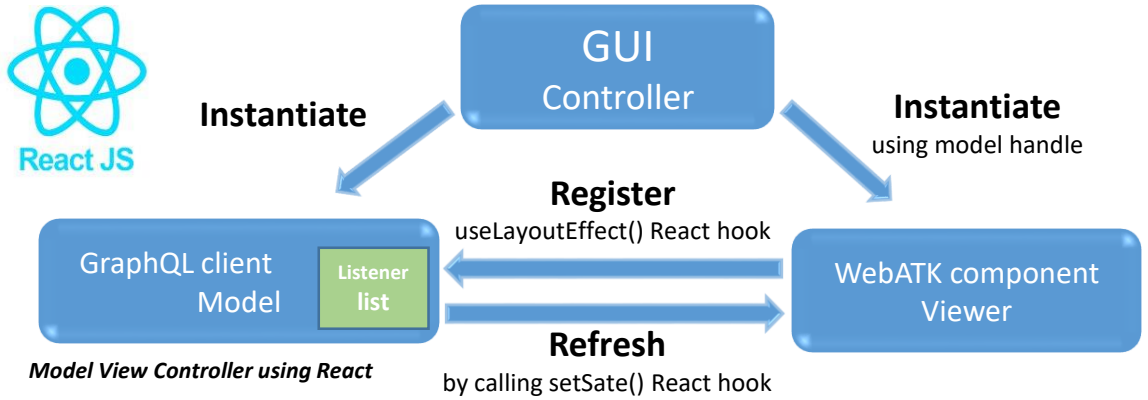
TangoGraphQL: A GraphQL binding for Tango control system Web-based applications

J.L. Pons , European Synchrotron (E.S.R.F), Grenoble, France

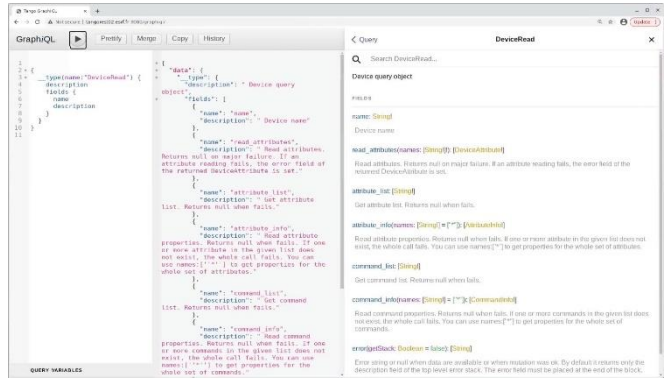
ICALEPCS 2021

Introduction and Challenge



Model View Controller using React

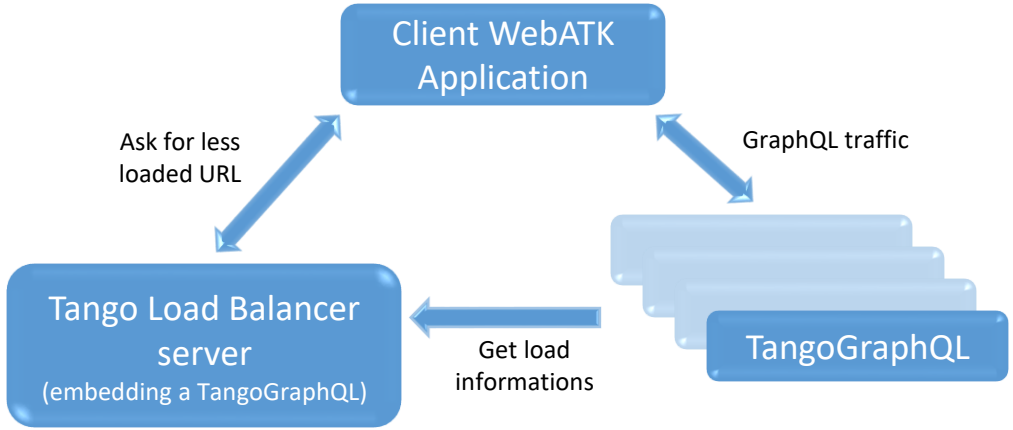
GraphQL



Tango GraphQL


```
subscription {
  subscribe(attNames:"simu/powersupply/1/current",
            modes:PERIODIC) {
    full_name
    index
    event
    value {
      value
      quality
      timestamp
      error
    }
    subscription_error
  }
}
```

