PortCDM API Introduction

DAT255 / DIT543

Niklas Mellegård

RISE Viktoria

niklas.mellegard@ri.se

The PortCDM Platform

- PortCDM is
  - "*A **concept** for **facilitating** machine to machine **communication**, to enable more **efficient** and **predictable port operations***"

- ***Concept*** – Enable M2M communication;
  - **Why**, for what **purpose**?
  - **How** is the communication envisioned?
- ***Realization***
  - How has the concept been realized in the STM project?

PortCDM is also…
- In heavy development
  - APIs change
- Sometimes buggy!
- Doc. is not complete

Concept: Why, for what purpose?

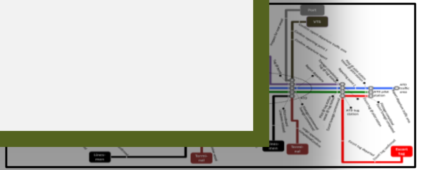
Predictability allows for optimization in several stages

NB. If picking just **ONE aspect** to describe PortCDM, this might be it.

PortCDM, and especially its relation to other STM concepts is of course **much more**

M

Pred
allow
adapt its speed



Concept: Why, for what purpose?

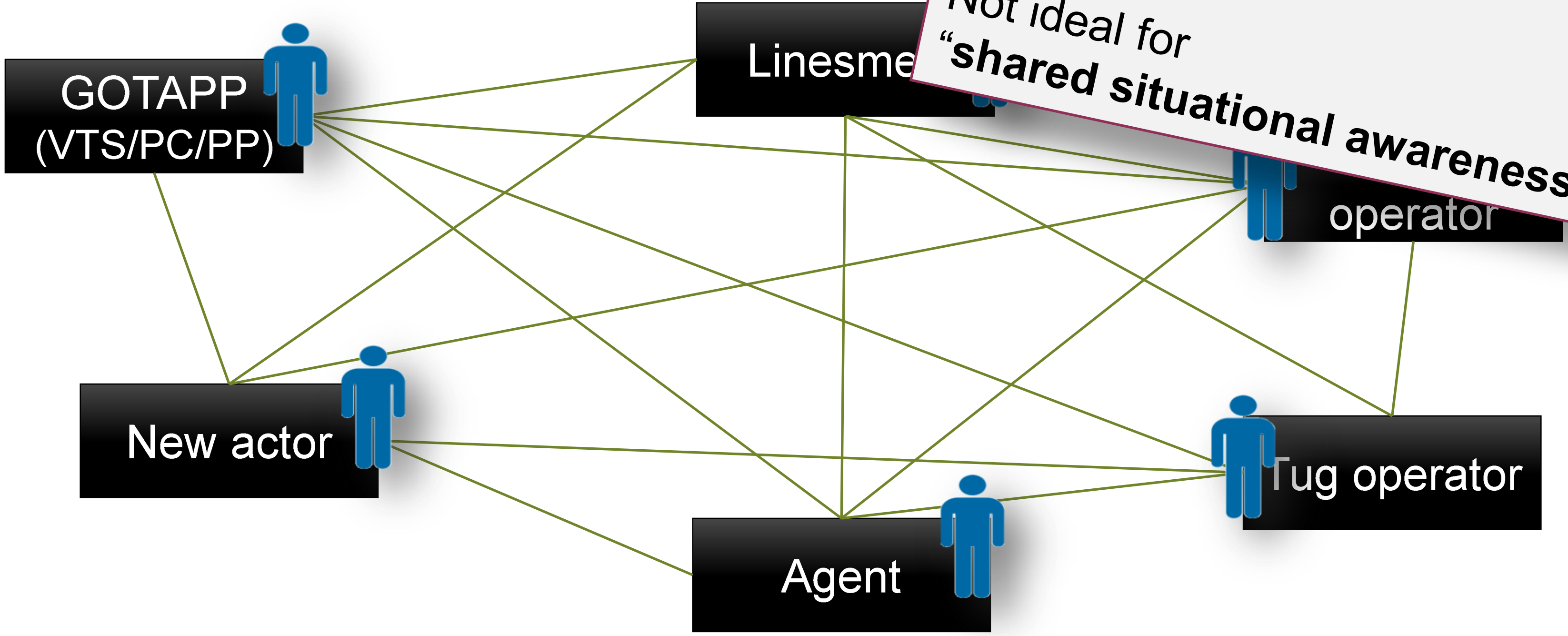
Predictability allows for optimization in several stages

