Engineering APIs for Accelerator Controls Software

Bartek Urbaniec BE-CSS-CSA (with special thanks to Anti Asko and Lukasz Burdzanowski)

02/12/2021

https://indico.cern.ch/event/1054892
**Agenda**

- Introduction to APIs
- APIs in the Accelerator Control System
- APIs in more depth – what, why and how?
- Use-case study: CCDA, the Controls Configuration Data API
- Practical APIs: security, monitoring, alerting, tracing
- Operational experiences: performance, reliability, availability, testing, to cache or not to cache,…
- Outlook for Controls APIs
Introduction to APIs
An API is a connection between software components. It is a type of software interface, offering a service to other pieces of software. [wikipedia]

APIs can be implemented using a variety of technologies e.g. RMI, WinCC OA, CORBA, OPC-UA, REST, SOAP, gRPC

In the presentation I will mainly refer to REST APIs implemented in Java and used in the Accelerator Control System
The main purpose of the Controls Configuration Service (CCS) is to unite and centralize all the information relevant to the Control systems (CS) in such a way that integration between various Control sub-systems is consistent and efficient.
Controls REST-based APIs in practice

Controls APIs are sets of defined rules that describe how software communicates with each another. The API is an intermediary layer, between a Controls service (Server) and its Clients to exchange data & commands.

Base client-server workflow:
1. A client initiates an API call - request
2. If request is valid, the API makes a call to external program - server
3. The server sends response to the API
4. The API transfers data to the client

The API is a contract between server and client.
Why we need an API

- To expose system data and functionalities to clients
- To allow the clients to program specific complex use cases → “script” the service
- To enable and to improve integration between systems and services → “translate” data structure/etc.
- Internally, API can increase quality of code (and product) by breaking large monoliths into smaller functional services → low coupling, high-cohesion – OOD principles

Typing “CERN” in YouTube search will execute a query like below

```
YouTube.Search.list('id,snippet', {q: 'CERN', maxResults: 25});
```

OBD2 (On-board diagnostics) interface which allows to access car data. i.e. PID 0D – car speed
Kinds of APIs

By Usage

- **Public API**
  open to *all* clients – any client (authenticated/authorised) may use it

- **Partner API**
  dedicated for *agreed* clients – available only to some clients, often via dedicated gateways

- **Internal API**
  not for *external* clients – inside internal network or between internal processes

- **Composite API (Proxy API)**
  used to combine several APIs into one

```json
POST /order
{
  "order-request": [
    {
      "path": "/client", "ref": "client", "body": {"name": "Bartek"}
    },
    {
      "path": "/order",
      "body": {
        "customer": "+@client.id", order: {"name": "My new book"})
    }
  ]
}
```
Typical kinds of APIs for the Web

**By Technology**

- **RPC** (Remote Procedure Call)
  request-response protocol (XML-RPC, JSON-RPC)

- **SOAP** (Simple Object Access Protocol)
  messaging protocol with XML as exchange format

- **REST** (REpresentational State Transfer)
  set of architectural constraints, not a protocol or a standard.
  - Client/server architecture
  - Stateless
  - Cacheable
  - Uniform interface
  - Layered system

- **gRPC** (Google)
  based on HTTP2.0, uses Protocol Buffer to serialise data to binary format
  *yes, technically gRPC generates stubs for any client, abstracting HTTP2.0

- **GraphQL** (Facebook)
  based on HTTP, allows clients to structure data

---

**POST /getAccelerator HTTP/1.1**
HOST: ccda
Content-Type: application/json
"name": "LHC"

**JSON-RPC**

```xml
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <m:GetAcceleratorDetails xmlns:m="https://ccda/accelerators">
      <m:Name>LHC</m:Name>
    </m:GetAcceleratorDetails>
  </soap:Body>
</soap:Envelope>
```

**SOAP**

```http
GET /api/core/accelerators/LHC HTTP/2.0
Host: ccda
Connection: keep-alive
Accept: text/html,application/json
Accept-Encoding: gzip, deflate, br
```

**REST**

```http
:method POST
:path /Accelerators/getAccelerator
(encoded message using protocol buffers)
```

**gRPC**

```json
Request:
{
  "accelerator":
  {
    "name": "LHC"
  }
}
```

**GraphQL**

```json
Response:
{
  "accelerator":
  {
    "name": "LHC"
  }
}
```
API Technology Trends

Worldwide interest over last 12months (Source: Google Trends)

- REST
- SOAP
- gRPC
- GraphQL
- RPC
API – Goals & Challenges

- Make it **easy to work** with – should be **intuitive** to the users

- Make it **stable** – limit **breaking changes** to bare minimum
  
  *no changes/API versioning/EOLs*

- Make it **fast and performant**
  
  *number of requests should not significantly impact the API*

- Make it **technology agnostic** when justified – driven by users needs
  
  *API should be accessible from as many technologies (languages) as it is possible*
APIs in practice
CCS APIs – how we do it – real-life examples

Complex DB model

Simple and intuitive model

Language independent

* Controls Device Class – an abstraction of physical equipment or a process/service, used to interact with the machines (acquisition, settings, commands).