TangoGraphQL internal



Load balancing model

Introduction

Web-based applications have seen a huge increase in popularity in recent years, replacing standalone applications. GraphQL provides a complete and understandable description of the data exchange between client browsers and back-end servers. GraphQL is a powerful query language allowing API to evolve easily and to query only what is needed. GraphQL also offers a WebSocket based protocol which perfectly fit to the Tango event system. Lots of popular tools around GraphQL offer very convenient way to browse and query data.

TangoGraphQL C++ server

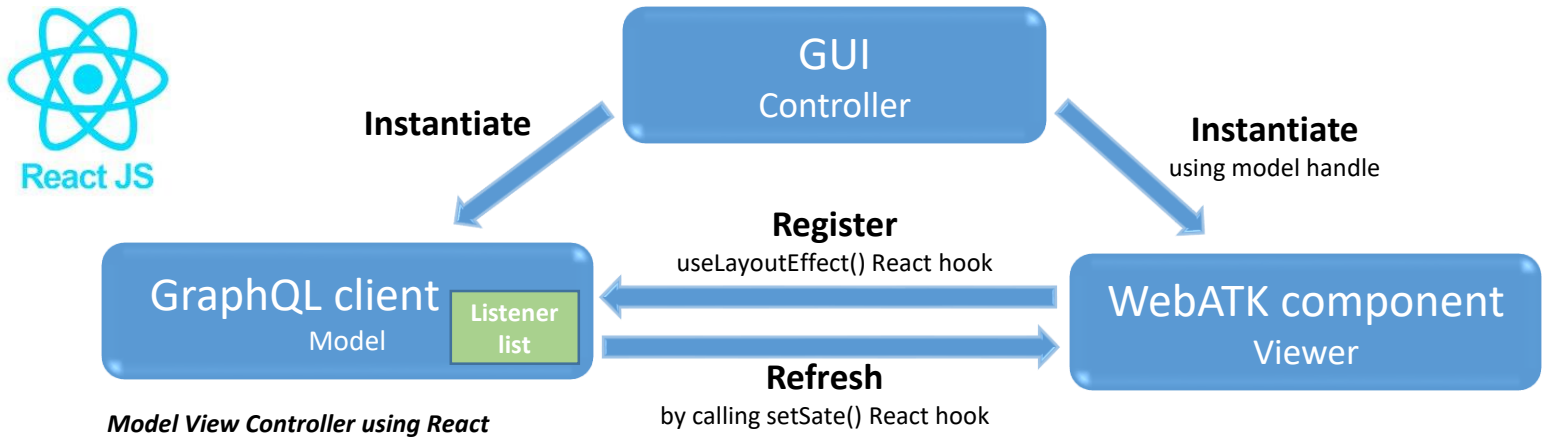
TangoGraphQL is a pure C++ http(s) server which exports a GraphQL binding for the Tango C++ API. TangoGraphQL also exports a GraphiQL web application which allows to have a nice interactive description of the API and to test queries. TangoGraphQL has been designed with the aim to maximize performances of JSON data serialization, a binary transfer mode is also foreseen