Plan voyage b ed on port readiness

Dyn plan as ailable

Need to “black-box” port operations

Shared situational Awareness
Enabling information sharing

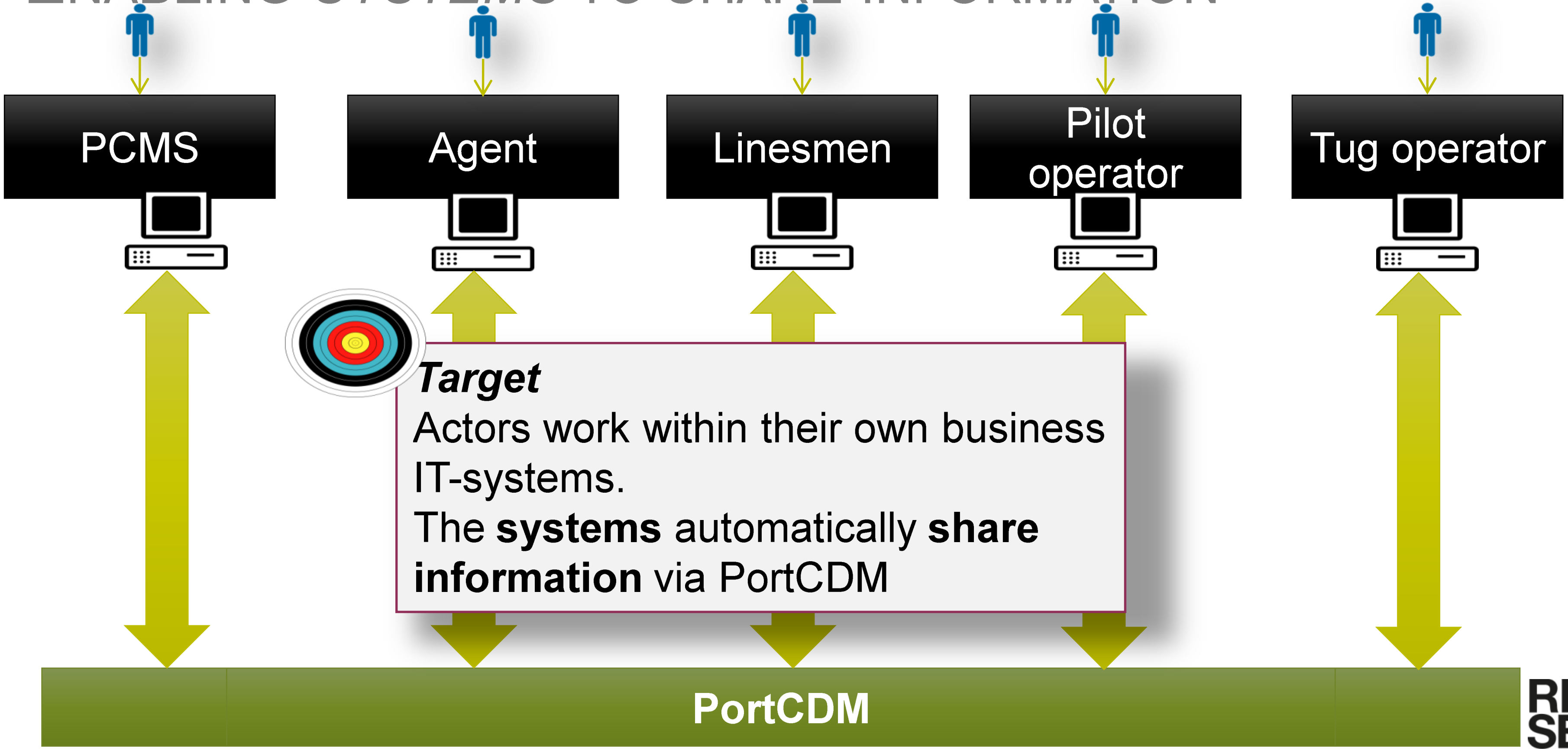


Current state-of-practice
Peer-to-peer communication, "islands" of information.

Not ideal for
"shared situational awareness"

PortCDM

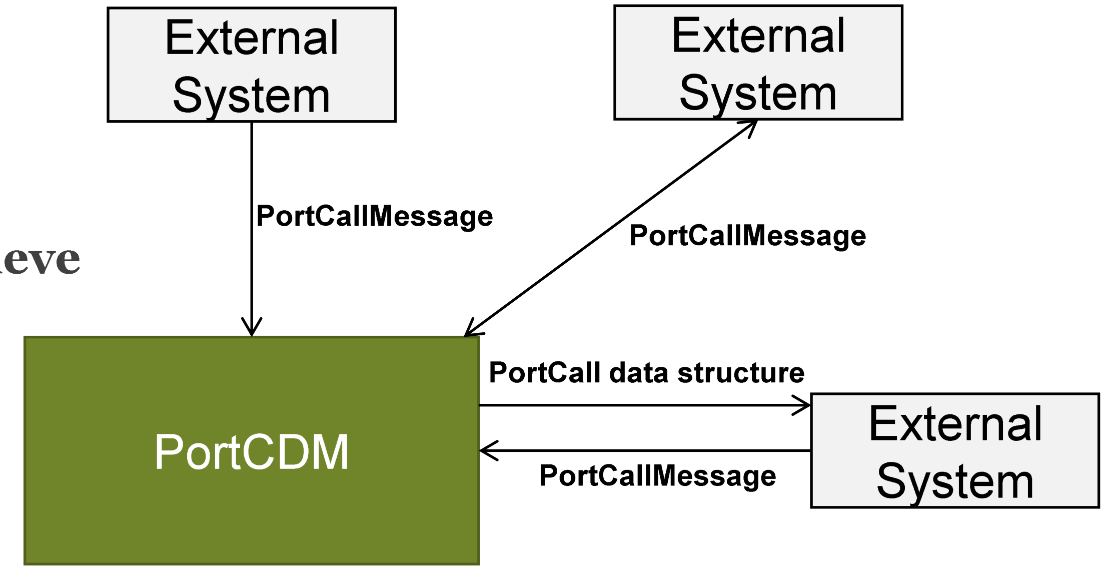
Enabling *systems* to share information



Target
Actors work within their own business IT-systems.
The **systems** automatically **share information** via PortCDM

Concept: How?

- PortCDM is in a sense a **data sharing hub**
  - But, also much more!
- Systems can **submit** and **retrieve** data



What is "data"?
PCM (PortCallMessage)
PortCall data structure

CONCEPT: HOW?
PORTCALLMESSAGE (PCM)

- A PCM is the basic piece of data
  - Represents either:
    a) A **movement** of an object (LocationMessage)
    b) The performance of a **service** (ServiceMessage)
  - Contains
    - Timestatement
    - Time type (Target, Recommended, Estimated, Actual, Cancelled)
- A PCM communicates the progress of an operation (i.e. a **State**)

Concept: How?

PortCallMessage (PCM)

Valid messages are contrained by the StateCatalogue.
API available in "Port CDM Services"
http://specification.portcdm.eu/#/default

