Advanced Big Data Mining

05/02/2019
— **Complex Event Processing**

Konstantopoulou Vasiliki
Karidis Athanasios
Mylonaki Aggeliki
Voulgari Eleni

Preliminary Data Analytics

Introduction

With the increased amount of ships sailing around the Earth's oceans and seas, the need for advanced real-time maritime monitoring has emerged. The purpose of such a monitoring is basically to assure safety (vessel collision avoidance), but also to discover important patterns in the contexts of concept, space and time. Such a system can take real-time information and recognize these patterns of vessels in local vicinity. The above patterns may include normal behaviors, as well as illegal and unsafe behaviors (intrusion detection, illegal fishing).

The purpose of the first part of this report is to present the findings of some basic data analysis on given real-world datasets from the maritime domain.

The main content of the datasets is information, obtained through the Automatic Identification System (AIS), which broadcasts position reports and short messages with information about ships and their voyage. Additionally, they contain a set of complementary spatial and temporal data. Four categories of data may be found: Navigation, Vessel-oriented, Geographic and Environmental. The covering area is the Channel and Bay of Biscay (France) in the Celtic Sea and the covering time is October 1st, 2015 to March 31st, 2016.Methods

After being downloaded from the following link: https://zenodo.org/record/1167595, the dataset was loaded in a PostgreSQL database, according to the creator's directions, for further analysis.

Basic and more advanced SQL queries were created and ran in order to provide an overall firstly and a deeper understanding of the involved data secondly.

The need to represent geospatial data led to the use of Postgis, an open-source software that provides support for geographic objects to the PostgreSQL database, and Qgis, an open-source, cross-platform, geographic information system application that supports viewing, editing, and analysis of geospatial data.

Finally, to create charts and diagrams with more comprehensible information, Tableau software, a Business Intelligence Tool for data visualization, was used.

Basic Analysis

Below, the basic SQL queries are being presented with a brief explanation of the outcome and the possible existence of conflicting results.

It is important to learn about the volume of data we have to work with so we started with queries that allow us discover such information.

1) The number of messages contained in the dynamic information of AIS messages.

Query	SELECT COUNT id
	FROM ais_data.dynamic_ships;

Result	19.035.360 messages

2) The number of distinct ships appearing in the table containing the dynamic information of AIS messages.

Query	SELECT DISTINCT mmsi
	FROM ais_data.dynamic_ships;

Result	5055 ships

3) The number of different types of ships.

Query	SELECT DISTINCT(type_name)
	FROM ais_status_codes_types.ship_types;

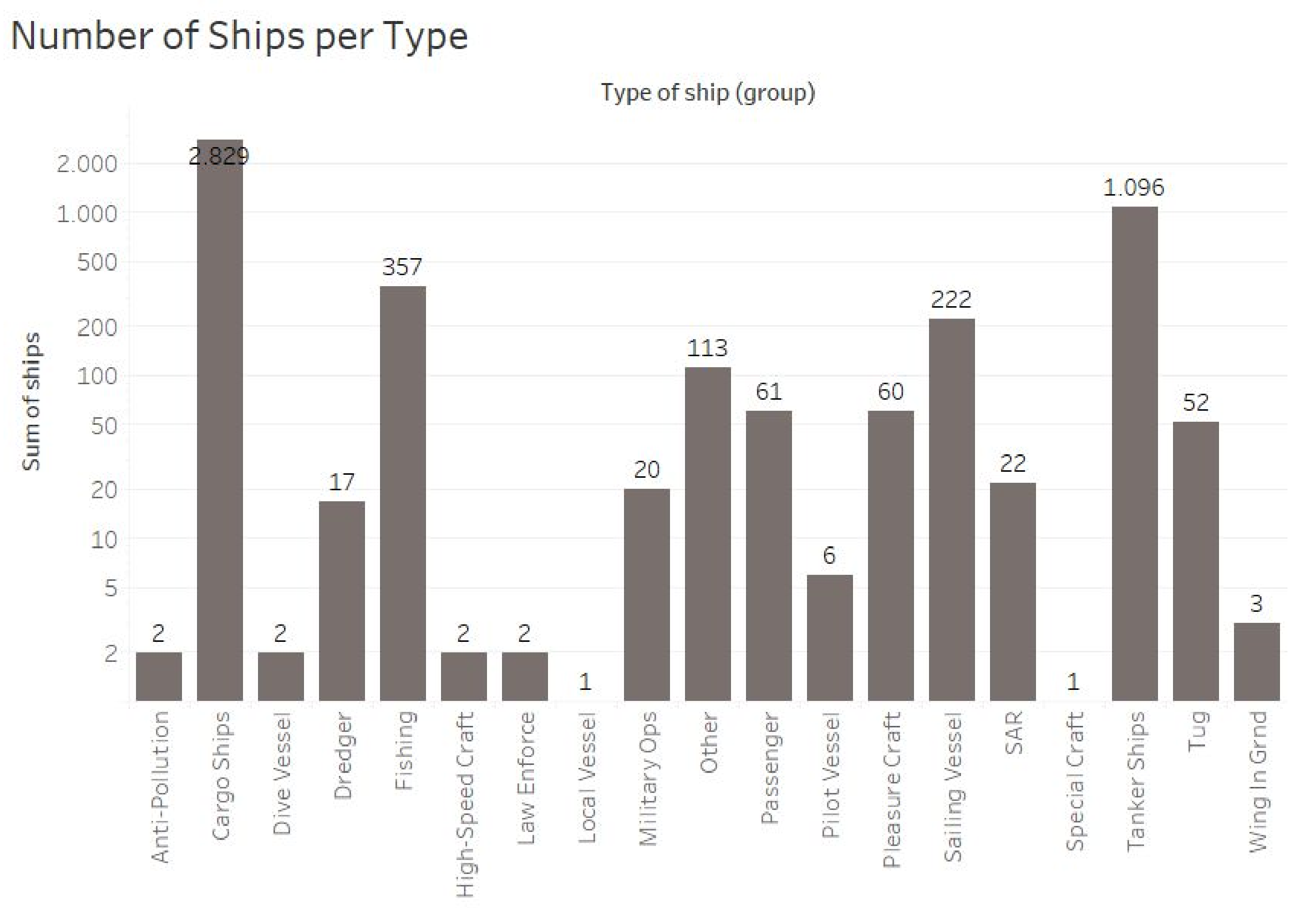
Result	31 types of ships

4) The number of the ships per type, appearing in the table containing the static information of AIS messages.

Query	SELECT COUNT(DISTINCT sourcemmsi), ship_types.type_name
	FROM ais_data.static_ships, ais_status_codes_types.ship_types
	WHERE shiptype between shiptype_min
	            AND shiptype_max
	GROUP BY type_name;

Result	

Count	**Types of Ships**	**Count**	**Types of Ships**	**Count**	**Types of Ships**
2829	Cargo	52	Tug	2	Anti-Pollution
1096	Tanker	22	SAR	2	Dive Vessel
357	Fishing	20	Military Ops	2	High-Speed Craft
222	Sailing Vessel	17	Dredger	2	Law Enforce
113	Other	6	Pilot Vessel	1	Local Vessel
61	Passenger	3	Wing In Grnd	1	Special Craft
60	Pleasure Craft				



Sum of Sum of ships for each Type of ship (group). The marks are labeled by sum of Sum of ships.

As we can observe the total number of unique ships in the ais_data.static_ships is 4.868 whereas the corresponding number of ships in the ais_data.dynamic_ships is 5.055. This might be lost messages because of erroneous transmissions or suspicious vessels that are unwilling to give information about their identity.

Conclusively, we can see that we obtain messages from several type of ships but messages from other types are not offered.

5) The mmsi of one of the two anti-pollution vessels.

Query	SELECT *
	 FROM ais_data.dynamic_ships
	 WHERE mmsi = 228064900

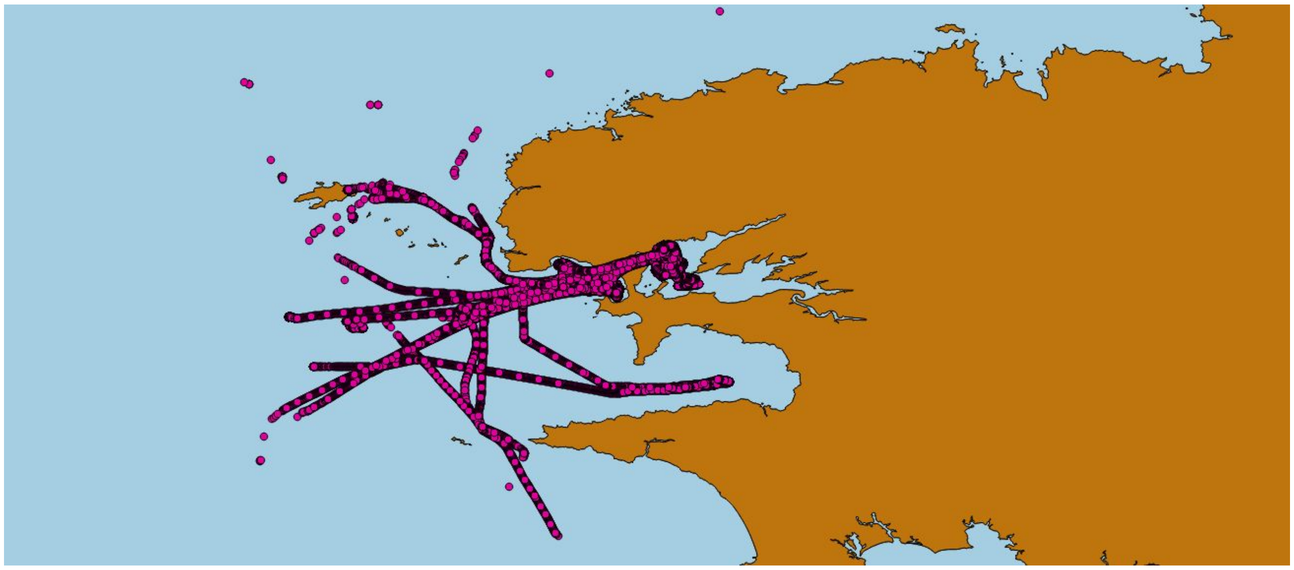


Image1: Trajectory of one of the anti-pollution vessels

Above we can see the trajectory that the anti-pollution ship has followed. As the environmental destruction is an enormous problem nowadays, this type of information could be helpful if we would like to investigate environmental cases.

6) The average, minimum and maximum speed of the moving ships.

Query	SELECT AVG(speed), MAX(speed), MIN(speed)
	FROM ais_data.dynamic_ships
	WHERE speed > 0;

Result	Average speed → 15.79 knots Maximum speed → 102.3 knots Minimum speed → 0.1 knots

Cargo and tanker ships usually travel with a speed between 12 and 19 knots, so the average speed makes sense as 77.65% of the ships fall into these two categories.

The maximum speed found is probably too high for a normal vessel and this suggests that it might be an erroneous entry or noise in the data.

7) The number of vessels that have speed above 60 knots.

Query	SELECT COUNT(DISTINCT mmsi)
	FROM ais_data.dynamic_ships
	WHERE speed > 60;

Result	108 vessels

There are 108 vessels that at some point had speed above 60 knots.

This gives us a hint to further investigate those ships. (what type of ships, erroneous entries, etc)

8) The cargo vessels that do not have a load.

Query	SELECT DISTINCT sourcemmsi, ship_types.type_name FROM ais_data.static_ships, ais_status_codes_types.ship_types WHERE shiptype between shiptype_min AND shiptype_max AND type_name like 'Cargo%' AND draught = 0;
Result	

sourcemmsi	type_name	sourcemmsi	type_name
304346000	Cargo	246625000	Cargo
255801560	Cargo	304010916	Cargo
305166000	Cargo	207138000	Cargo
247056200	Cargo	212058000	Cargo
305335000	Cargo		

As we can observe there are only 9 cargo ships that are not loaded some time in these 6 months according to their static messages.

9) The number of vessels that are heading to Brest port and the number of vessels that are just passing by.