Challenge

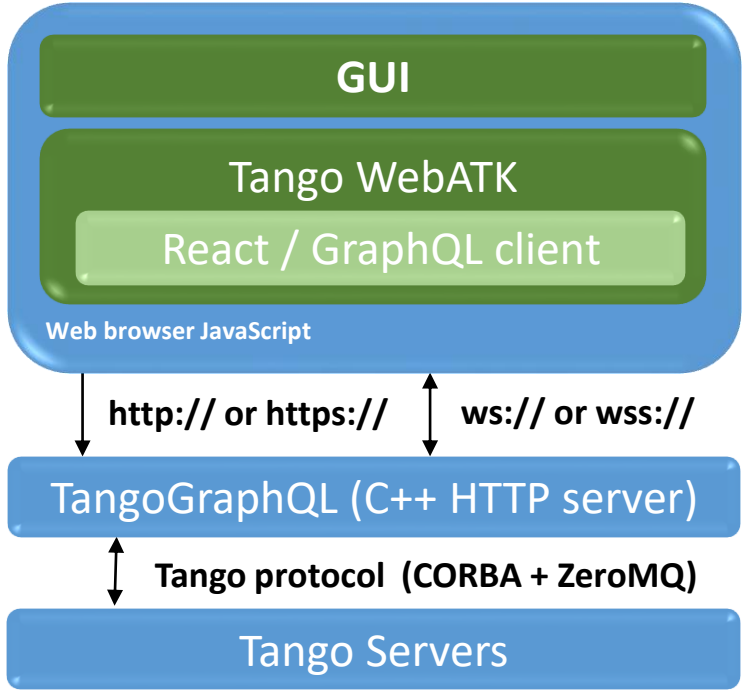
Today, at the ESRF, we use mainly Java standalone applications for the accelerator control system. These applications are built on top of the Java Swing ATK framework and Tango java APIs. Today, regarding GUI technologies, we see almost only development around web technologies such as React, Angular, Vue.js, Bootstrap, Material UI, etc... It is natural that we migrate our GUI to web based applications.

React / GraphQL / MVC

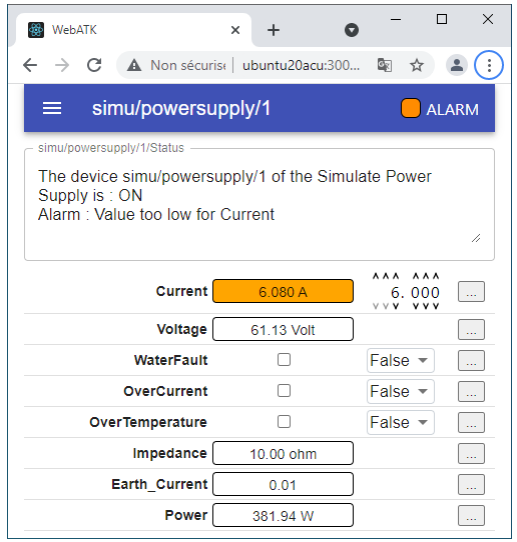
Java ATK is based on the Model View Controller model. React (Facebook) offers a very convenient way to implement this model using hooks. GraphQL , initially developed by Facebook in 2012, was moved as open source to the GraphQL foundation. A JavaScript Tango Web ATK built on top of React and GraphQL is currently under development. This framework is designed in order to ease as much as possible the migration of our Java applications.



Model View Controller using React



Architecture



Application example

This WebAtkPanel uses GraphQL API to make introspection of the Tango server. It uses **Material UI** graphical components.