Relational DB model of Controls Device Classes *

JSON representation

CCDA object-model (Java)
The classic 3-tier architecture

**Clients**
- Java Client libraries and language independent access for all our users
- OpenFeign
- Ribbon

**System entry-point**
- HTTP routing
  - Spring Gateway
- Eureka

**Server nodes**
- Stand-alone embedded Java HTTP servers
  - Spring Boot

**Server**
- Server-side load balancing and health-checks
  - Eureka

**Load-balancing**

**Database**
- Controls Configuration
- Database
  - Oracle 19C

**Traffic telemetry, metrics, ...**
- ELK (BE-CSS Tracing)

* Fast Application Notification (FAN) is a notification mechanism used to notify other processes about cluster configuration and service-level information.

* Oracle Real Application Clusters (Oracle RAC) runs on Oracle Clusterware, which provides a highly available (HA) application framework.

Yes, We want to containerize server side and horizontally scale it...

---

Java Client libraries and language independent access for all our users

OpenFeign
Ribbon

Client-side load balancing and resiliency

ReST

HTTP routing

Spring Gateway

Eureka

Server-side load balancing and health-checks

Spring Boot

Oracle Connection Pool using FAN* with RAC*

* Oracle Real Application Clusters (Oracle RAC) runs on Oracle Clusterware, which provides a highly available (HA) application framework.

* Fast Application Notification (FAN) is a notification mechanism used to notify other processes about cluster configuration and service-level information.
API in action

Client

```
Accelerator accelerator = 
CedaClient.newInstance().getService(AcceleratorService.class).findByName("CPS");
```

Server

```
public class AcceleratorController {
  private final AcceleratorService acceleratorService;

  public listAccelerators findALL() { return acceleratorService.findALL(); }
  public Accelerator findByName(String name) { return acceleratorService.findByName(name); }
}
```

Database

```
public class AcceleratorRepository extends Repository<AcceleratorEntity, String> {
  Optional<AcceleratorEntity> findByNameIgnoreCase(String name);
}
```
How to protect the API

Authentication verifies client/user identity

Common Authentication Methods:
- password *(what you know)*
- token/digital id card *(what you possess)*
- finger print/face recognition *(what you are)*
- location *(where you are, e.g.: all connections from CCC)*

Authorization determines client/user access to data

Common Authorization Methods:
- RBAC - Role-based access control
  - with a specific role (OP-Expert) you may control accelerator equipment
- ABAC – Attribute-based access control
  - with a specific attributes (time, location, role) you may control accelerator equipment only at a given time and from a given location
Authentication and authorisation basic workflow

Client

Login and password

User token

Authentication and authorization server
SSO Keycloak based on OpenID/SAML and OAuth2

Persisted data

Persistance server
Oracle DB

Requested data

Request + user token
User token as a cookie or request header

Resources server
JAVA REST based Web server protected by Spring Security

Server will authenticate a user if credentials are correct
HTTP 401: Unauthorized incorrect user/password

Server will provide resources if user is authenticated and authorised to obtain a data
HTTP 401: Unauthorized user is not authorized
HTTP 403: Forbidden user doesn't have privileges
Monitoring:

- shows that all process (servers) are up and running
- gives base information about processes condition
Telemetry with HTTP Load balancer based on HAProxy

Free and open source high availability load balancer for TCP and HTTP based applications

On 64-core ARM server HAProxy is able to handle more than 2 million requests per second.

after HAProxy Team analysis
Monitoring, alerting and tracing

Alerting
- notifies us about all anomalies – helps to prevent system downtime
- allows to improve quality of service – problems are fixed before users spot them

monit alert -- Does not exist ACCSOFT- CCS- CCDA-1

<table>
<thead>
<tr>
<th>Monit</th>
<th>cern.ch</th>
</tr>
</thead>
<tbody>
<tr>
<td>controls-configuration-notifications (Automatic notifications from Controls Configuration Service)</td>
<td>Thursday, 18 November 2021 at 10:12</td>
</tr>
</tbody>
</table>

Show Details

Does not exist Service ACCSOFT- CCS- CCDA-1

- Date: Thu, 18 Nov 2021 10:12:15
- Action: restart
- Most: 
- Description: process is not running

Your faithful employee,
Monit

1 alert for job: ACCSOFT- CCS- CCDA-DEV

Instance is DOWN (0 active)

- alert: Instance is DOWN
- expr: up == 0
- for: 1m
- labels: severity: CRITICAL
- annotations:
  - description: '{${labels.instance}} of job {{ ${labels.job} }} has been down for more than 1 minute.'
  - summary: Instance {{ ${labels.instance} }} is down