Query	SELECT COUNT(DISTINCT sourcemmsi) FROM ais_data.static_ships WHERE destination LIKE 'BREST%';
Result	194 vessels

Query	SELECT COUNT(DISTINCT sourcemmsi) FROM ais_data.static_ships WHERE destination NOT LIKE 'BREST%';
Result	4431 vessels

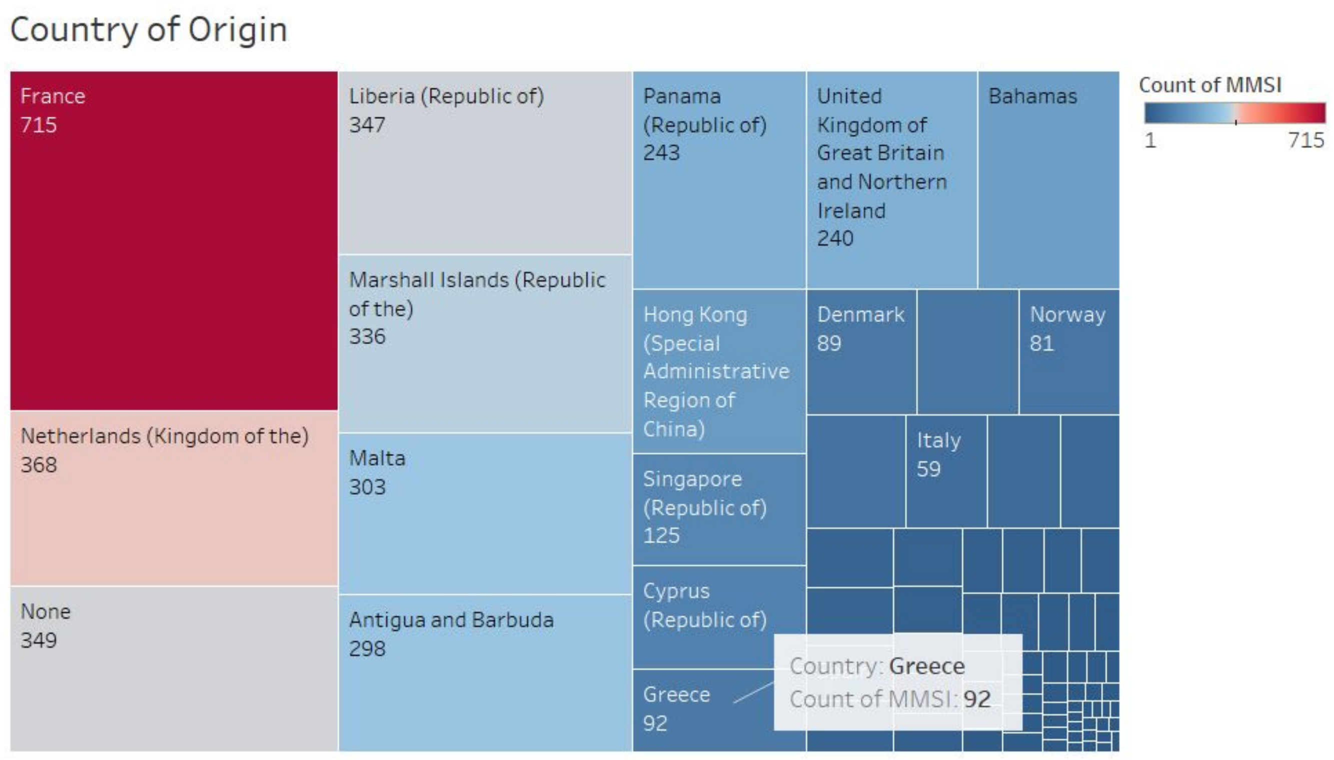
As we can observe, most of the vessels are just passing by the wider area of Brest port over these six months. Furthermore, there are 243 vessels that had not declared their destination.

10) (a) The origin country of the vessels and (b) ships that have no origin or their country is not in the list of countries.

Query (a)	SELECT DISTINCT(sourcemmsi), country
	FROM ais_data.static_ships, ais_status_codes_types.mmsi_country_codes
	WHERE substring(cast(sourcemmsi as varchar),1,3) = country_code::text;

Query (b)	SELECT DISTINCT(sourcemmsi)
	FROM ais_data.static_ships
	WHERE sourcemmsi NOT IN
	(SELECT DISTINCT(sourcemmsi)
	FROM ais_data.static_ships, ais_status_codes_types.mmsi_country_codes
	WHERE substring(cast(sourcemmsi as varchar),1,3) = country_code::text);
	ORDER BY sourcemmsi

Result:



Country and count of MMSI. Color shows count of MMSI. Size shows count of MMSI. The marks are labeled by Country and count of MMSI.

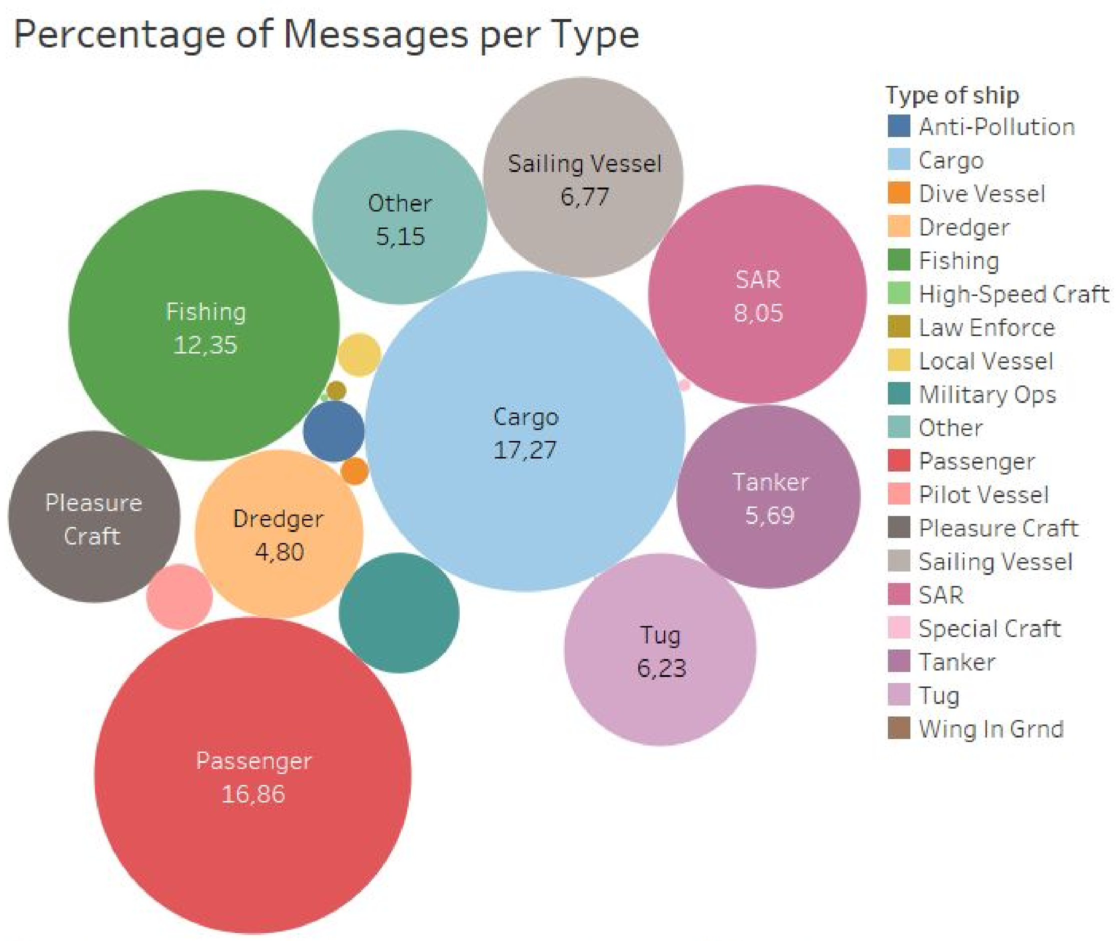
The majority of the ships originated from France which makes sense, as we are examining the ports and sea area of Brittany, France.

But also there is a considerable amount of ships where the origin is unknown and this is due to the fact that the first three digits of their mmsi code do not match any of these in the dataset with the country codes.

11) The percentage of messages per type of vessel.

Query	SELECT (CAST(COUNT(sourcemmsi) AS FLOAT)/1078617)*100, ship_types.type_name
	FROM ais_data.static_ships, ais_status_codes_types.ship_types
	WHERE shiptype BETWEEN shiptype_min AND shiptype_max
	GROUP BY by type_name;

Results:



Type of ship and sum of Percentage. Color shows details about Type of ship. Size shows sum of Percentage. The marks are labeled by Type of ship and sum of Percentage.

The fact that the cargo vessels combined with the tanker vessels have sent only 22,96% of the messages is something to further investigate, because the vessels of the above types cover 77,65% (3.925 vessels) of all the types. This could imply that these kind of vessels turn off their transmitter for some periods of time and thus didn't send the AIS messages as often as they should.

Moreover, we observe that the passengers vessels (just 61 vessels) have sent 16,86% of the messages, which shows the importance of AIS messages transmission for the safety of the people travelling by sea.

12) Number of messages sent each month of the six month interval.

Query October 2015	SELECT COUNT(dynamic_ships.mmsi)
	FROM ais_data.dynamic_ships
	WHERE ts > 1443657601 AND ts < 1446335999;

Result	3.128.690 messages

Query November 2015	SELECT COUNT(dynamic_ships.mmsi)
	FROM ais_data.dynamic_ships
	WHERE ts > 1446336000 AND ts < 1448927999;

Result	3.244.078 messages

Query December 2015	SELECT COUNT(dynamic_ships.mmsi)
	FROM ais_data.dynamic_ships
	WHERE ts > 1448928000 AND ts < 1451606399;

Result	2.761.374 messages

Query January 2016	SELECT COUNT(dynamic_ships.mmsi)
	FROM ais_data.dynamic_ships
	WHERE ts > 1451606400 AND ts < 1454284799;

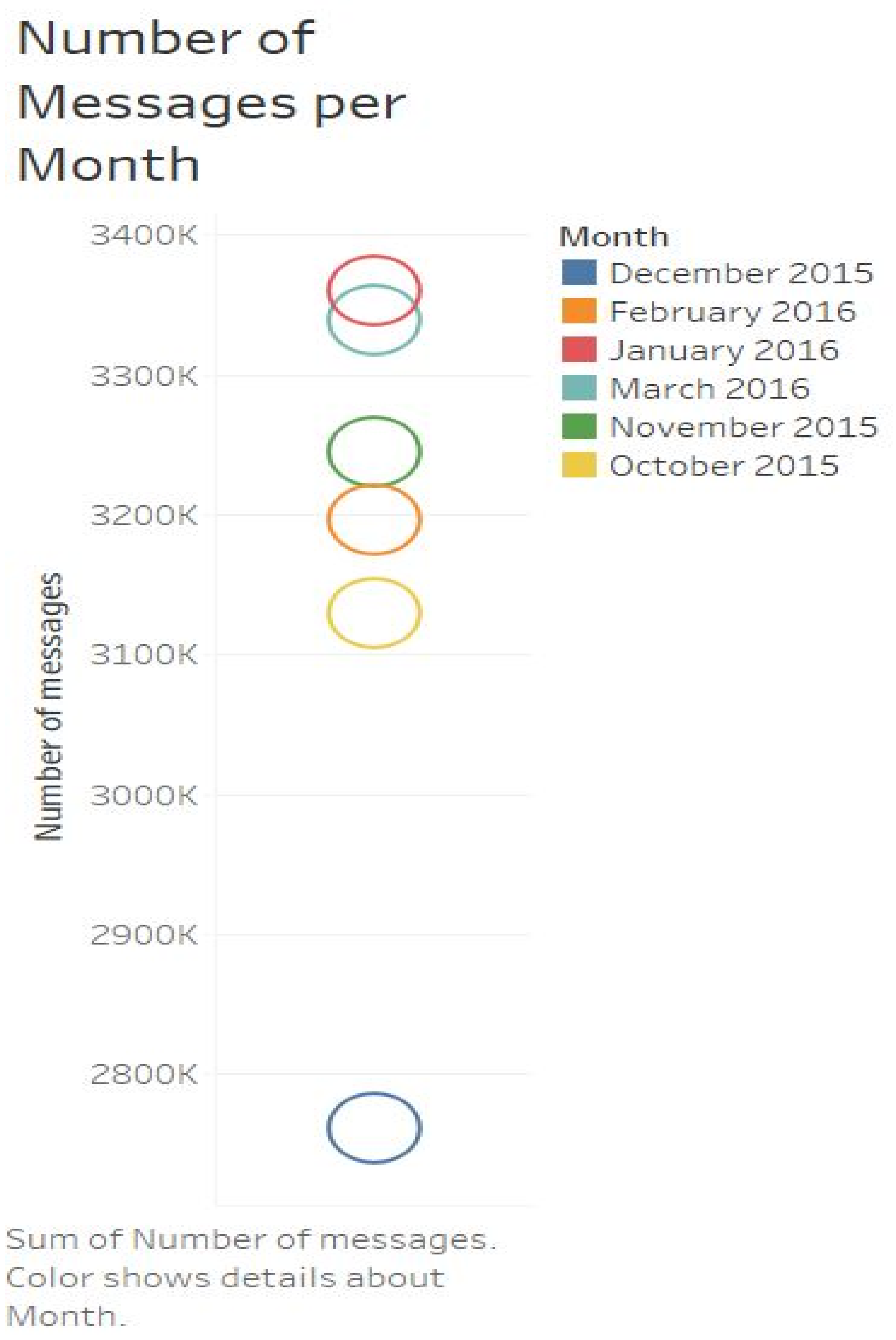
Result	3.359.264 messages

Query February 2016	SELECT COUNT(dynamic_ships.mmsi)
	FROM ais_data.dynamic_ships
	WHERE ts > 1454284800 AND ts < 1456790399;

Result	3.196.105 messages

Query March 2016	SELECT COUNT(dynamic_ships.mmsi)
	FROM ais_data.dynamic_ships
	WHERE ts > 1456790400 AND ts < 1459468799;

Result	3.339.098 messages



The number of messages per month ranges between 3 to 3,35 million, except for December 2015 which is approximately 2,7 million messages. Maybe Christmas Holidays are to blame and the fact that less vessels travel at that time of the year due to bad weather conditions.

13) The number of atons' messages in the dynamic_aton table

Query	SELECT COUNT(DISTINCT(id))
	FROM ais_data.dynamic_aton;

Result	505.717 messages

14) The number of atons' messages per type of aton.

Query

```
SELECT DISTINCT COUNT(*),ais_data.dynamic_aton.typeofaid,
                                    ais_status_codes_types.aton.definition
FROM ais_data.dynamic_aton, ais_status_codes_types.aton
WHERE ais_status_codes_types.aton.id_code=ais_data.dynamic_aton.typeofaid
GROUP BY  ais_data.dynamic_aton.typeofaid,ais_status_codes_types.aton.definition
ORDER BY count(*) ;
```

Result

Definition	Count	Type of aid	Definition	Count	Type of aid
Light, with sectors	232756	6	Cardinal Mark W	86	23
Light, without sectors	178790	5	Isolated danger	18	28
Beacon, Cardinal W	49043	12	Starboard hand Mark	17	25
Default, Type not specified	37411	0	Leading Light Front	5	7
Special Mark	7089	30	RACON	1	2
Safe Water	414	29	Beacon, Safe water	1	18
Leading Light Rear	86	8			

The majority of the messages from atons have been sent by lighthouses (with or without sectors) with 81,38% coverage of the data.

