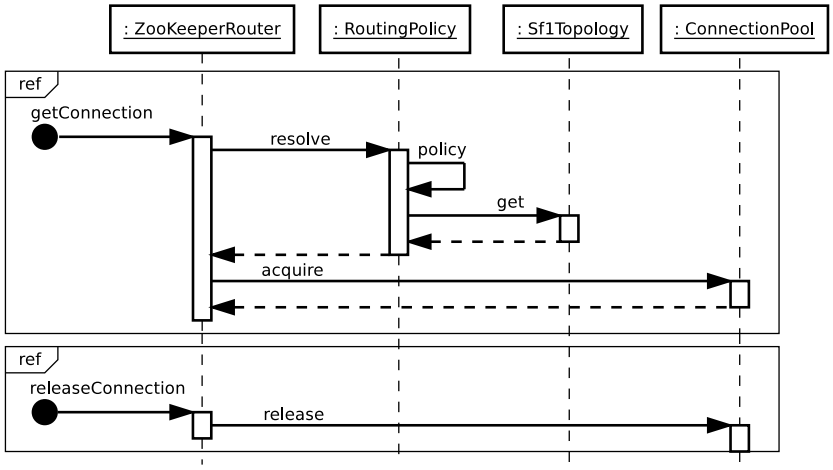


Figure 7: Sequence diagram for the `ZooKeeperRouter`.

3.4 ZooKeeper interaction

Topology

The distributed driver uses ZooKeeper for tracking the actual topology, which is implemented using different containers (see Figure 8 on the following page):

NodeContainer a multi-index container providing two ways of accessing a node:

- by its ZooKeeper path[1]
- by its index number (used for random access and for iterations)

NodeCollectionsContainer a multimap associating a collection and the nodes where it currently is hosted

CollectionsIndex a set of all the collections actually served by the SF1 cluster

All these containers need to be synchronized in order to be consistent with actual topology.

Event handling

On startup, the ZooKeeper client launches two threads, one for the I/O operations with the server, and one for event handling. The watcher registered for event handling will then notify the router for topology changes. The Sf1Watcher responds to the following events:

node creation the new SF1 node is added to the topology and a new connection pool is created

[1]The ZooKeeper hierarchical namespace on which SF1 instance are registered is defined as:

```
/
└─ SF1R-host
   └─ SearchTopology
      └─ ReplicaN [node count]
         └─ NodeM [node data]
```

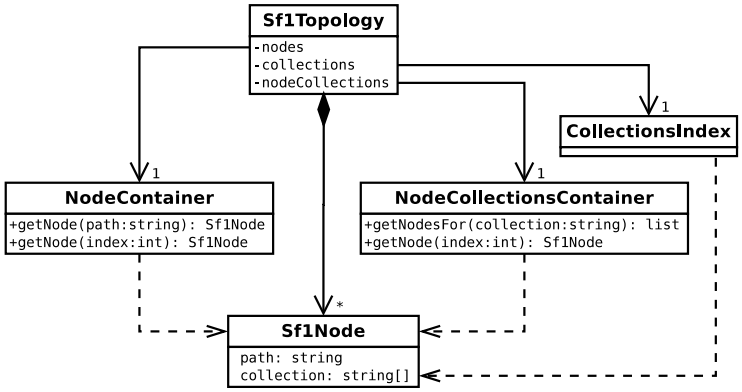


Figure 8: SF1 topology implementation.

node data changed the topology is updated in order to mirror node changes (e.g. the collections hosted)

node deletion the SF1 node is removed from the topology, so that no request will be forwarded to it, and its connection pool is closed

node children changed the hierarchical namespace must be watched in order to correctly detect the topology changes

In Figure 9 on the next page is described how the topology is loaded. Topology changes are monitored in the same way, e.g. when a SF1 has been deleted from the ZooKeeper namespace, its corresponding Sf1Node is removed from topology and its ConnectionPool closed.

It is worth reminding that threads synchronization mechanism are required in order to access resource shared among threads. Please refer to the ZooKeeper documentation for more details.

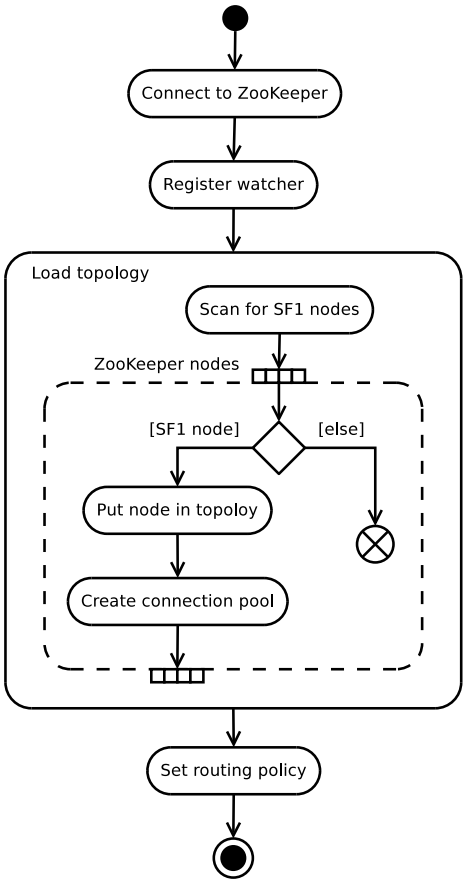


Figure 9: Activity diagram for topology.