GraphQL

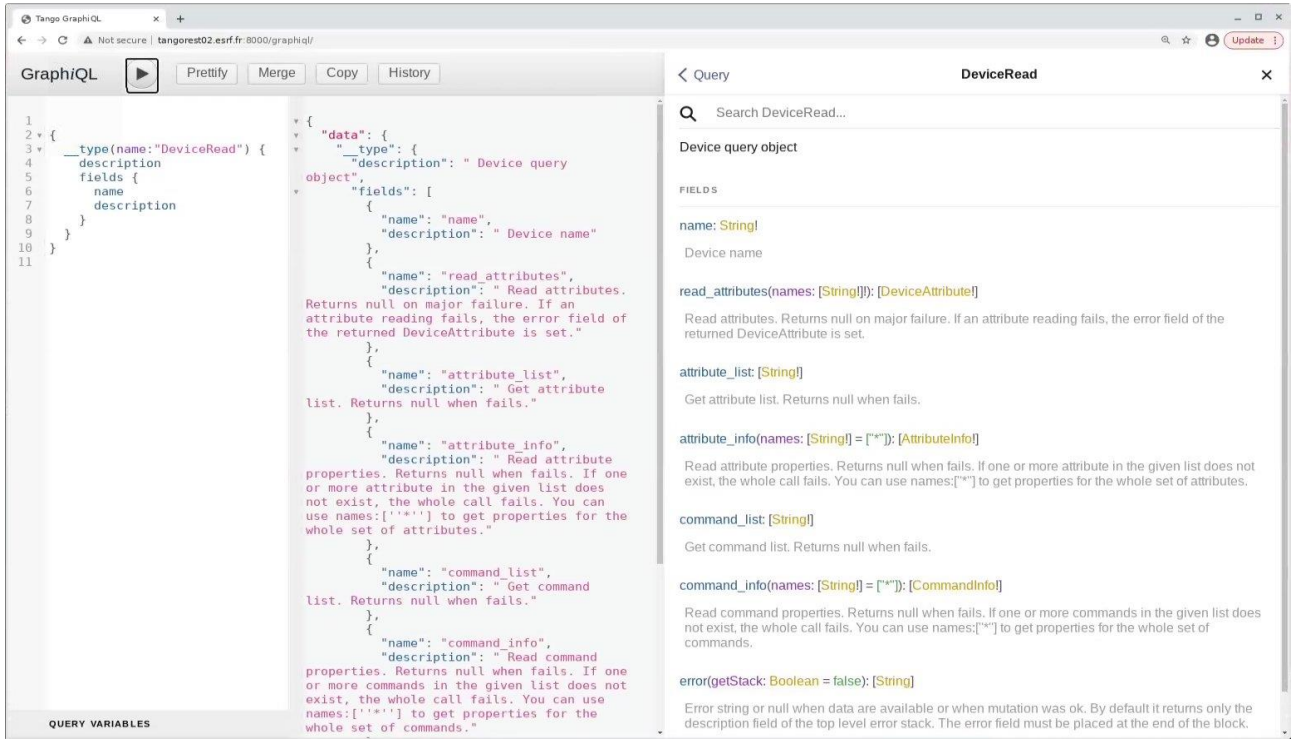
Introspection mechanism

GraphQL provides an introspection system which allows to query information about the supported schema. GraphQL uses GraphQL query to perform introspection. **GraphiQL** is a web application based on this introspection system which allow to write query using modern tools such as completion, syntax checking and documentation browsing. It is an official project of the GraphQL Foundation. It uses the Tango GraphQL schema definition to provides all these features to the TangoGraphQL API . TangoGraphQL C++ server also exports a GraphiQL interface.

GraphQL in JavaScript

On the frontend side, GraphQL queries and mutations are classic HTTP requests so they can be made using the native browser FetchAPI for instance.

```
query=`
{
  device(name:"simu/powersupply/1") {
    read_attributes(names:["state","current","voltage"]) {
      value(resolveEnum:true)
    }
  }
}`
let response: Response = await fetch("http://debian9acu:8000/graphql", {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'Accept': 'application/json',
    },
    body: JSON.stringify({ query: query })
});
let json;
try {
  json = await response.json();
} catch (error) {
  throw new Error(response.status + " " + response.statusText);
}
console.log("State=" + json.data.device.read_attributes[0].value);
console.log("Current=" + json.data.device.read_attributes[1].value);
console.log("Voltage=" + json.data.device.read_attributes[2].value);
```



GraphiQL executing an introspection request

GraphQL over WebSocket in JavaScript

The GraphQL foundation offers a WebSocket JS client API embedded in a js module called graphql-ws. It provides 3 asynchronous callbacks to handle incoming events.

```
import { createClient } from 'graphql-ws';
const client = createClient({url: "ws://tangorest01.esrf.fr:8000/graphql-ws"});
async () => {
  const onNext = (value: any) => { /* handle incoming values */ };
  const onError = (error: any) => { /* handle error */ };
  const onResolve = () => { /* handle termination */ };
  await new Promise(() => {
    unsubscribe = client.subscribe(
      { query: 'subscription { my GraphQL subscription request }', },
      {
        next: onNext, error: onError, complete: onResolve,
      },
    );
  });
})();
```

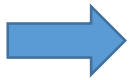
Tango GraphQL

GraphQL Request

GraphQL defines 3 types of request: **query**, **mutation** and **subscription**. Queries are read only requests (basically a read_attribues call), mutations are write requests (write_attributes or command_inout) and subscriptions are used to register to Tango event. The requests are sent to the server via a HTTP POST request.