15) The number of messages sent by physical and virtual atons.

Query

```
SELECT VirtualAtons ,PhysicalAtons
FROM (SELECT COUNT(*)
          FROM ais_data.dynamic_aton WHERE virtual=true) AS VirtualAtons,
          (SELECT COUNT(*)
          FROM ais_data.dynamic_aton WHERE virtual=false) AS PhysicalAtons;
```

Result

Physical Atons → 480.458 messages
Virtual Atons → 25.269 messages

Physical atons' messages, such as lighthouses and beacons which send navigational information are a lot higher than virtual atons' messages, which are messages containing navigational information that the authorities transmit where no physical atons exist (broadcast messages to all or messages targeted to a specific vessel/mmsi).

Consequently, we provide some further general information.

16) The number of SAR vessels.

Query	SELECT COUNT(DISTINCT mmsi)
	FROM ais_data.dynamic_sar;

Result	10 vessels

17) The number of SAR messages in the dynamic_sar table.

Query	SELECT COUNT(DISTINCT(id))
	FROM ais_data.dynamic_sar;

Result	4.566 messages

18) The average, minimum and maximum speed of the moving SAR vessels.

Query	SELECT AVG(speed), MAX(speed), MIN(speed)
	FROM ais_data.dynamic_sar
	WHERE speed > 0;

Result	Average speed → 164.98 knots Maximum speed → 353 knots Minimum speed → 1 knots

Advanced Analysis

The next step is to perform more advanced analysis of the datasets.

1) The average number of messages per day.

Query	SELECT count(*)/183
	FROM ais_data.dynamic_ships as cte
	WHERE extract(hour from to_timestamp(cte.ts) AT TIME ZONE 'UTC +1')>8
	and extract(hour from to_timestamp(cte.ts) AT TIME ZONE 'UTC +1')<20

Result	51.729 messages

2) The average number of messages per night.

Query	SELECT COUNT(*)/183
	FROM ais_data.dynamic_ships as cte
	WHERE (EXTRACT(HOUR FROM to_timestamp(cte.ts) AT TIME ZONE 'UTC +1')<8
	AND EXTRACT (HOUR FROM to_timestamp(cte.ts) AT TIME ZONE 'UTC +1')>=0)
	OR (EXTRACT(HOUR FROM to_timestamp(cte.ts) AT TIME ZONE 'UTC +1')>20
	AND EXTRACT(HOUR FROM to_timestamp(cte.ts) AT TIME ZONE 'UTC +1')<=23);

Result	43.277 messages

So the average number of messages per day is higher than the average messages sent at night.

3) The number of ships that are present in ais_data.static_ships but not present in ANFR registrations (limited by 100.000 records of static_data).

Query	with static AS
	(
	SELECT * FROM ais_data.static_ships LIMIT 100000
	)
	SELECT COUNT(DISTINCTstatic.sourcemmsi)
	FROM ais_data.static_ships AS static
	WHERE static.sourcemmsi NOT IN (
	SELECT fr.mmsi
	FROM static, vesselregister.anfr_vessel_list as fr
	WHERE fr.mmsi= static.sourcemmsi AND static.sourcemmsi
	IS NOT null)

Result	4.519 ships

Considering that the static dataset contains approximately 1.000.000 records we can see that examining 10% of it does not give a full perspective of the appearance of the ships in the ANFR registrations. This would lead us to lost entries or unregistered vessels.

4) The number of ships that are more than 30 years old.

Query	SELECT COUNT(DISTINCT fr.mmsi)
	FROM vesselregister.anfr_vessel_list AS fr, ais_data.dynamic_ships AS cte
	WHERE date_part('year',age(now(),
	to_date(fr.date_first_license::text,'DD/MM/YYY')::timestamp))>30
	AND fr.mmsi IS NOT null AND cte.mmsi=fr.mmsi;

Result	139 ships

5) The number of ships that are less than 5 years old.

Query	SELECT COUNT(DISTINCT fr.mmsi)
	FROM vesselregister.anfr_vessel_list AS fr, ais_data.dynamic_ships AS cte
	WHERE date_part('year',age(now(),
	            to_date(fr.date_first_license::text,'DD/MM/YYY')::timestamp))>5
	            AND fr.mmsi IS NOT null AND cte.mmsi=fr.mmsi;

Result	93 ships

6) The number of unique ships in Natura areas (France) (limited by 1.000.000 records of dynamic_ships).

Query	with cte as
	      (
	          SELECT * FROM ais_data.dynamic_ships LIMIT 1000000
	      )
	
	          SELECT COUNT(DISTINCT cte.mmsi)
	          FROM natura2000.spatialfeatures, natura2000.sites, cte
	          WHERE natura2000.spatialfeatures.sitecode = natura2000.sites.sitecode
	          AND natura2000.sites.country_code = 'fr' AND
	         ST_Contains(natura2000.spatialfeatures.geom,
	                              ST_Transform(cte.geom,3035)) = TRUE;

Result	480 ships

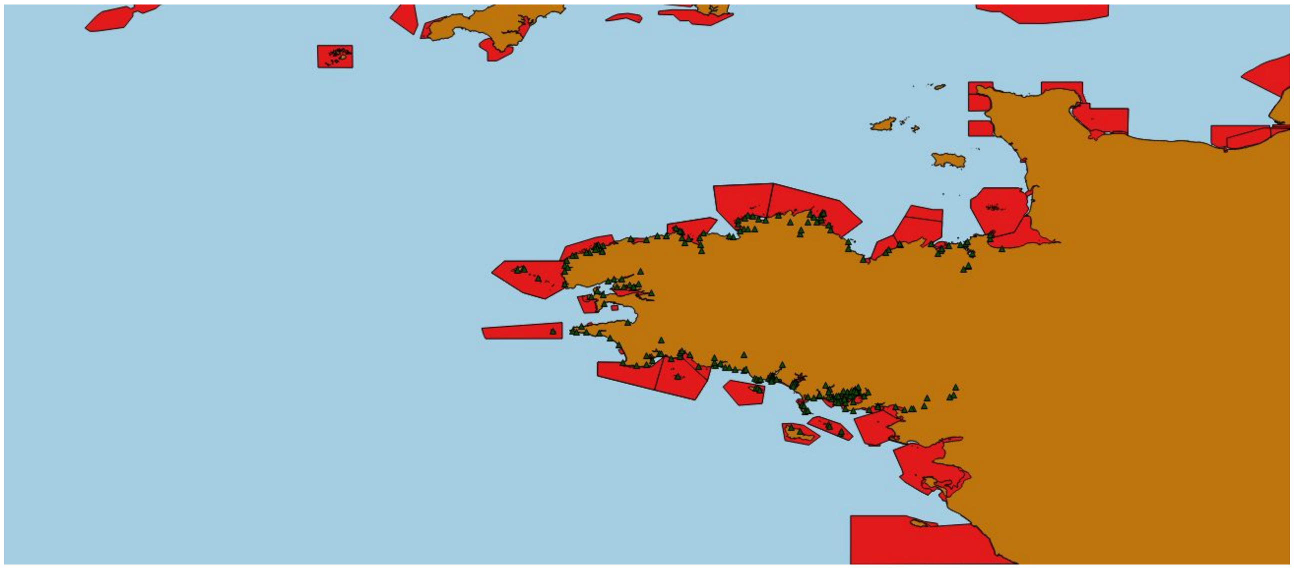


Image2: snapshot of Qgis showing the protected areas near the ports of Brittany.

7) The number of vessels in fishing areas in Europe (limited by 10.000 records of dynamic_ships).

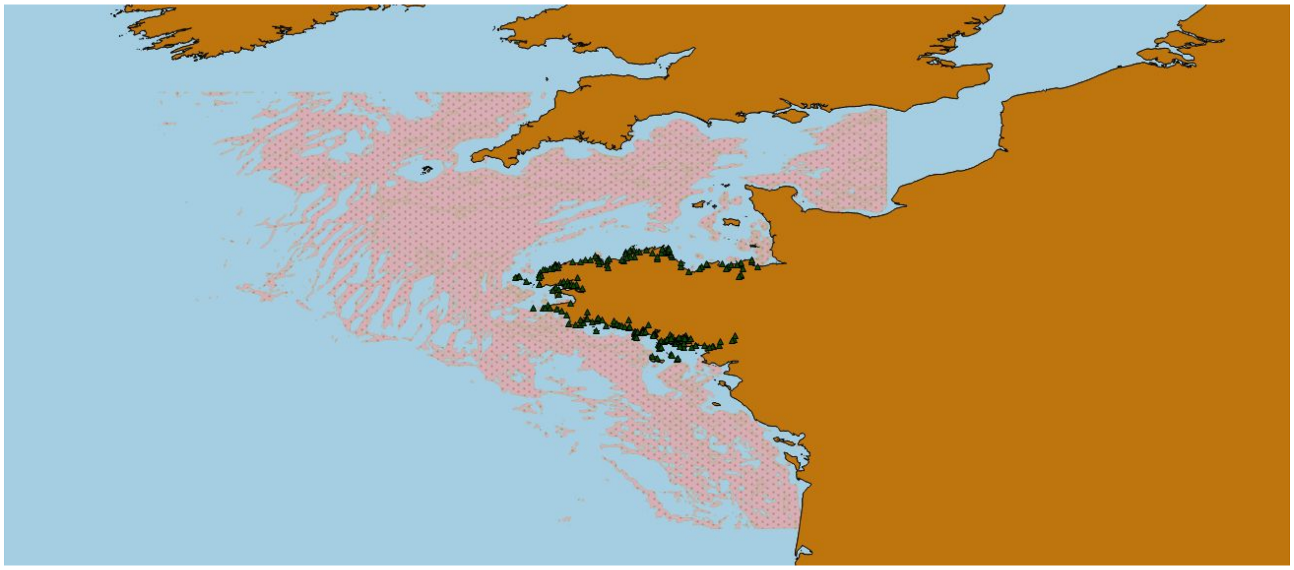


Image3: snapshot of Qgis showing the fishing areas in the wider sea area of France.

Query	with cte AS
	(
	SELECT * FROM ais_data.dynamic_ships LIMIT 10000
	)
	SELECT COUNT(DISTINCT mmsi)
	FROM geographic_features.fishing_areas_eu, cte
	WHERE ST_Contains(geographic_features.fishing_areas_eu.geom,
	ST_Transform(cte.geom,4326))=TRUE;

Result	4 ships

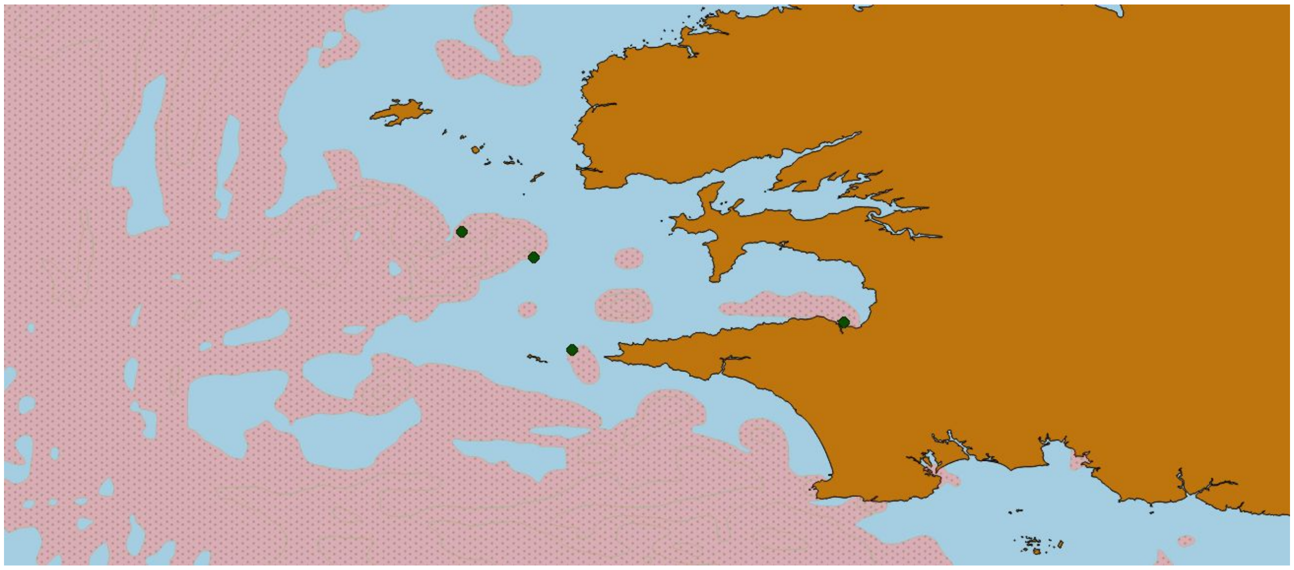


Image4: snapshot of Qgis showing the ships found in fishing areas

8) The number of messages from fishing areas.

Query	with cte AS
	    (
	        SELECT * FROM ais_data.dynamic_ships LIMIT 10000
	    )
	    SELECT COUNT(*)
	    FROM geographic_features.fishing_areas_eu, cte
	    WHERE ST_Contains(geographic_features.fishing_areas_eu.geom,
	                    ST_Transform(cte.geom,4326))=TRUE;

Result	1139 messages

9) The number of vessels in fishing areas that have no fishing registration.