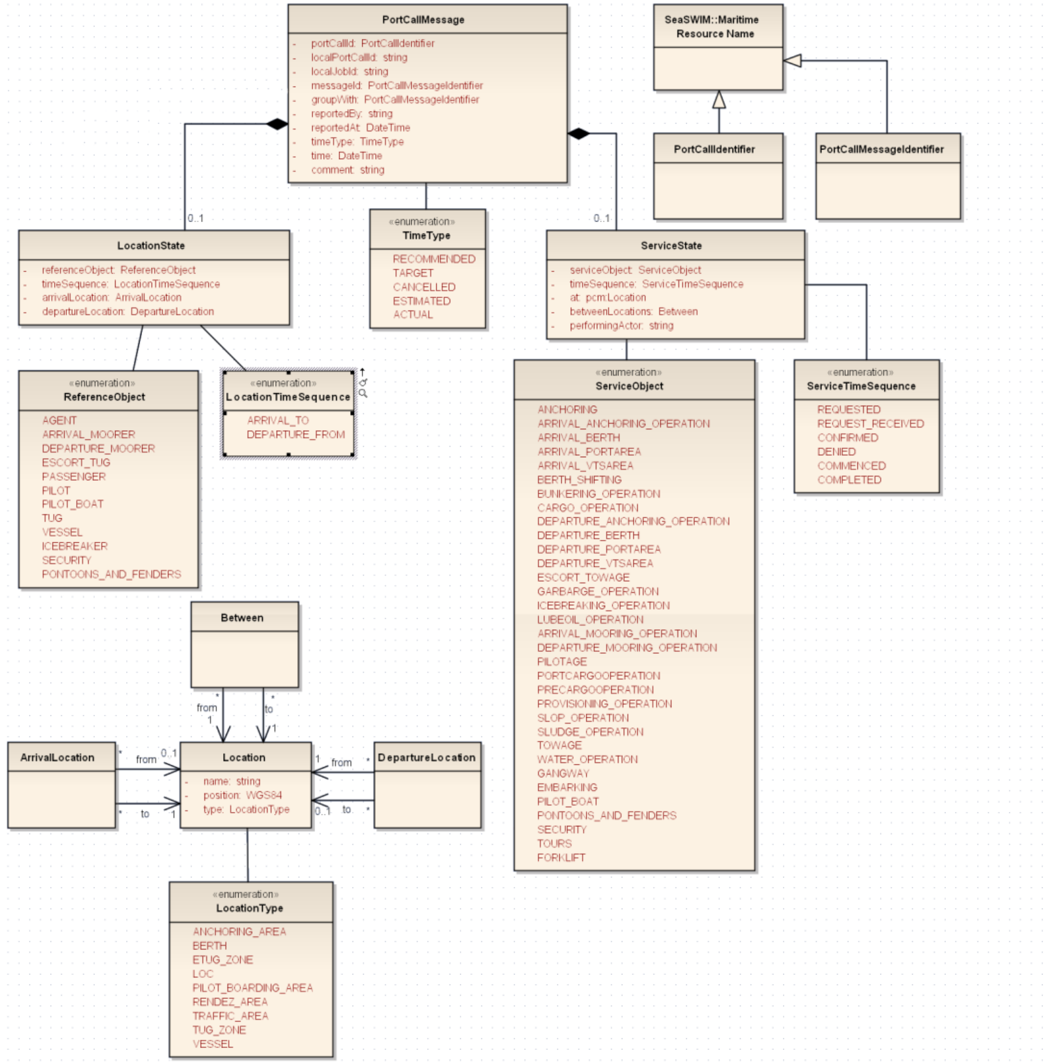
- Examples of data using a PCM
  - "**Estimated** departure of vessel from the port"
  - "**Recommended** arrival of vessel to the port"
  - "**Targeted** commencing of cargo operations"
  - "**Actual** completion of cargo opertion"
  - "**Actual** request for towage operation"
  - "**Estimated** commencing of pilotage operation"

Concept: How?

- Typically,
  - One instance per port, thus
    all messages sent to an instance are implicitly in that port's context
  - Port-to-port communication will be realized as a PortCDM addon (most likely)
- A PCM is (currently) represented in XML

```xml
<?xml version="1.0" encoding="utf-8"?>
<portCallMessage xmlns="http://www.portcdm.eu/PortCallMessage">
  <portCallId>The portCallID</portCallId>
  <messageId>A UUID</messageId>
  <vesselId>A vessel identifier</vesselId>
  <reportedAt>2016-04-13T12:12:12</reportedAt>
  <reportedBy>viktoria</reportedBy>
  <timeType>ESTIMATED</timeType>
  <time>2016-12-12T21:12:12</time>
  <comment></comment>
  <locationState>
    <arrivalLocation>
      <to>
        <locationType>TRAFFIC_AREA</locationType>
        <position>
          <latitude>3.1415926535</latitude>
          <longitude>3.1415926535</longitude>
        </position>
        <name></name>
      </to>
    </arrivalLocation>
    <referenceObject>VESSEL</referenceObject>
  </locationState>
</portCallMessage>
```

Though, typically a PCM is generated using client APIs (more later)