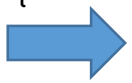
When registering to an event, the TangoGraphQL server will push JSON frames when a Tango event is sent by the Tango server. When using the GraphQL over WebSocket protocol, the client (and the server) use the socket in a bidirectional mode (full duplex), which means that the client (or the web server) can send or receive requests at any time.

```
{
  device(name:"simu/powersupply/1") {
    read_attributes(names:["Current","Voltage"]) {
      value
      write_value
      quality
    }
    error
  }
}
```



```
{
  "data": {
    "device": {
      "read_attributes": [
        {
          "value": 0,
          "write_value": 0,
          "quality": "ATTR_ALARM"
        },
        {
          "value": 0,
          "write_value": null,
          "quality": "ATTR_VALID"
        }
      ],
      "error": null
    }
  }
}
```


```
mutation {
  device(name:"simu/powersupply/1") {
    write_attribute(name:"Current",value:10) {
      error
    }
    error
  }
}
```



```
{
  "data": {
    "device": {
      "write_attribute": {
        "error": null
      },
      "error": null
    }
  }
}
```

```
subscription {
  subscribe(attNames:"simu/powersupply/1/current",
            modes:PERIODIC) {
    full_name
    index
    event
    value {
      value
      quality
      timestamp
      error
    }
    subscription_error
  }
}
```

```
{
  "full_name": "simu/powersupply/1/current",
  "index": 0,
  "event": "periodic",
  "value": {
    "value": 0,
    "quality": "ATTR_ALARM",
    "timestamp": "1633601964.304923",
    "error": null
  },
  "subscription_error": null
}
```


```
# Subscription type contains all subscriptions.
# GraphQL WS protocol create one WebSocket per subscribe request.
type Subscription {
  # Main event subscription request.
  # The attNames array and the modes array must have the same length.
  # When one or more subscriptions fail, the subscription_error field of the returned
  # DeviceAttributeFrame is set.
  subscribe(attNames:[String!]!,modes:[EventSubscriptionMode!]!):AttributeFrame
}
```

Part of the Tango GraphQL schema

Error Management

The GraphQL error management standard is not very convenient. When a node cannot be returned, the GraphQL standard impose that the node must be null and that an additional errors JSON block has to be filled with errors. With large queries which may contain lots of errors, the browsing of this additional error tree is a bit heavy. TangoGraphQL server handles errors as query for best performance.

TangoGraphQL internal

TCP connection

TangoGraphQL server uses multi threaded connection handling and is **HTTP 1.1** compliant. It uses the fact that in HTTP 1.1 the TCP connection can be persistent (keep-alive). The server keeps a **"device factory"** during all the TCP connection life cycle. This prevents from reimporting each time Tango devices and avoid Tango database overload. When the connection is closed, TangoGraphQL releases all allocated resource associated to the connection.

Performance

GraphQL internal parser has been written from scratch and designed to add a true **binary JSON transfer**. Binary transfer is not a part of the GraphQL standard but it can be added without breaking the GraphQL compatibility by using HTTP header. Despite the fact that TangoGraphQL uses fast floating point serialization algorithm Grisu2 from Florian Loitsch (~2 times faster than standard c++ formatting), image and spectrum need to be transferred in binary format in order to reach good performance at both server and client side. On the client side, binary transfer can be easily achieved using native DataViews and ArrayBuffer.

double_spectrum: 16383 random values
16 digits, TEXT mode
Get HTTP Request :1 ms
Reading Tango device:0.9 ms
Encoding JSON:6 ms (using Grisu2)
Send HTTP Response :0.4 ms

double_spectrum: 16383 random values
64bits double, BIN mode
Get HTTP Request :0.9 ms
Reading Tango device:0.9 ms
Encoding BINData:0.3 ms
Send HTTP Response :0.3 ms

TangoGraphQL server timing

```
try {
  arrayBuffer = await response.arrayBuffer();
} catch (error) {
  // Handle error
}
let dv = new DataView(arrayBuffer);
console.log("buff size="+arrayBuffer.byteLength);
var sum = 0;
for(let i=0;i<arrayBuffer.byteLength;i+=8) {
  sum += dv.getFloat64(i, true); // Little endian
}
```