Query	with cte as
	        (
	                select * from ais_data.dynamic_ships LIMIT 80000
	        )
	
	        select fr.mmsi,to_timestamp(cte.ts) AT TIME ZONE 'UTC +1', to_date(eu.eventstartdate::text,'YYYYMMDD')::timestamp,
	        to_date(eu.eventenddate::text,'YYYYMMDD')::timestamp
	        from vesselregister.eu_fishingvessels as eu, vesselregister.anfr_vessel_list as fr,cte,
	        geographic_features.fishing_areas_eu as fish
	        where eu.licenseind='N' and eu.vesselname=fr.ship_name and cte.mmsi=fr.mmsi
	        and to_timestamp(cte.ts)>=to_date(eu.eventstartdate::text,'YYYYMMDD')::timestamp

	and to_timestamp(cte.ts)<=to_date(eu.eventenddate::text,'YYYYMMDD')::timestamp and ST_Contains(fish.geom,ST_Transform(cte.geom,4326))=true;
Result	0 vessels

With a limit of 80000 records from ais_data.dynamic_ships, no entries returned, meaning no ships without a fishing registration in fishing areas, for that data selection.

Conclusion

In this (part of the) exercise we are investigating real Marine Time Information data and aim to extract valuable information regarding:

1. The size of the dataset
2. The behavior of the ships based on type/season/time of day
3. The number of messages ships send on different scenarios
4. The different type of atons and how they are used to broadcast information
5. Environmental and Spatial restrictions
6. Erroneous data and possible explanations reasoning their outlier values

In the next parts, we will use the information gathered to perform Complex Event Recognition, using FlinkCEP and RTEC.

Part I: Automata-based CEP with FlinkCEP

Introduction

In this part of the project, the AIS messages stored in the database, are treated like a data stream, meaning that each AIS message represents an event in time. The input data of the Apache Flink engine, which is being used, is supposed to be a boundless dataset from continuously "produced" events, being processed in real time.

Apache Flink is a stream processing engine for stateful computations over unbounded and bounded data streams and comes with a rich DataStream API, windowing semantics and a lot of useful libraries for complex event processing (CEP).

Complex event processing (CEP) is the process where continuously incoming events are being matched against a pattern. Contrary to the traditional databases, where stored data are being used for executing a query, complex event processing uses stored queries to "execute" data on them. All the events that are not incompatible with the query can be instantly rejected. The advantage is that CEP queries can be applied on a boundless stream of data with the input events being processed in real time and the results of matches being emitted, immediately after the sequence of events needed are consumed. This means that CEP can produce real time results.

A Flink process is built by data streams and their transformations. A stream is an infinite flow of incoming events and a transformation is an action, that takes a stream as input, and produces another stream as output. When executed, Flink programs are mapped to streaming dataflows, consisting of streams and transformation operators.

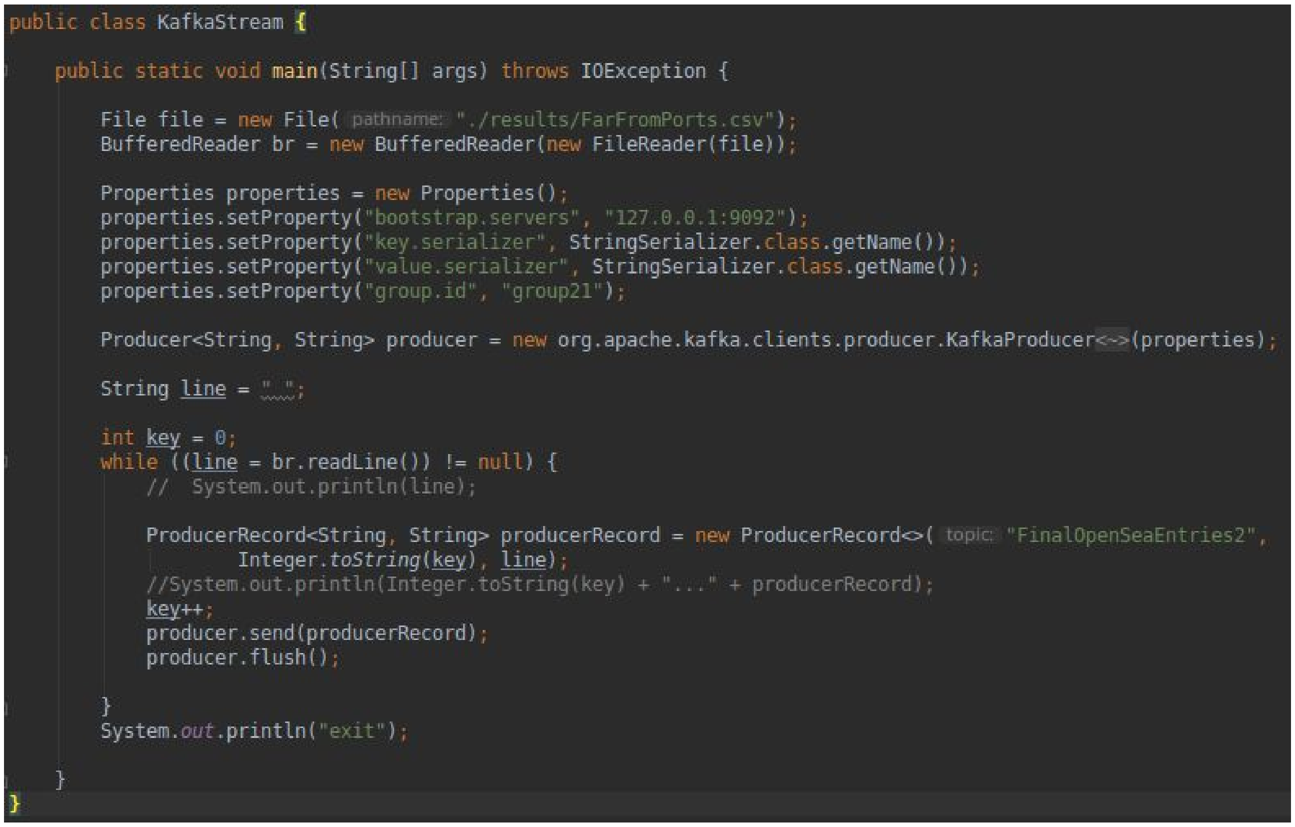
Data Consuming

To simulate the AIS messages as a stream of events, suitable for ingestion by Flink, two methods are used.

The first method is by directly using the .csv file that contains the AIS messages (one in each line) and pass each line to the Flink processes. The lines are being read as a datastream of strings, creating a stream of events.

The second method is by feeding the lines of the .csv file to Apache Kafka, a high-throughput message queuing system, designed for making streaming data available to multiple data consumers and for providing publish-subscribe functionality. The events from the file, are being pushed to a Kafka topic, through the Kafka Producer API, which are then consumed by Flink jobs, through the FlinkKafkaConsumer connector, into a datastream of strings, representing the events. This allows various Flink jobs to read the stream at different speeds and also to sink their output to other topics, so that other consumers can ingest it.

The Kafka source was used for the simple pattern executor, while the complex pattern executors use the .csv as the input method.



```
public class KafkaStream {

    public static void main(String[] args) throws IOException {

        File file = new File( pathname: "./results/FarFromPorts.csv");
        BufferedReader br = new BufferedReader(new FileReader(file));

        Properties properties = new Properties();
        properties.setProperty("bootstrap.servers", "127.0.0.1:9092");
        properties.setProperty("key.serializer", StringSerializer.class.getName());
        properties.setProperty("value.serializer", StringSerializer.class.getName());
        properties.setProperty("group.id", "group21");

        Producer<String, String> producer = new org.apache.kafka.clients.producer.KafkaProducer<~>(properties);

        String line = "";

        int key = 0;
        while ((line = br.readLine()) != null) {
            //  System.out.println(line);

            ProducerRecord<String, String> producerRecord = new ProducerRecord<>( topic: "FinalOpenSeaEntries2",
                    Integer.toString(key), line);
            //System.out.println(Integer.toString(key) + "..." + producerRecord);
            key++;
            producer.send(producerRecord);
            producer.flush();

        }
        System.out.println("exit");

    }
}
```

Code0: pattern for feeding the lines into a Kafka topic

No matter the method of consuming the events (lines) of the .csv file, the next step is to use the *map operator* to transform the datastream of strings into a datastream of Objects of the class DynamicShipClass, which is a class that represents an event and contains the individual elements of it, like the ship's mmsi, speed, turn, course, timestamp, as well as some extra information, which is being deduced from the simple patterns execution.

At the same time, KeyBy operator groups the input in partitions regarding the selected key. For example, if mmsi is selected as the key, the AIS messages will be grouped by mmsi, so messages from a single vessel will form a separate group of data. The aforementioned keyBy strategy is useful in order to investigate patterns involving the actions of a single vessel. Another strategy used in our implementation is to use the grid Id as the key in order to investigate patterns involving the vessels appearing in the same grid.

Timestamp Assignment

Apache Flink supports different methods for timestamp assignment for the stateful stream processing:

- **Processing time**: the system time of the machine that executes the operation.
- **Event time**: the time that each individual event occurred on its producing device.
- **Ingestion time:** the time that events enter Flink.

In our implementation, the choice is the default choice, thus the *Processing time.* Even though each event (each AIS message) has an assigned UNIX timestamp, we assume that there is no delay in the arrival of the events and the events cannot be out of order, as in our case, we have already checked and sorted the events in ascending timestamp . So there is no need to use the Event time processing because it will create useless latency. As far as time restriction for a pattern is needed, the timestamp of each event is being used to create the context, in which the pattern has meaning.

Windowing

Windows are an important notion in the processing of infinite streams. What windows are doing is splitting the stream into "chunks" of finite size, over which computations are being applied. So, a window is created as soon as the first element that should belong to this window arrives, and is removed when the time passes its end timestamp.

There are four different types of windowing supported by Flink:

- **Tumbling windows**: assigns each element to a window of a specified window size.
- **Sliding windows**: assigns elements to windows of fixed length, but with an additional window slide parameter that controls how frequently a sliding window is started.
- **Session windows**: groups elements by sessions of activity.
- **Global windows**: assigns all elements with the same key to the same single global window.

Spatial Information

To be able to use the spatial information provided by each event, thus the latitude and longitude values, we create a conceivable grid. The boundaries of this grid are the lowest and highest values of latitude and longitude in our dataset, because there are no vessels that we investigate above this boundaries.

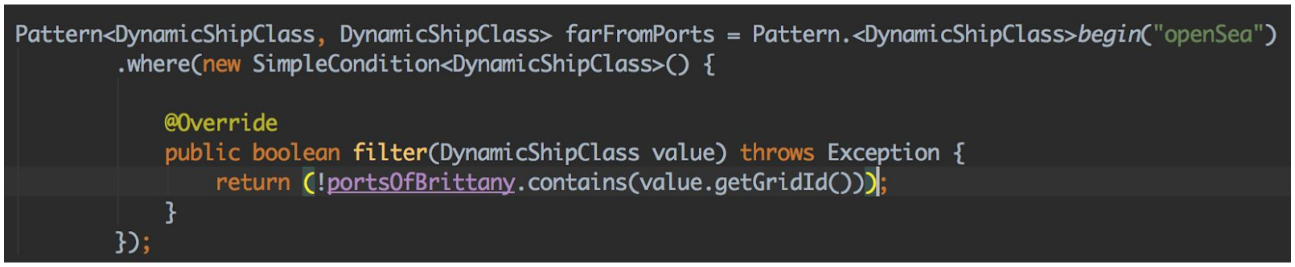
Then, this grid is being split into cells. The grid cells are roughly 500 x 500 meters, in real distance, and sequentially numbered from NorthWest to SouthEast starting by zero. Each of the AIS messages is being assigned to a specific grid cell according to its latitude and longitude values. This way, we can use the information of where each vessel is on the map (grid id) at a given timestamp, when we want to check its proximity with some other spatial node.

Vessels Near Ports Exclusion

Our first step is to find all the vessels that are near ports and exclude them from the rest of the streaming pipeline. The reason is that when vessels are near the ports, they demonstrate a behavior that is rather normal for their position, like communication gaps, slow speed and instantaneous turns, but would be suspicious if it occured in the open sea.

To achieve this we use the same mapping to the aforementioned grid cells for the ports of Brest. Each port is mapped to a grid cell and an array of integers containing the numbers of cells where the ports “exist” is created.

Then, the following pattern is created to “discover” which of the AIS messages are transmitted from vessels that are near ports, by checking whether the grid cell of the message is not in the list of grid cells where the ports lie.



```
Pattern<DynamicShipClass, DynamicShipClass> farFromPorts = Pattern.<DynamicShipClass>begin("openSea")
        .where(new SimpleCondition<DynamicShipClass>() {

            @Override
            public boolean filter(DynamicShipClass value) throws Exception {
                return (!portsOfBrittany.contains(value.getGridId()));
            }
        });
```

Code1: pattern for excluding the vessels near ports

Due to the large amount of AIS messages included in the nari_dynamic file and the amount memory and time needed to process it, we chose to create a smaller dataset. So, we process the file with the above pattern and stopped it when it reached 2.000.000 messages. This resulted in the creation of the smaller file containing messages for about twenty five days, thus from Wednesday, 30-Sep-15 22:00:01 UTC to Sunday, 25-Oct-15 05:16:35 UTC. This time corresponds to 1.919.741 AIS messages.