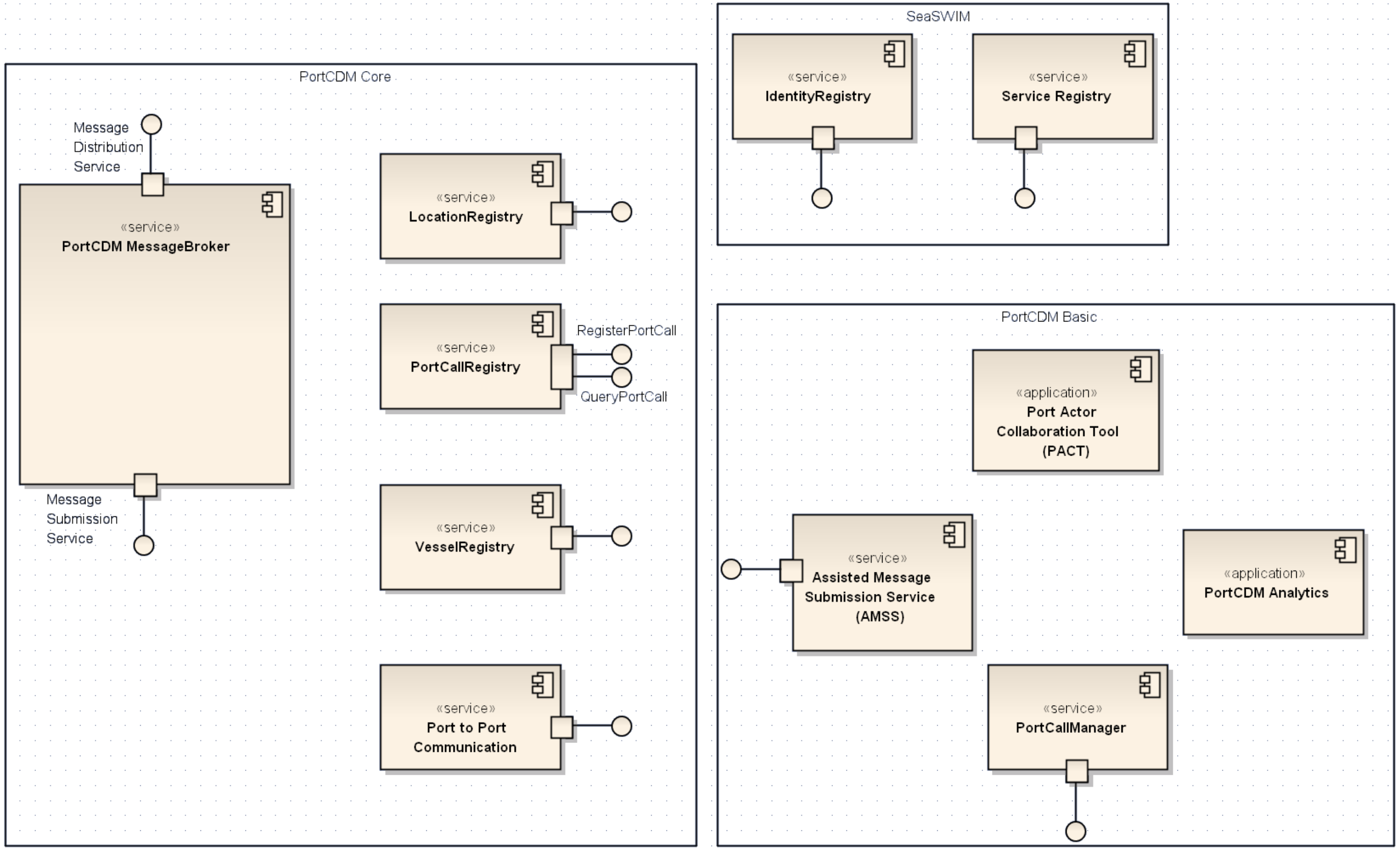
The Structure of a PortCallMessage

- This model is for an older version of the format (0.0.14)
  - Although, the changes done since are minor
- Refer to the specification for up-to-date details

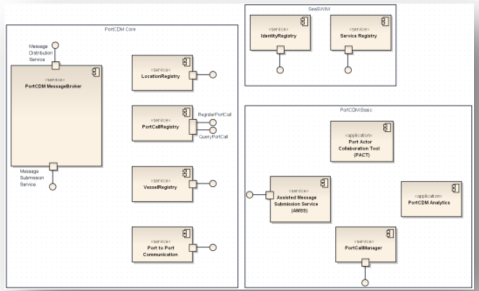
Some notes

- Requires a portCallId
  - To submit a PCM to PortCDM, the id is (typically) mandatory
  - portCallId is generated by PortCDM
    - AMSS supports when id is unknown
- messageId is chosen by client
  - Typically, use a UUID
- vesselId is optional, but recommended to include
- Given a bunch of PCM, **how to assemble** them **into** a coherent, understandable, and **useful structure**?
  - This is a **real** challenge
  - The PortCallManager service does this
    - There are currently two different PortCall data structures provided by the service

PortCDM Service overview

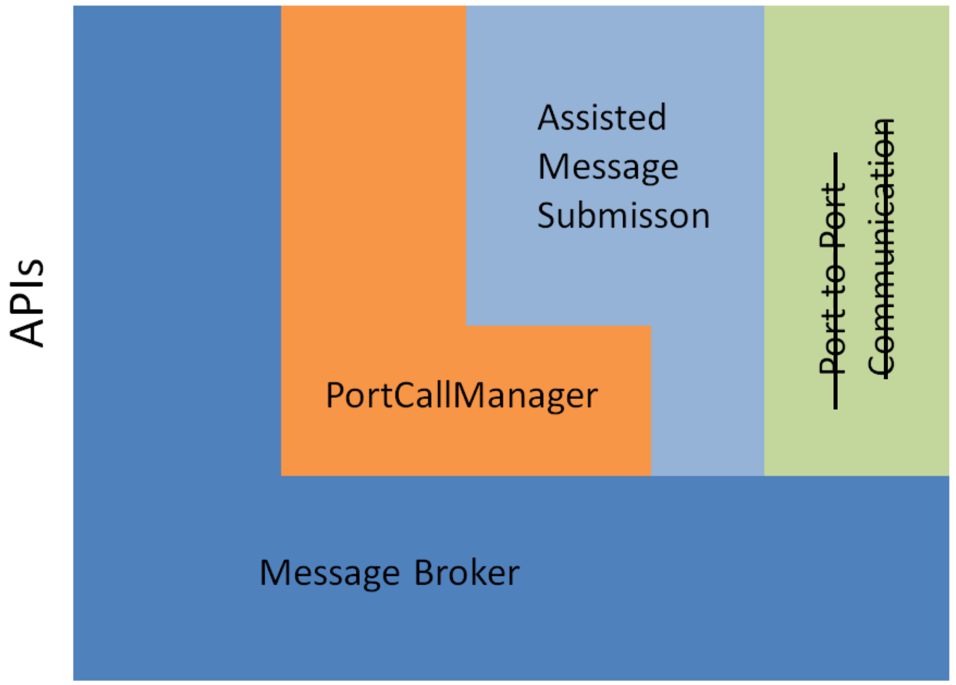


Service overview



- **Message Broker**
  - Submit and subscribe to PCMs
- **LocationRegistry**
  - Provide access to all logical locations in the port
- **VesselRegistry**
  - Provide access to details about vessel
- PortCallRegistry
  - Not implemented
- Port-to-Port
  - Not implemented
- **Assisted Message Submission**
  - Accepts PCM without a portcallId
- **PortCallManager**
  - Provide access to PortCall data structure
- PACT and Analytics
  - Applications using the PortCDM platform

API Overview



- http://specification.portcdm.eu
  - Message broker
    - PortCDM – Message Broker
  - AMSS
    - Assisted Message Submission Service
  - PortCallManager
    - PortCDM Services/port_calls

Message Broker

API Overview

- Message Submission Service
  - For submitting PortCallMessages
- Message Distribution Service
  - For subscribing to PCMs:
    1. Register a message queue (an Id is returned)
    2. Poll the queue to receive all PCMs submitted since queue was created or last polled

Tip: To get older messages, the endpoint "state_update" under "***Port CDM Services***" at http://specification.portcdm.eu/#/default might do just that…

Assisted Message Submission Service

API Overview

- The catch with the Message Submission Service…
  - A valid **portCallId** is required to submit a PCM
    - It is generated by PortCDM when the portcall is created
    - So how to aquire it?
- AMSS
  - Accepts a message without portCallId and does magic
    - VesselId is required here!
- Ok, but how to find the portCallId?
  - Listen to the message queue in the Message Broker
    - Check for messageId
    - Note, it ***may*** take time before message appears…