02 December 2021
Bartek Urbaniec | Engineering APIs for Accelerator Controls Software
Monitoring, alerting and tracing

**Tracing:**

- informs us about usage of our API – who, how, when
- allows us to analyse and discover anomalies of a running system
Tracing - Grafana

- Server errors: 0
- Last 1 minute: 0

Non HTTP200 responses:
- 8K: 2
- 6K: 0
- 4K: 0
- 2K: 0
- 0: 1

Top HTTP Error codes:
- xCccsStatusCode: 1
  - 404: 18646
  - 400: 58
  - 401: 1

CCDA client versions:
- Version: Count
  - unknown: 12948
  - 1.3.18: 9924
  - 1.5.2: 4527
  - 1.5.0: 650
  - 1.5.1: 293
  - 1.4.0: 50

Top repetitive calls:
- Request URI: Count
  - /api/ffes/device-field-values: 7467
  - /api/deviceClasses/BLMDIAMONDVC/versions/1.10.2: 1802
  - /api/deviceClasses/MkipoChannel/versions/1.3.0: 1356
  - /api/deviceClasses/TimingEntDistributor/versions/1.2.0: 869
  - /api/devices/nameOrAlias/SP.BA2.BLMDIAMOND: 773
  - /api/deviceClasses/ALLAcqData/versions/3.2.0: 734
  - /api/deviceClasses/XTIM_SPS/versions/0.4.1: 577
  - /api/devices/ObbBoxBuffer/versions/1.3.0: 577
  - /api/computers/search?query=0&query-type=inv(DIC): 556
  - /api/devices/search?query=deviceClassInfo.name%3D%27ACCELERATOR.INFO%27&page=0: 547

Number of requests per minute:
Availability vs Reliability

**Availability**

The percentage of time when system or service is operational from point of view of its clients

*Highly-available system:* “zero-downtime” operations, including rolling-upgrades

Service is available for 99%

**Reliability**

The probability that system will meet designed performance standards and produce correct output for a specific time

*Mean Time Between Failure (MTBF):* total time in service / number of failures

*Failure Rate (λ):* number of failures / total time in service.

No more than 5 failures per day

{  
"status": 404,
"errorType": "PATH_NOT_FOUND",
"message": "Current path '/api/core/accelerators/LHC' was not found",
"timestamp": "2021-11-19T10:52:00.578311Z"
}
Performance metrics

Computer **performance** is the amount of useful work accomplished by a computer **system**. **Performance** can be estimated in terms of accuracy, efficiency and speed of execution.

We can measure this with various metrics:

- **Technical** (hardware) – health of the system
  Memory, CPU, disk space, number of running processes, networking etc

- **Performance** – Indicator of the problem
  Mean/Max response time, latency, throughput, errors rate

- **Implementation specific** – JVM metrics, Spring Actuator
  GC configurations (type of GC), number of threads (running, waiting, blocked), process memory

* contains bring another layer of HW abstraction, and own metrics
How to verify our system - testing

- Unit, integration, regression tests – availability and reliability

- Stress testing – performance
Improving API throughput

- **Horizontal scaling** adds resources to handle more client requests
- **Client throttling** (e.g. Leaky bucket algorithm) limits misbehaving clients by rejecting requests
- **Circuit breaker** stops requests in case of consecutive errors
- **Caching mechanism** reduces time needed to get data
To cache or not to cache

The premise of a cache - to store and provide already processed data

Benefits:
• limits unnecessary IOs → in many systems, physical IOs are the slowest operations
• load on the server and related services is reduced to minimum → output/data is processed only once

Drawbacks:
• increased complexity of the system (embedded cache vs standalone)
• eviction strategy* – especially difficult for complex system with mutable data
• challenge of consistency in distributed systems – every client should see the same state of cached data → there are solutions: e.g.: distributed cache like Apache Ignite, Redis

Cache eviction strategies:
- by time, e.g.: after n-seconds (TTL)
- by access frequency, e.g.: after 10k cache reads
- on-demand, e.g.: explicit cache purge
- by size, e.g.: max no. of elements in the cache

Caching is not a silver bullet for performance issues

02 December 2021
Bartek Urbaniec | Engineering APIs for Accelerator Controls Software
Summary and outlook for Controls APIs

Designing any API is a real challenge:

- Intuitive, easy to use, without unnecessary complexity, consistent with existing API.
- Requires understanding of current needs and be open to follow the future needs

Development work of hundreds of people is based on Controls APIs

- Services, applications, scripts – all are based on our APIs

As software technology evolves, our APIs must follow, whilst remaining as stable as possible

- Technology obsolesce of Java RMI is reason to renovate to more modern solutions like ReST
- The renovation must limit any negative impact on our existing users, yet it is an opportunity to:
  - Facilitate programming technology-agnostic access for clients using other languages (e.g. Python)
  - Increase consistency across our multitude of client APIs