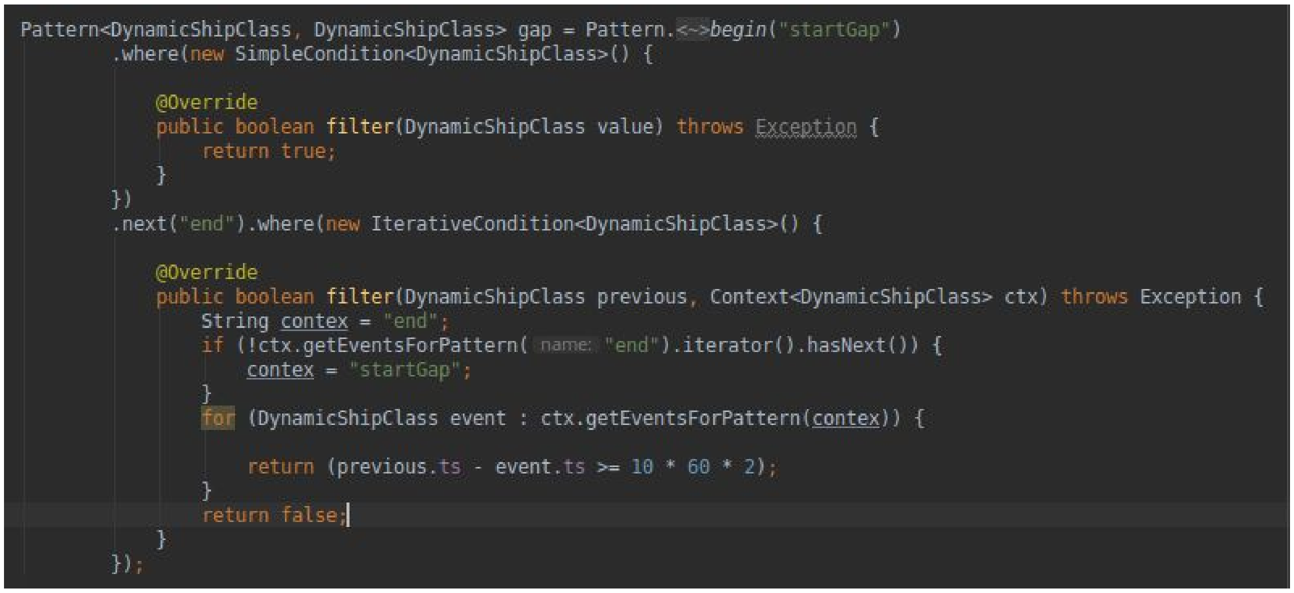
Patterns of Interesting Maritime Activity

1) Communication Gaps

Considering that a ship is supposed to send a signal containing its dynamic AIS message every (approximately) 10 seconds, we want to check if a vessel has not sent a message over a longer period of time, e.g. for the past 20 minutes. This is important, because it might be an indication that the vessel might be involved in some kind of illegal activity and has turned off the transmitter.

So the pattern below is trying to export these gaps in the transmittings, by investigating each vessel's messages and checking if the duration between the timestamps of two consecutive events is more than 20 minutes.

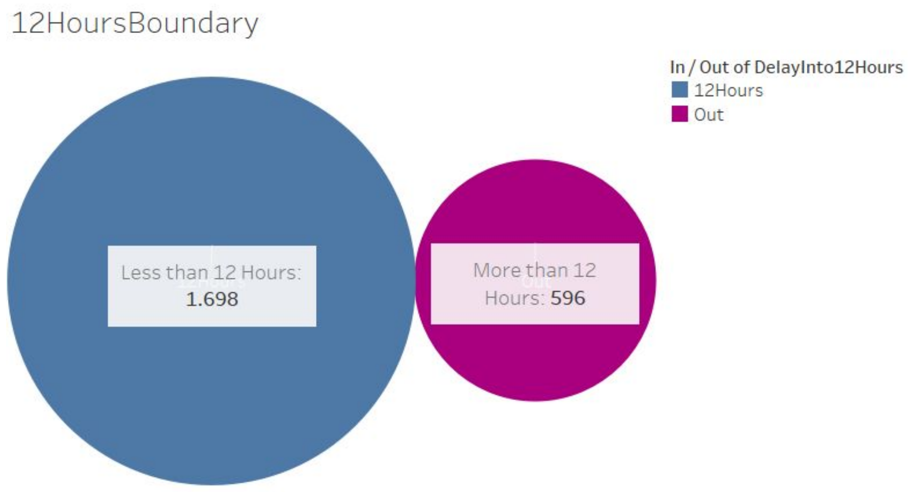
We expect a result to match this pattern if the second message is received after more than 20 minutes from the first, by comparing the two unix timestamps.



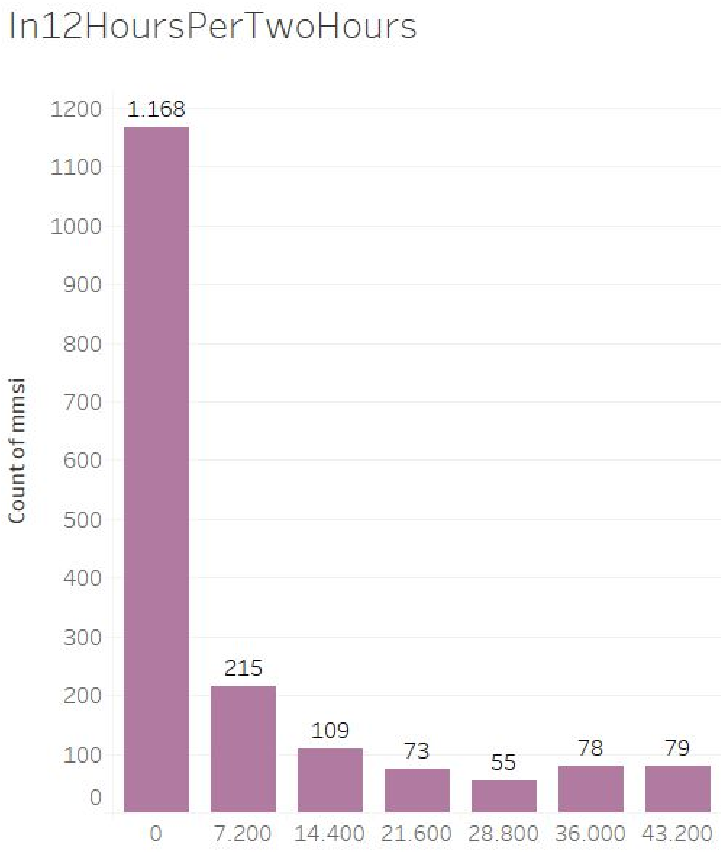
```java
Pattern<DynamicShipClass, DynamicShipClass> gap = Pattern.<~>begin("startGap")
        .where(new SimpleCondition<DynamicShipClass>() {

            @Override
            public boolean filter(DynamicShipClass value) throws Exception {
                return true;
            }
        })
        .next("end").where(new IterativeCondition<DynamicShipClass>() {

            @Override
            public boolean filter(DynamicShipClass previous, Context<DynamicShipClass> ctx) throws Exception {
                String contex = "end";
                if (!ctx.getEventsForPattern( name: "end").iterator().hasNext()) {
                    contex = "startGap";
                }
                for (DynamicShipClass event : ctx.getEventsForPattern(contex)) {

                    return (previous.ts - event.ts >= 10 * 60 * 2);
                }
                return false;
            }
        });
```

Code2: pattern for detecting communication gaps

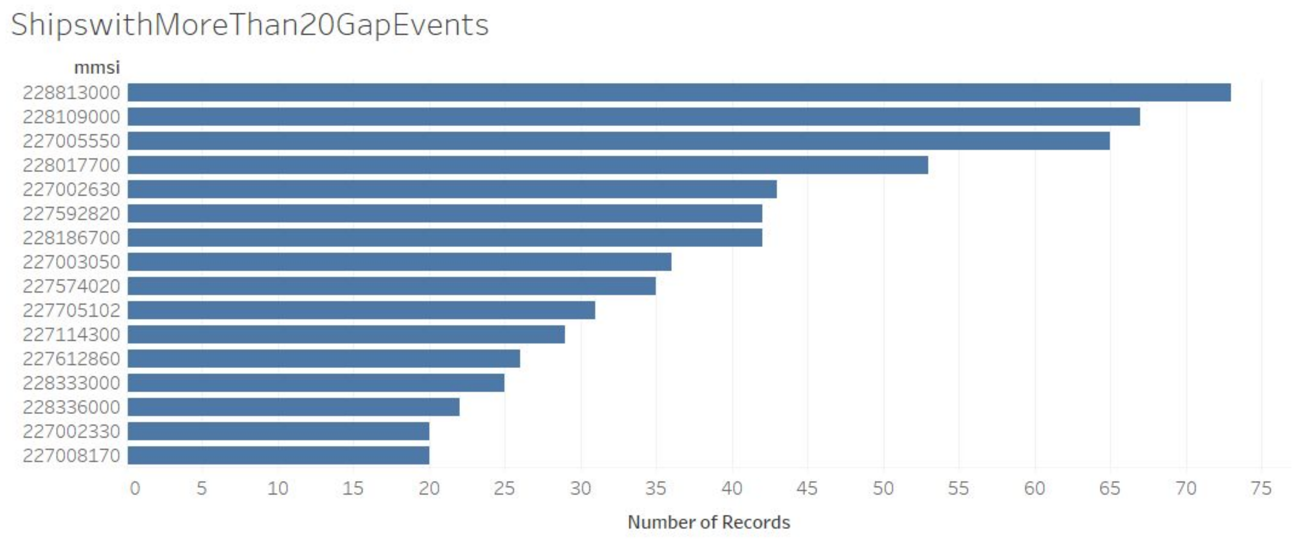
The above pattern matching can provide us with some useful insights. As we can observe from the visualization below, there are 1.698 vessels for which communication gaps more than 12 hours was detected and 596 vessels with duration of gaps less than 12 hours. This needs to be further investigated, for the reasons of such a long gap period.



For the gaps with duration less than 12 hours a second visualization shows that the majority of them lasts less than two hours.



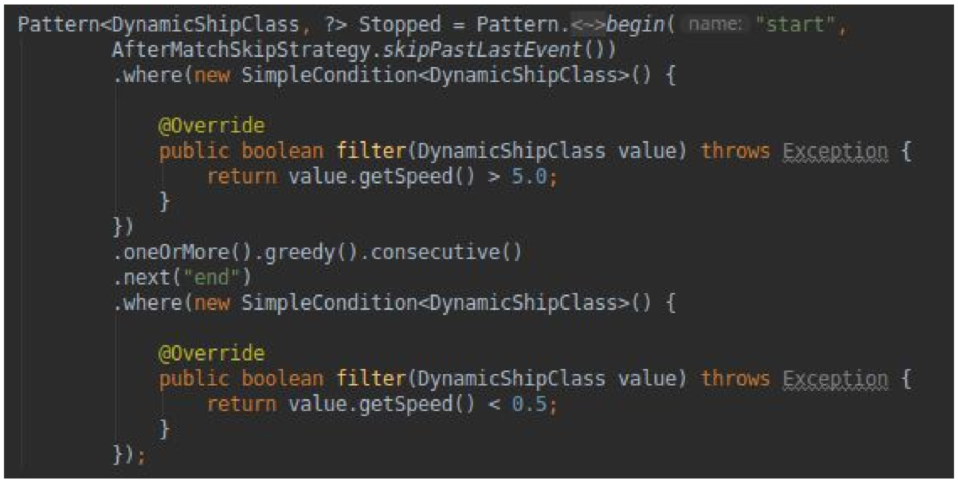
Last but not least, we can show the vessels, for which more than 20 gaps are detected during this period of approximately twenty five days and try to explain the reason behind this, by examining maybe the type of the vessel and other circumstances that may affects the results.



2) Stopped Vessel

A vessel stopped in the open sea indicates that it might faces a problem (movement ability affected) or that is involved in some kind of abnormal situation like vessels' rendez-vous or polluting the sea.

There are two ways we can define the stopped vessel. The pattern below is one of them the is used to detect the stopped vessel in the open sea.

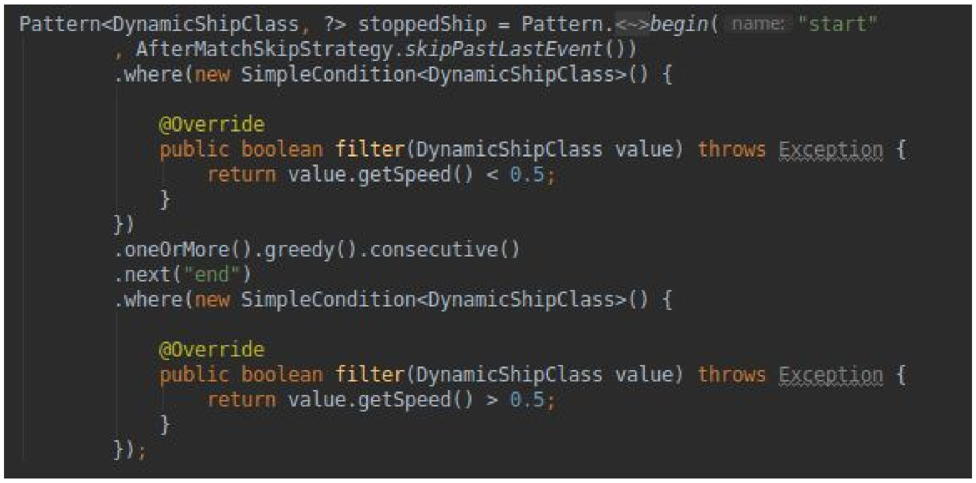


```
Pattern<DynamicShipClass, ?> Stopped = Pattern.<~>begin( name: "start",
        AfterMatchSkipStrategy.skipPastLastEvent())
        .where(new SimpleCondition<DynamicShipClass>() {

            @Override
            public boolean filter(DynamicShipClass value) throws Exception {
                return value.getSpeed() > 5.0;
            }
        })
        .oneOrMore().greedy().consecutive()
        .next("end")
        .where(new SimpleCondition<DynamicShipClass>() {

            @Override
            public boolean filter(DynamicShipClass value) throws Exception {
                return value.getSpeed() < 0.5;
            }
        });
```

Code3: pattern for detecting the stopped vessels (1st way)

The pattern is searching for one or more consecutive events with speed higher than 0.5 knots, followed by an event with very low (almost stopped) speed, which indicates the end of the period that the vessel was moving.

The second way to detect a stopped vessel is by searching for one or more consecutive events with very low (less than 0.5) speed, followed by an event with higher speed, which indicates the end of the period that the vessel was stopped. This is implemented with the pattern below.