PortCallManager

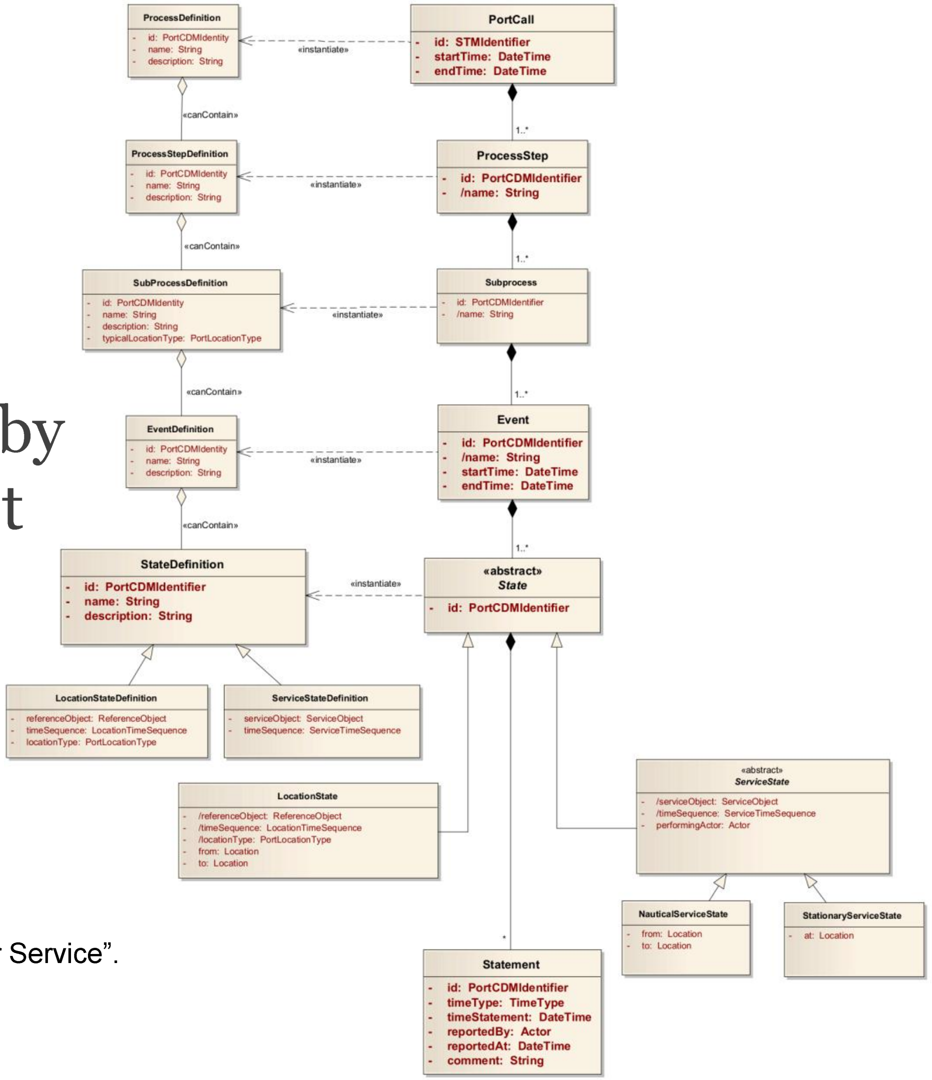
API Overview

- The service aggregates PCMs and…
  - …builds a coherent data structure with all data for a port call
  - …provides nice search functions for port calls
- There are currently two data structures provided
  - They have different purposes
  - One fairly well documented, the other is in development

PORTCALLMANAGER

API OVERVIEW

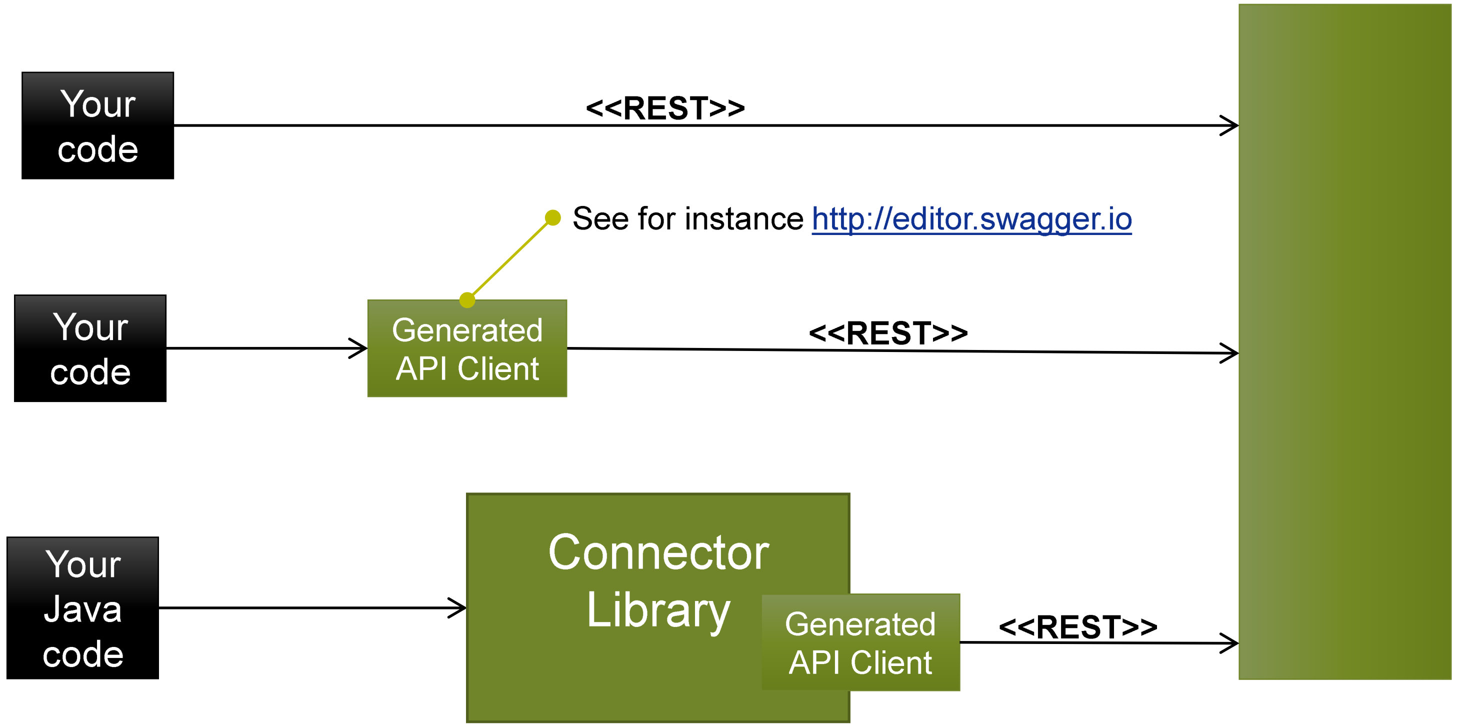
- One of the PortCall data structures
  - A PCM is partly represented by State and partly by Statement