Authentication and Access Control

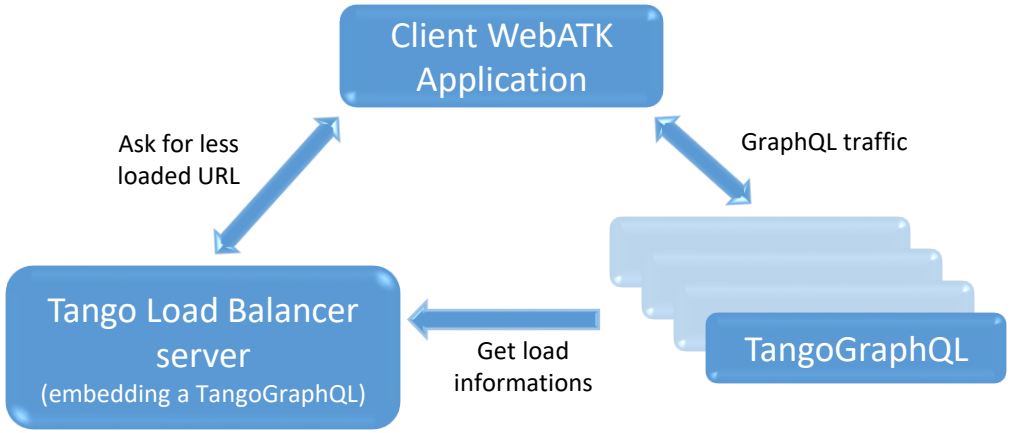
TangoGraphQL uses **Open SSL** for https with basic HTTP authentication scheme. Single Sign-On authentication scheme is currently under development. Tango Access Control cannot be implemented using Tango 9 as it is not possible to set a username for a Tango client thread. This will require new feature of Tango 10.

Asynchronous Group Calls

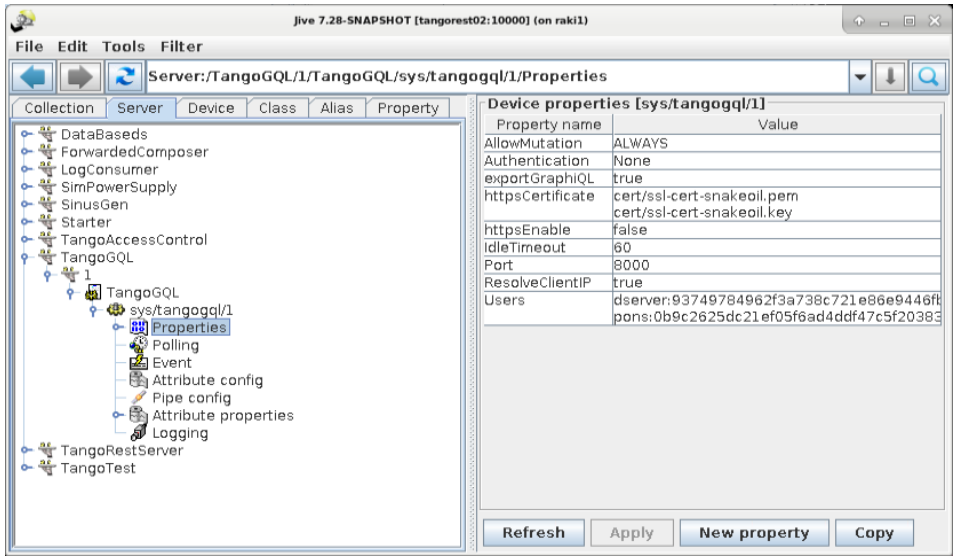
When the client needs to retrieve data from several devices (typically the state of n devices) , the client can construct a GraphQL request using labeling. The server can detect labeled read_attributes calls and make parallel asynchronous calls using Tango Group calls.

Load balancing

TangoGraphQL is also a **Tango server** and can be configured, monitored or instantiated using standard Tango tools. It also has attributes that indicates number of connected clients, number and type of connections, network transfer, etc… The client can use this information to select the less loaded instance among a set of TangoGraphQL server.



Load balancing model



TangoGraphQL server configuration using Jive