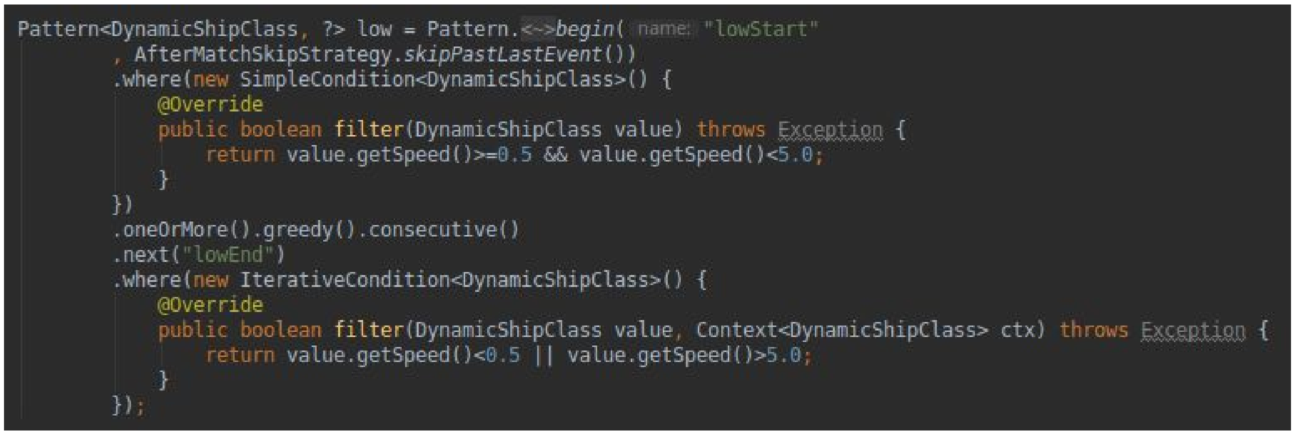


```java
Pattern<DynamicShipClass, ?> stoppedShip = Pattern.<~>begin( name: "start"
        , AfterMatchSkipStrategy.skipPastLastEvent())
        .where(new SimpleCondition<DynamicShipClass>() {

            @Override
            public boolean filter(DynamicShipClass value) throws Exception {
                return value.getSpeed() < 0.5;
            }
        })
        .oneOrMore().greedy().consecutive()
        .next("end")
        .where(new SimpleCondition<DynamicShipClass>() {

            @Override
            public boolean filter(DynamicShipClass value) throws Exception {
                return value.getSpeed() > 0.5;
            }
        });
```

Code4: pattern for detecting the stopped vessels (2nd way)

3) Low Speed

The next thing we want to check is whether a vessel has low speed in the open sea, because It is rather unusual for a vessel to travel with low speed when its position is away from the shore.



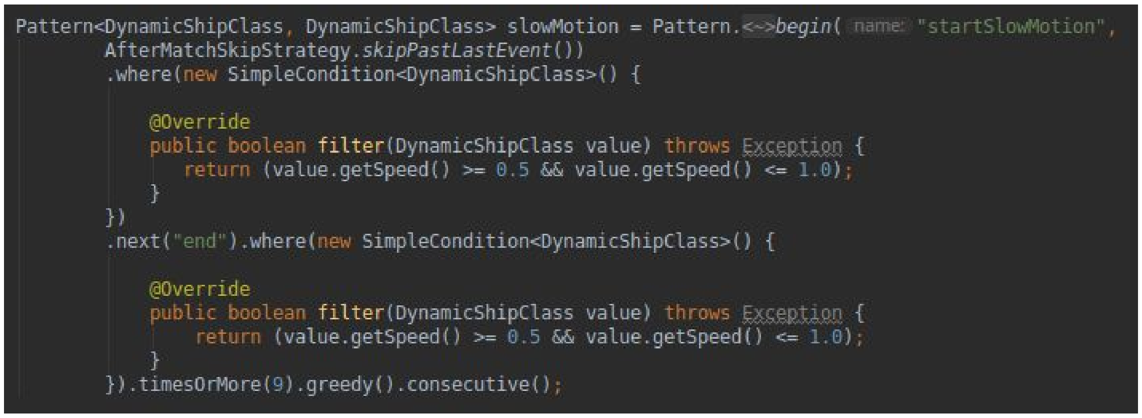
```java
Pattern<DynamicShipClass, ?> low = Pattern.<~>begin( name: "lowStart"
        , AfterMatchSkipStrategy.skipPastLastEvent())
        .where(new SimpleCondition<DynamicShipClass>() {
            @Override
            public boolean filter(DynamicShipClass value) throws Exception {
                return value.getSpeed()>=0.5 && value.getSpeed()<5.0;
            }
        })
        .oneOrMore().greedy().consecutive()
        .next("lowEnd")
        .where(new IterativeCondition<DynamicShipClass>() {
            @Override
            public boolean filter(DynamicShipClass value, Context<DynamicShipClass> ctx) throws Exception {
                return value.getSpeed()<0.5 || value.getSpeed()>5.0;
            }
        });
```

Code5: pattern for detecting vessels with low speed

The pattern above is searching for one or more consecutive events with low speed (between 0,5 and 5 knots) followed by an event speed outside the above boundaries which indicates the end of the period that the vessel was having low speed, either because it developed higher speed or because it stopped.

4) Slow Motion

After getting the vessels with low speed, the next thing is to detect vessels that move with slow motion away from the shore.



```
Pattern<DynamicShipClass, DynamicShipClass> slowMotion = Pattern.<~>begin( name: "startSlowMotion",
        AfterMatchSkipStrategy.skipPastLastEvent())
        .where(new SimpleCondition<DynamicShipClass>() {

            @Override
            public boolean filter(DynamicShipClass value) throws Exception {
               return (value.getSpeed() >= 0.5 && value.getSpeed() <= 1.0);
            }
        })
        .next("end").where(new SimpleCondition<DynamicShipClass>() {

            @Override
            public boolean filter(DynamicShipClass value) throws Exception {
                return (value.getSpeed() >= 0.5 && value.getSpeed() <= 1.0);
            }
        }).timesOrMore(9).greedy().consecutive();
```

Code6: pattern for detecting the vessels that move with slow motion

The above pattern is similar to the previous one but this time the slow speed (between 0.5 and 1.0 knot) has to maintain for at least nine events for the stream to be considered as slow motion.

We choose a sequence of events that matches with the above pattern to show the trajectory of a specific vessel which is detected to be moving in slow speed.

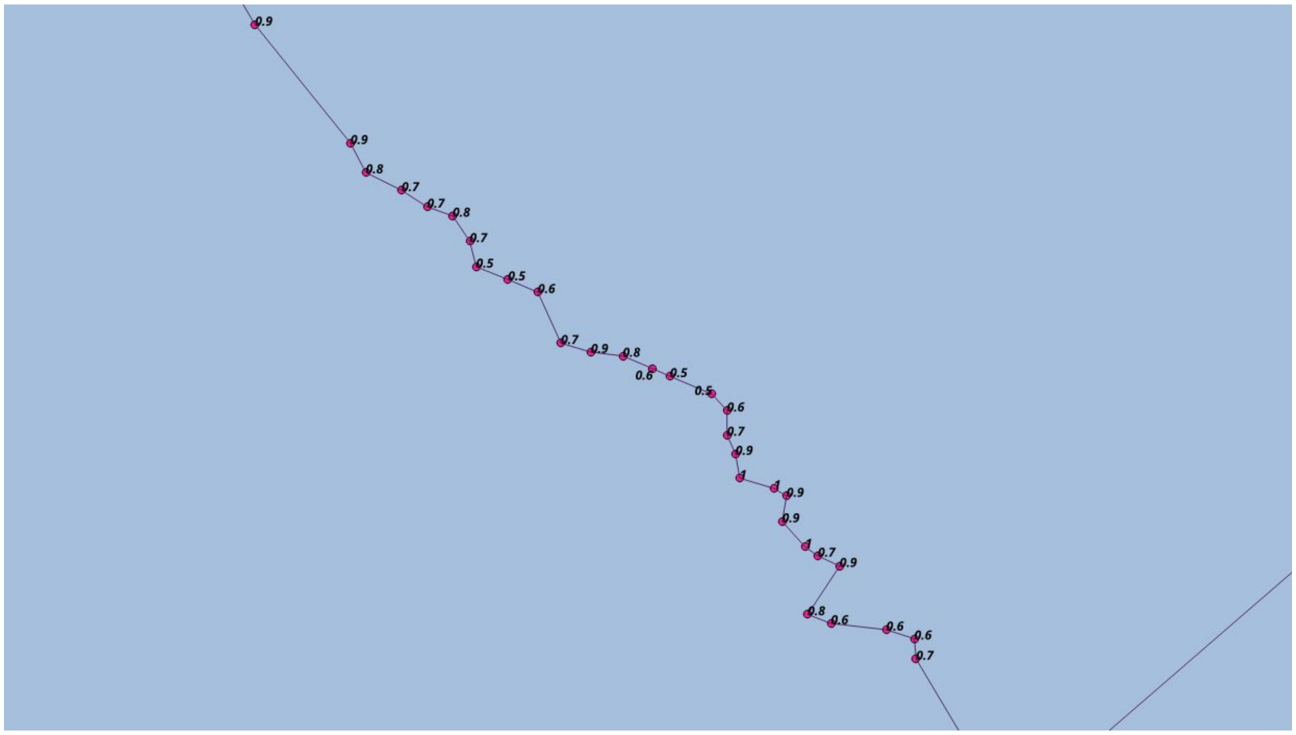


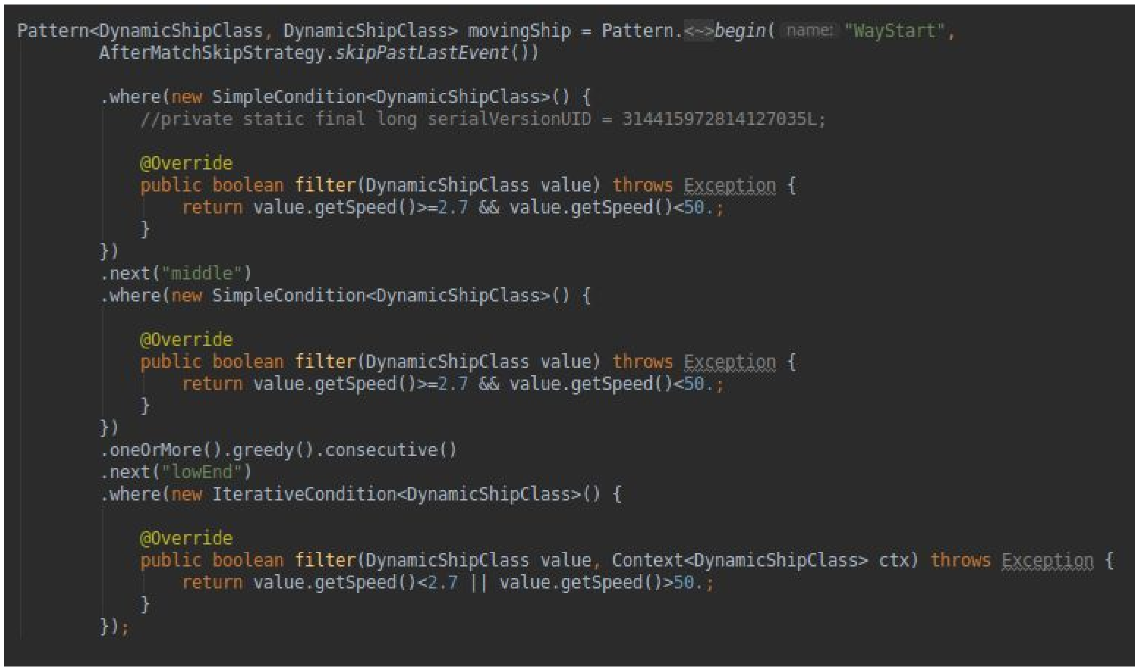
Image5: snapshot of Qgis showing a vessel moving in slow motion

As we can observe, the vessel moves slowly and its trajectory is rather abnormal. It takes continuous turns in a small amount of time which might indicate that it is involved in fishing activity, something that should be investigated further by the type of the ship and the area it is into.

5) Under Way

When a vessel is said to be under way, this means it is not at anchor, not at shore or aground and it is maneuvering using its engine. It is important for us to know when a ship is under way, because this way we can show there is a possibility that the status transmitted by the vessel is not the true one.

The pattern below is searching for one or more consecutive events where the speed of the vessel is between 2.7 and 25 knots followed by an event with speed lower than 2.7 knots, to find the ones that stopped or those that increased speed to higher than 50 knots.



```
Pattern<DynamicShipClass, DynamicShipClass> movingShip = Pattern.<~>begin( name: "WayStart",
        AfterMatchSkipStrategy.skipPastLastEvent())

        .where(new SimpleCondition<DynamicShipClass>() {
            //private static final long serialVersionUID = 3144159728141270351;

            @Override
            public boolean filter(DynamicShipClass value) throws Exception {
                return value.getSpeed()>=2.7 && value.getSpeed()<50.;
            }
        })
        .next("middle")
        .where(new SimpleCondition<DynamicShipClass>() {

            @Override
            public boolean filter(DynamicShipClass value) throws Exception {
                return value.getSpeed()>=2.7 && value.getSpeed()<50.;
            }
        })
        .oneOrMore().greedy().consecutive()
        .next("lowEnd")
        .where(new IterativeCondition<DynamicShipClass>() {

            @Override
            public boolean filter(DynamicShipClass value, Context<DynamicShipClass> ctx) throws Exception {
                return value.getSpeed()<2.7 || value.getSpeed()>50.;
            }
        });
```

Code7: pattern for detecting the vessels that are under way

The following images show a vessel's trajectory that matched with the above pattern and at first is moving away from the shore, being under way, and then following a weird trajectory inside the Brest marine Area.