The APIs are listed as "***Port CDM Services***" at http://specification.portcdm.eu/#/default

The other, more experimental service is listed as "Portcall Builder Service". It provides a flatter, less ridgid data structure…

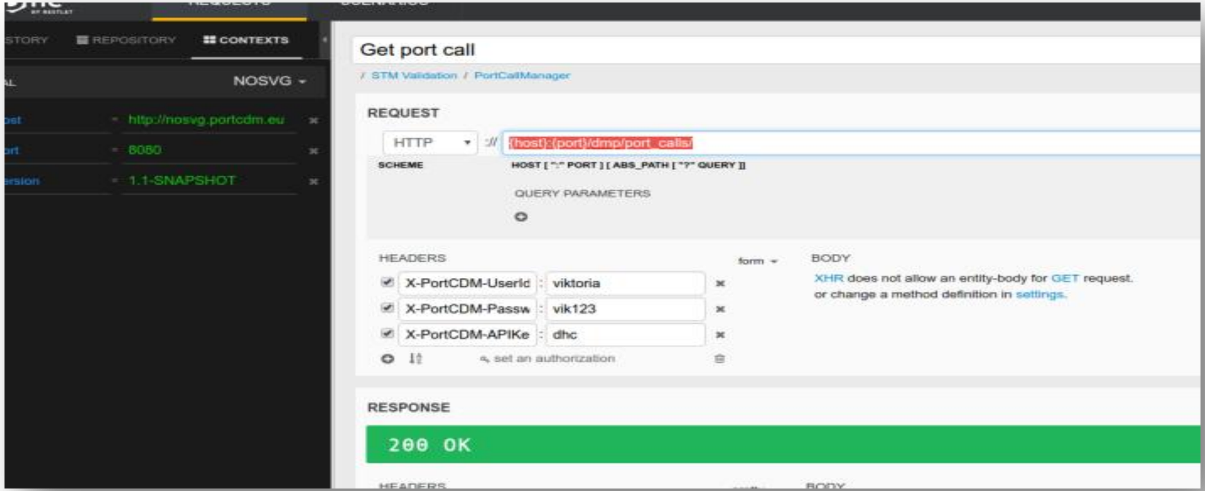
Interating with PortCDM APIs



Barebone REST
Your code
<<REST>>

BAREBONE REST

ACCESSING PORTCDM APIS



STORY | REPOSITORY | CONTEXTS

NOSVG

ost	http://nosvg.portcdm.eu
ort	8080
ersion	1.1-SNAPSHOT

Get port call

/ STM Validation / PortCallManager

REQUEST

HTTP :// {host}:{port}/dmp/port_calls/

SCHEME HOST [":" PORT] [ABS_PATH ["?" QUERY]]

QUERY PARAMETERS

HEADERS

X-PortCDM-UserId	viktoria
X-PortCDM-Passw	vik123
X-PortCDM-APIKe	dhc

set an authorization

form

BODY

XHR does not allow an entity-body for GET request.
or change a method definition in settings.

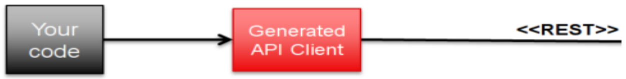
RESPONSE

200 OK

- All APIs have REST interfaces, e.g.
  - Get the 30 most recently updated portcalls
    `<base_url>/dmp/port_calls`
  - Use e.g. the DHC Chrome extension to test!

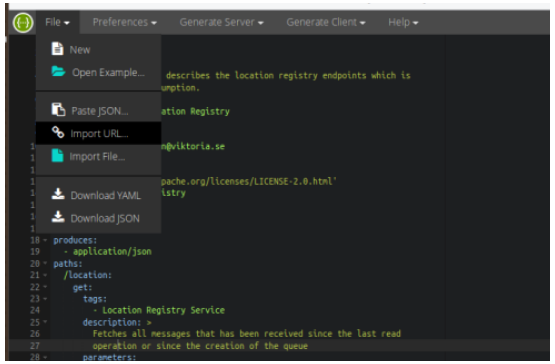
Use DHC to play around with the APIs and see how PortDM responds!

A DHC configuration package is available



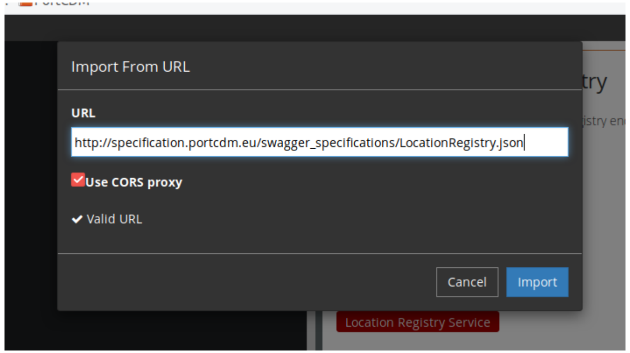
GENERATE CLIENT CODE STUBS

- API Clients can be generated for a number of languages
  - http://editor.swagger.io



```
 describes the location registry endpoints which is
umption.
ation Registry
n@viktoria.se
pache.org/licenses/LICENSE-2.0.html'
istry
produces:
  - application/json
paths:
  /location:
    get:
      tags:
        - Location Registry Service
      description: >
        Fetches all messages that has been received since the last read
        operation or since the creation of the queue
      parameters:
```

1. Choose to import URL



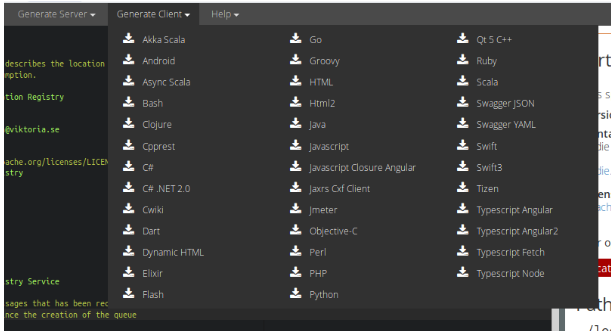
URL

http://specification.portcdm.eu/swagger_specifications/LocationRegistry.json

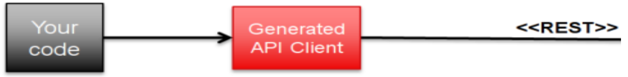
[✓ Use CORS proxy]

✓ Valid URL

2. Provide URL to the service specification



3. Generate client API for your language of choice

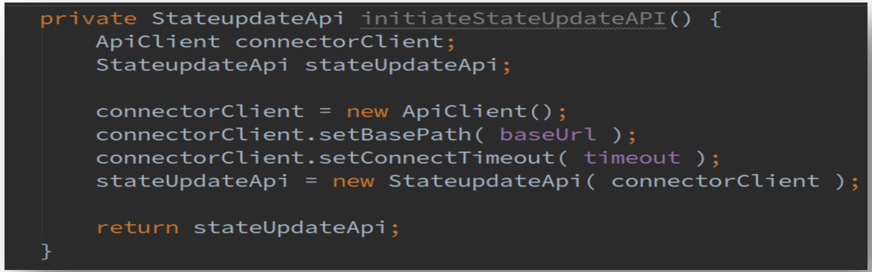


Generated API Clients
Accessing PortCDM APIs

- How to use
  1. Inititate API client
  2. Create PCM
  3. Submit PCM

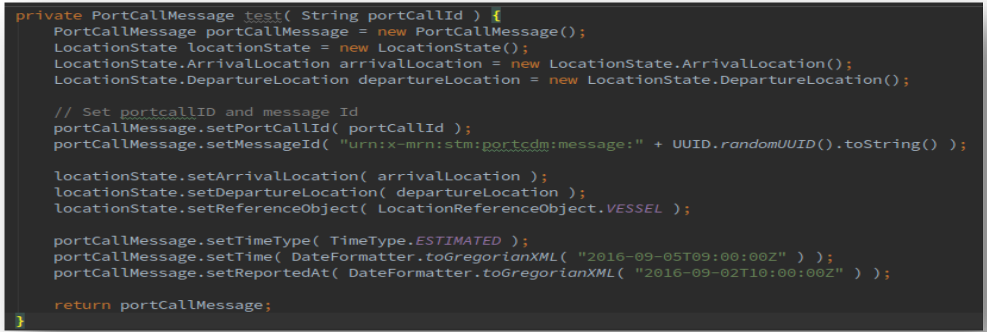
Example shows the Java client

1. Initiate API client



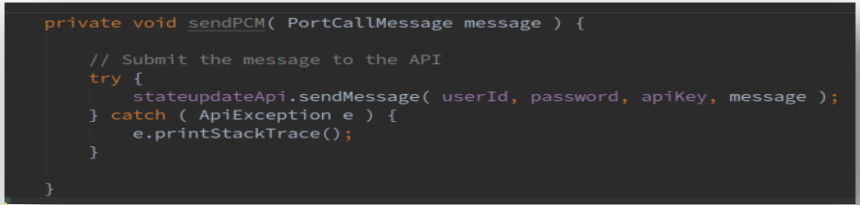
```
private StateupdateApi initiateStateUpdateAPI() {
    ApiClient connectorClient;
    StateupdateApi stateUpdateApi;

    connectorClient = new ApiClient();
    connectorClient.setBasePath( baseUrl );
    connectorClient.setConnectTimeout( timeout );
    stateUpdateApi = new StateupdateApi( connectorClient );

    return stateUpdateApi;
}
```

2. Create a PCM



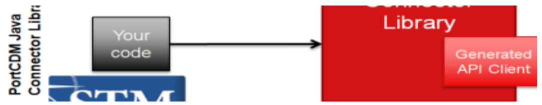
```
private PortCallMessage test( String portCallId ) {
    PortCallMessage portCallMessage = new PortCallMessage();
    LocationState locationState = new LocationState();
    LocationState.ArrivalLocation arrivalLocation = new LocationState.ArrivalLocation();
    LocationState.DepartureLocation departureLocation = new LocationState.DepartureLocation();

    // Set portcallID and message Id
    portCallMessage.setPortCallId( portCallId );
    portCallMessage.setMessageId( "urn:x-mrn:stm:portcdm:message:" + UUID.randomUUID().toString() );

    locationState.setArrivalLocation( arrivalLocation );
    locationState.setDepartureLocation( departureLocation );
    locationState.setReferenceObject( LocationReferenceObject.VESSEL );

    portCallMessage.setTimeType( TimeType.ESTIMATED );
    portCallMessage.setTime( DateFormatter.toGregorianXML( "2016-09-05T09:00:00Z" ) );
    portCallMessage.setReportedAt( DateFormatter.toGregorianXML( "2016-09-02T10:00:00Z" ) );

    return portCallMessage;
}
```

3. Send the PCM



```
private void sendPCM( PortCallMessage message ) {

    // Submit the message to the API
    try {
        stateupdateApi.sendMessage( userId, password, apiKey, message );
    } catch ( ApiException e ) {
        e.printStackTrace();
    }

}
```

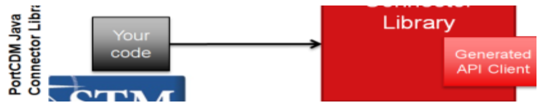
PortCDM Java Connector Lib



- Developed to simplify connectors written in Java
  - Wraps the generated Java client API
  - Provides some protection against API changes

- Features
  - Can multiplex messages to multiple backends
    - E.g. to production and various test PortCDMs
  - Simple file configuration
  - Automatically aquires portCallId
    - Can map to a local jobId if one is available in the external system