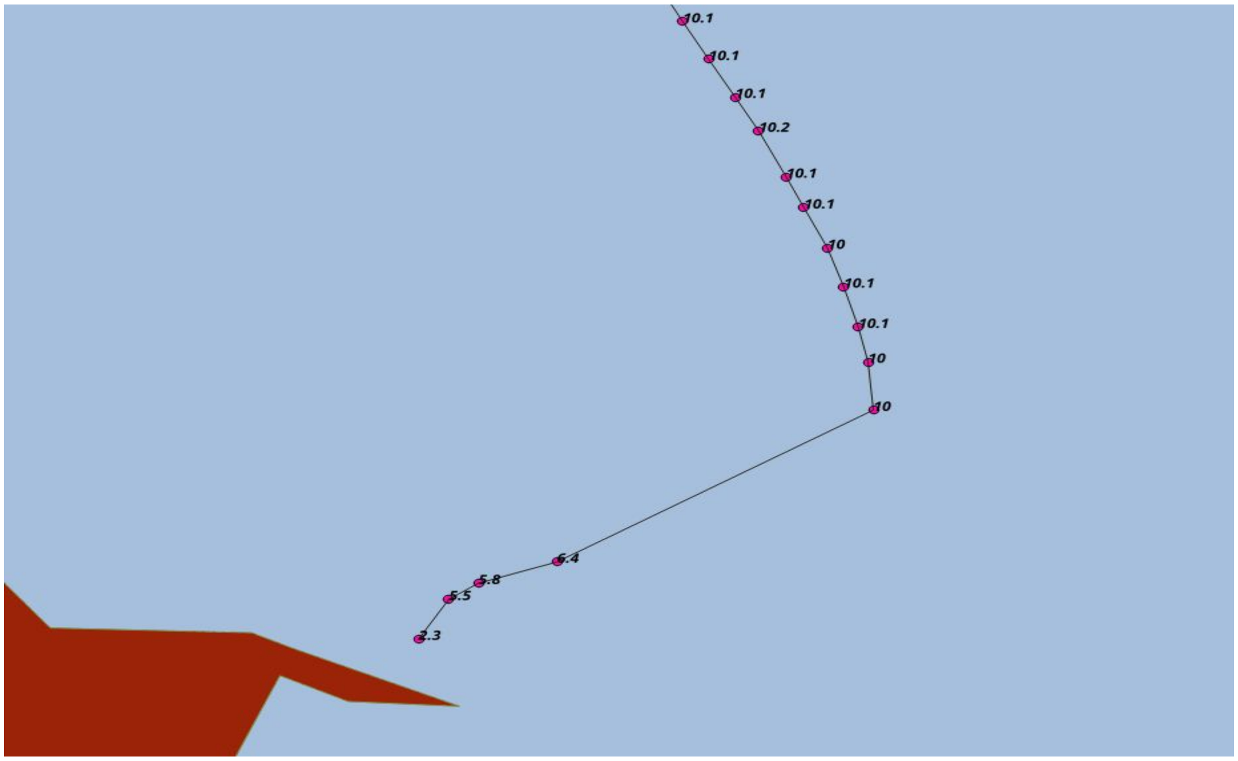


Image6: snapshot of Qgis showing a vessel under way moving away from the shore

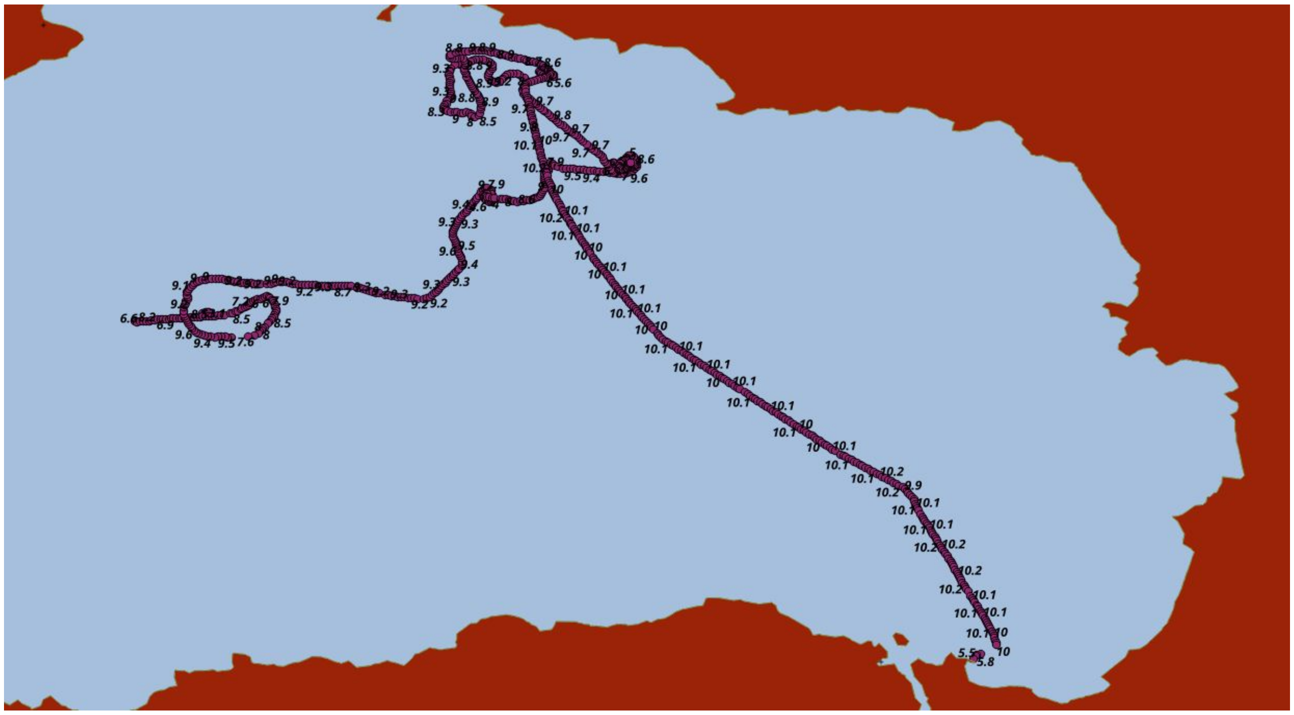
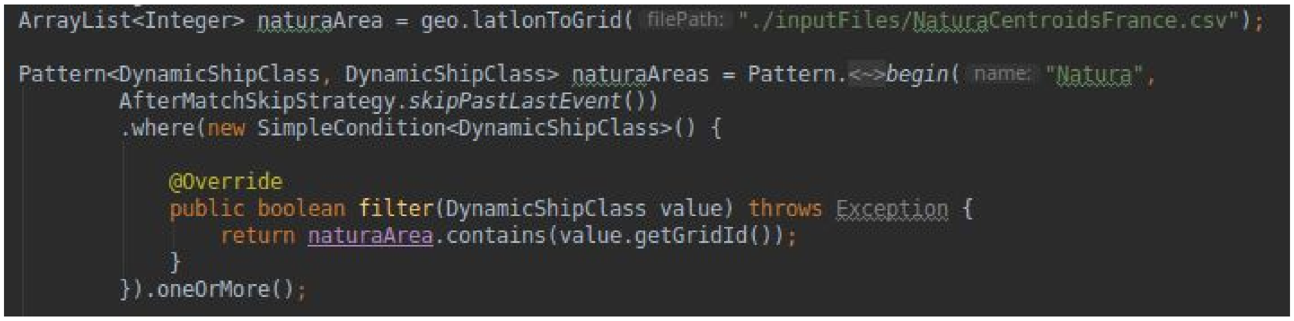


Image7: snapshot of Qgis showing a vessel under way in the marine area of Brest, France

6) Natura Areas

Our next thought is to find vessels that are travelling inside the protected areas of natura2000 sites. Natura 2000 is a list of sites for the breeding and resting of rare and threatened species. These sites are spread around the EU countries, and include both land and sea areas. Monitoring the traffic of the vessels in these sites can help us understand what impact it has on their preservation and the long-term survival of Europe's most valuable and threatened species and habitats.



```
ArrayList<Integer> naturaArea = geo.latlonToGrid( filePath: "./inputFiles/NaturaCentroidsFrance.csv");

Pattern<DynamicShipClass, DynamicShipClass> naturaAreas = Pattern.<~>begin( name: "Natura",
        AfterMatchSkipStrategy.skipPastLastEvent())
        .where(new SimpleCondition<DynamicShipClass>() {

            @Override
            public boolean filter(DynamicShipClass value) throws Exception {
                return naturaArea.contains(value.getGridId());
            }
        }).oneOrMore();
```

Code8: pattern for detecting the vessels that are travelling inside the natura areas

The pattern above searches for vessels that, according to their latitude and longitude (grid cell), their position lies into the cells of the centroids of the protected area. For creating the arraylist with the grid ids where the Natura areas' centroids are, we select the Natura areas of France from our SQL database and then export the latitudes and longitudes of their centroids to a file. Using this file, then we map the values of lat and lon to the proper grid cell id.

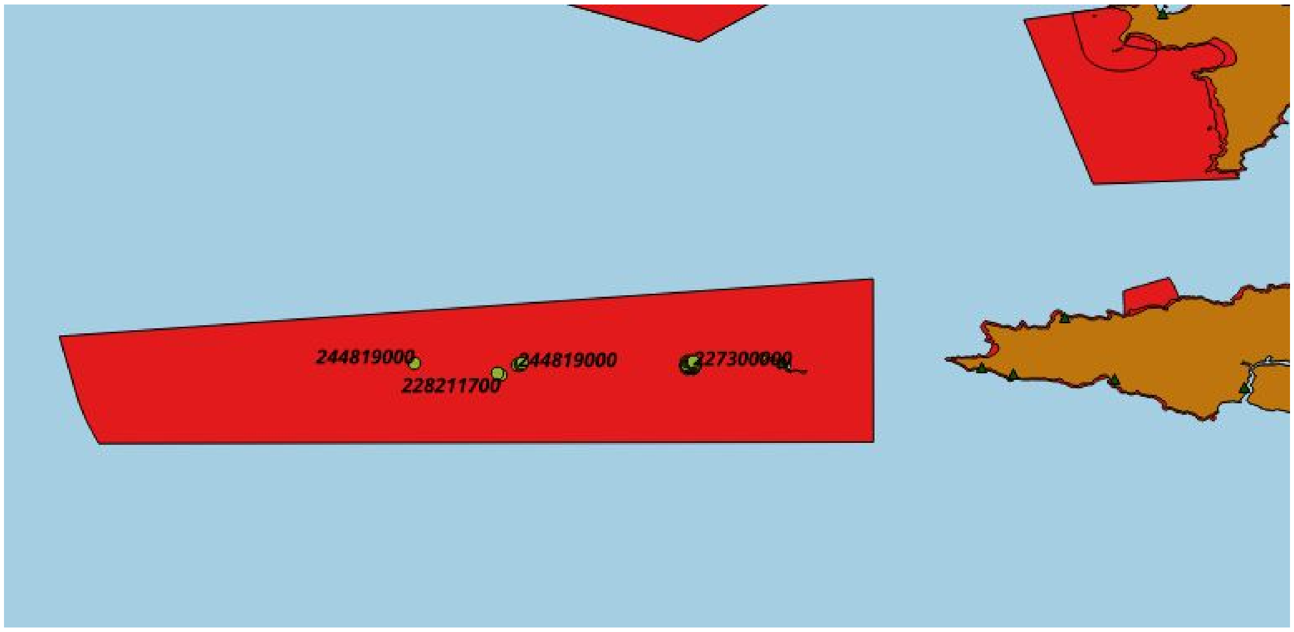
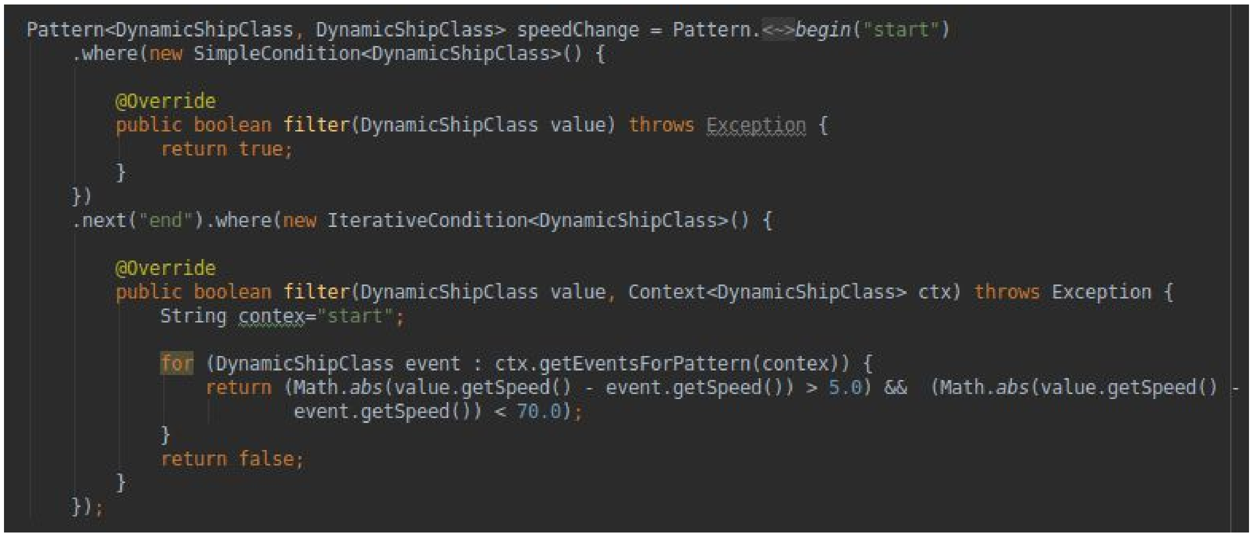


Image8: snapshot of Qgis showing four vessels moving inside one of the NaturaAreas2000

7) Speed Change

The event where a vessel is suddenly changing its speed can be of importance because it could indicate that an unexpected event might have happened causing it to suddenly speed up or slow down.



```
Pattern<DynamicShipClass, DynamicShipClass> speedChange = Pattern.<~>begin("start")
    .where(new SimpleCondition<DynamicShipClass>() {

        @Override
        public boolean filter(DynamicShipClass value) throws Exception {
            return true;
        }
    })
    .next("end").where(new IterativeCondition<DynamicShipClass>() {

        @Override
        public boolean filter(DynamicShipClass value, Context<DynamicShipClass> ctx) throws Exception {
            String contex="start";

            for (DynamicShipClass event : ctx.getEventsForPattern(contex)) {
                return (Math.abs(value.getSpeed() - event.getSpeed()) > 5.0) &&  (Math.abs(value.getSpeed() -
                        event.getSpeed()) < 70.0);
            }
            return false;
        }
    });
```

Code9: pattern for detecting the speed change of a vessel

The following image shows a vessel’s trajectory. The green dots are transmitted messages for which the speed is unchanged, whereas the red dots are messages where the speed has actually changed.