The SubmissionService class is your API



IMPORTING CONNECTOR LIBRARY

- Configure Maven to import the connector library

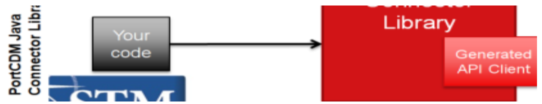
1. Configure maven repositories (settings.xml)

```
<repository>
    <id>monalisa-viktoria-snapshots</id>
    <name>MonaLisa Viktoria Snapshots</name>
    <url>http://brink.viktoria.chalmers.se/nexus/content/repositories/monalisa-viktoria-snapshots</url>
    <releases>
        <enabled>false</enabled>
        <updatePolicy>always</updatePolicy>
        <checksumPolicy>warn</checksumPolicy>
    </releases>
    <snapshots>
        <enabled>true</enabled>
        <updatePolicy>never</updatePolicy>
        <checksumPolicy>fail</checksumPolicy>
    </snapshots>
    <layout>default</layout>
</repository>
<repository>
    <id>monalisa-viktoria</id>
    <name>MonaLisa Viktoria</name>
    <url>http://brink.viktoria.chalmers.se/nexus/content/repositories/monalisa-viktoria</url>
    <releases>
        <enabled>true</enabled>
        <updatePolicy>always</updatePolicy>
        <checksumPolicy>warn</checksumPolicy>
    </releases>
    <snapshots>
        <enabled>false</enabled>
        <updatePolicy>never</updatePolicy>
        <checksumPolicy>fail</checksumPolicy>
    </snapshots>
        <layout>default</layout>
</repository>
```

2. Add dependency to pom.xml

```
<dependency>
  <groupId>se.viktoria.stm.portcdm.demonstrator-connector</groupId>
  <artifactId>common</artifactId>
</dependency>
```

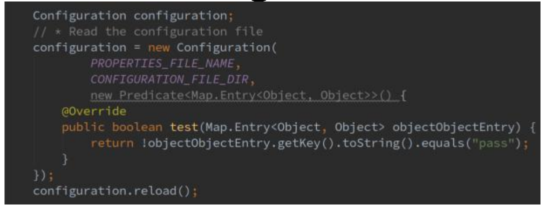
PortCDM Java Connector Lib



Submit a PCM

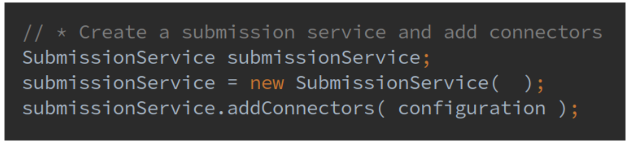
- How to use
  1. Initiate SubmissionService with config file
  2. Create PCMWrapper
  3. Submit

Read the config file



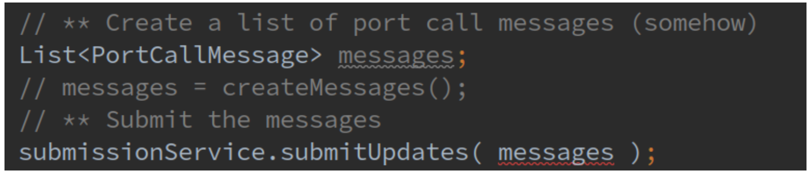
```
Configuration configuration;
// * Read the configuration file
configuration = new Configuration(
        PROPERTIES_FILE_NAME,
        CONFIGURATION_FILE_DIR,
        new Predicate<Map.Entry<Object, Object>>() {
    @Override
    public boolean test(Map.Entry<Object, Object> objectObjectEntry) {
        return !objectObjectEntry.getKey().toString().equals("pass");
    }
});
configuration.reload();
```

Inititate SubmissionService



```
// * Create a submission service and add connectors
SubmissionService submissionService;
submissionService = new SubmissionService(  );
submissionService.addConnectors( configuration );
```

Extract data and submit



```
// ** Create a list of port call messages (somehow)
List<PortCallMessage> messages;
// messages = createMessages();
// ** Submit the messages
submissionService.submitUpdates( messages );
```

Example config file

```
global.stm.activeprefixes =  vm
global.stm.dryrun = false

## VM
vm.stm.host = http://192.168.56.103:8080/dmp
vm.stm.userid = fenix
vm.stm.password = password
vm.stm.apikey=Fenix-SMA
vm.stm.timeout=7000

## DEV
dev.stm.host=http://dev.portcdm.eu:8080/dmp
dev.stm.userid=fenix
dev.stm.password=password
dev.stm.apikey=Fenix-SMA
dev.stm.timeout=20000

# Data source specific settings (excluded)
```

Deployment

Development installation

- Local deployment for testing
    - Each team can deploy any number of “private” instances for testing
        - E.g on localhost
            - IP: 192.168.56.103
    - Distributed as a VirtualBox appliance

Shared installation

- Shared sandbox for collaboration testing and demonstration
    - Hosted on sandboxX.portcdm.eu
    - This will be used by all teams when demonstraing

Tips: Configuration and debugging

- PortCDM logs are available at (assuming local install)
  - /var/log/wildfly
    - Log on to the PortCDM instance
      - E.g ‘`ssh 192.168.56.103`’
        - USER/PASS: pact / pact
- Configuration and administration console
  - Start the service
    - `ssh 192.168.56.103 (pact /pact)`
    - `cd /home/pact/portcdm-administration-console`
    - `nodejs .`
  - Connect to the service
    - From browser: `http://192.168.56.103:1337`

Reference material

- API specifications
  - http://specification.portcdm.eu
  - The raw swagger files (for importing into editor.swagger.io)
    - http://specification.portcdm.eu/swagger_specifications/
- PortCallMessage specification (0.0.16/17 is currently in use)
  - http://specification.portcdm.eu/pcm/
- Message Broker specification
  - Service specification (PDF)
    - Intended reader is implementer of the service (thus much is out of scope)
  - Design Specification (PDF)
    - D.o
- Assisted Message Broker specification
  - Service specification (PDF)

- DHC configuration
  - Easily get configs to test REST calls against PortCDM with the DHC Chrome extension
  - The configuration points to a locally installed VirtualBox deployment (see below)
  - LINK to DHC config
- PortCDM deployment
  - Link to DL (vbox)
    1. Download and install VirtualBox
    2. Configure VB with the appppliance
    3. Your PortCDM is now on 192.168.56.103