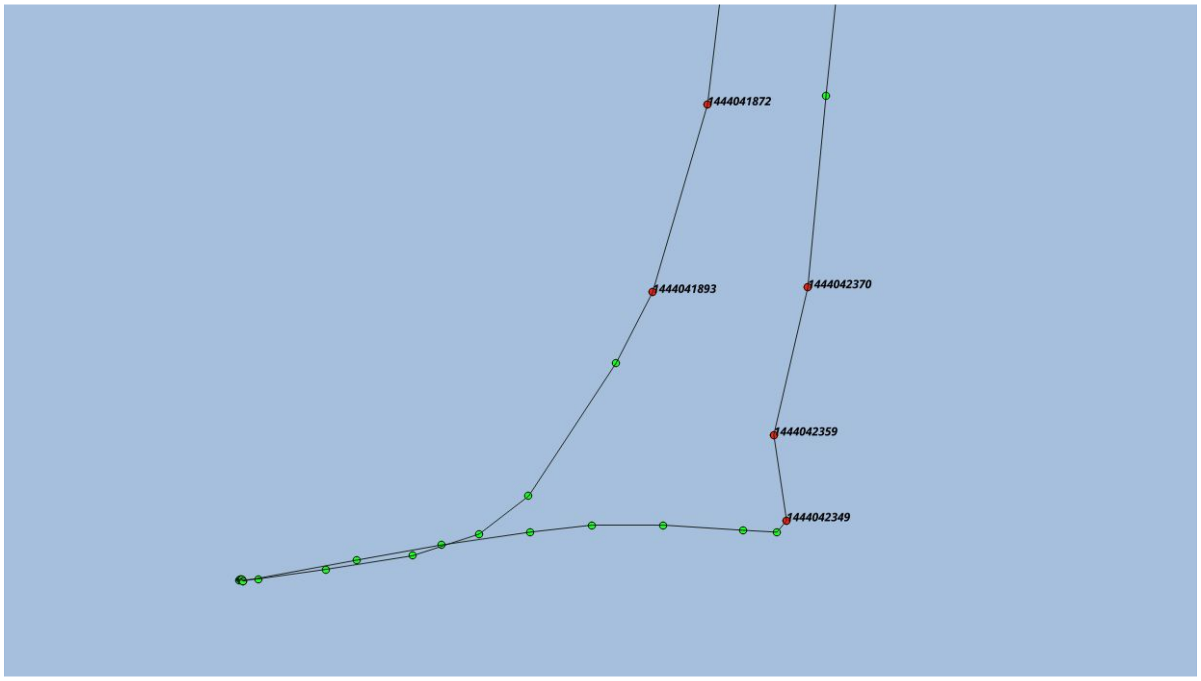
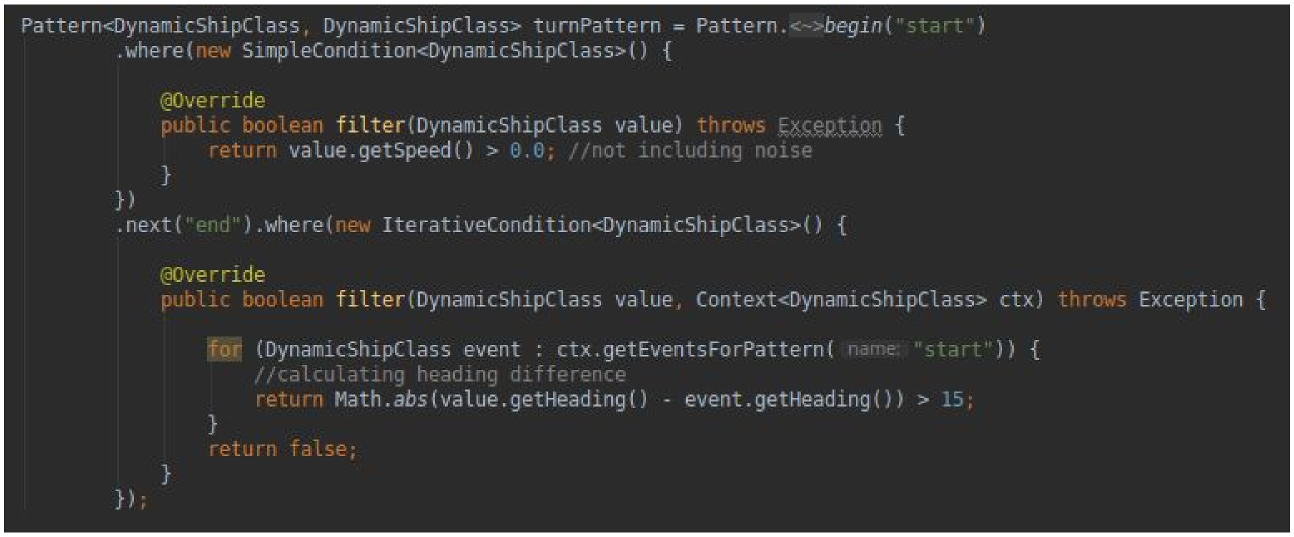


Image9: snapshot of Qgis showing the trajectory of a vessel. 5 speed change events occured

8) Instantaneous turn

For a vessel, and especially a cargo or a tanker ship, the procedure of taking a turn takes a long time and the degrees of turn change gradually over this period. It is interesting to know if a vessel changes its position by more than 15 degrees between two AIS messages transmissions.



```
Pattern<DynamicShipClass, DynamicShipClass> turnPattern = Pattern.<~>begin("start")
        .where(new SimpleCondition<DynamicShipClass>() {

            @Override
            public boolean filter(DynamicShipClass value) throws Exception {
                return value.getSpeed() > 0.0; //not including noise
            }
        })
        .next("end").where(new IterativeCondition<DynamicShipClass>() {

            @Override
            public boolean filter(DynamicShipClass value, Context<DynamicShipClass> ctx) throws Exception {

                for (DynamicShipClass event : ctx.getEventsForPattern( name: "start")) {
                    //calculating heading difference
                    return Math.abs(value.getHeading() - event.getHeading()) > 15;
                }
                return false;
            }
        });
```

Code10: pattern for detecting the instantaneous turn of a vessel

Of course, a lot of consecutive events of the above type can be considered to be an instantaneous turn in the sense that a the vessel keeps turning and its position changes by 90 or degrees in a short period of time. An example of such a turn is shown on the image below.

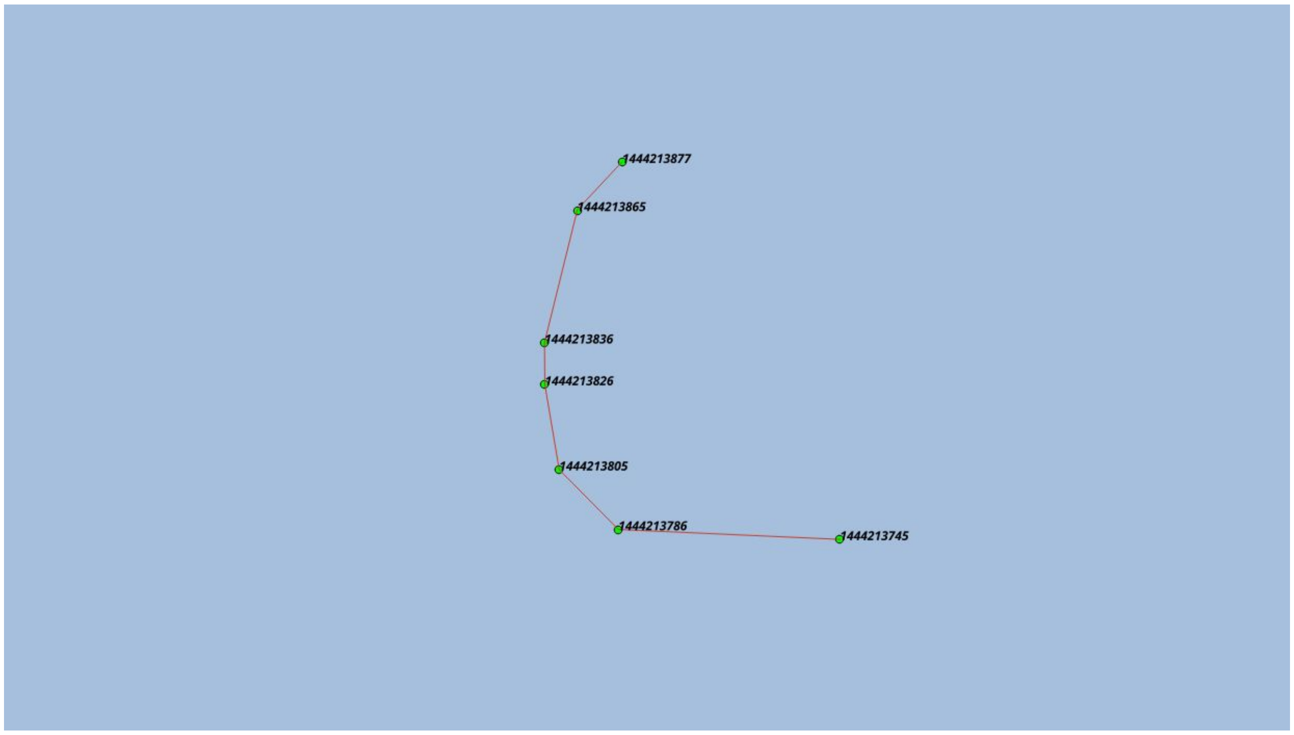


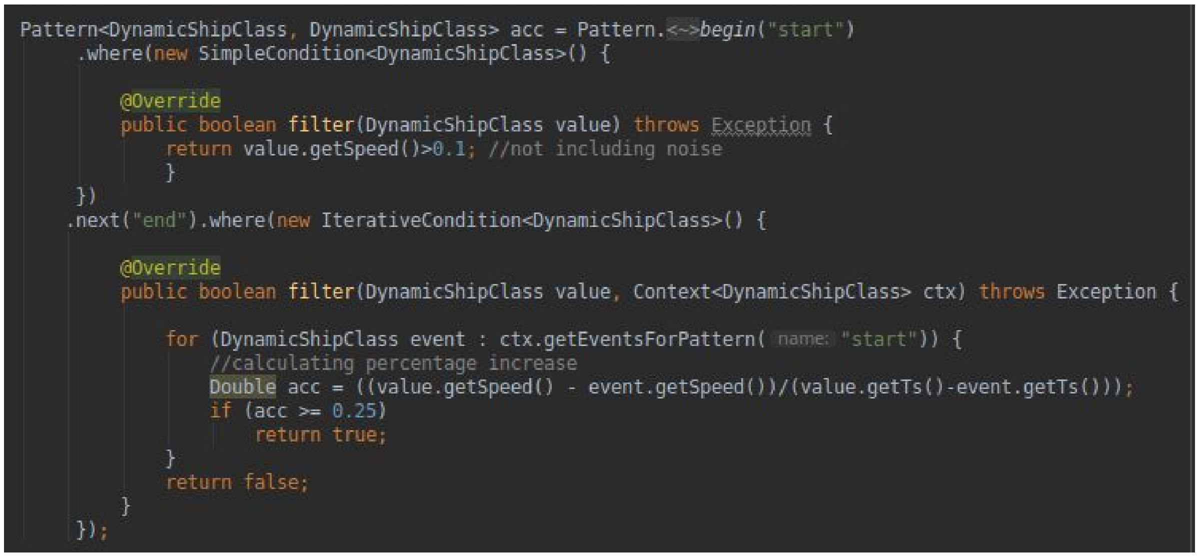
Image10: snapshot of Qgis showing the trajectory of a vessel performing an instantaneous turn

9) Acceleration

Acceleration of a vessel, is called a derivative of its speed relative to time.

$$\mathbf{a}(t) = \frac{\mathrm{d}\,\mathbf{v}(t)}{\mathrm{d}t}$$

The next pattern takes into consideration the vessels that are travelling with speed more than 0.1 knots and tries to find the ones that perform an acceleration in their speed by measuring the above function and checking if the result is more than 0.25 knots per second.



```
Pattern<DynamicShipClass, DynamicShipClass> acc = Pattern.<~>begin("start")
     .where(new SimpleCondition<DynamicShipClass>() {

         @Override
         public boolean filter(DynamicShipClass value) throws Exception {
             return value.getSpeed()>0.1; //not including noise
             }
     })
    .next("end").where(new IterativeCondition<DynamicShipClass>() {

         @Override
         public boolean filter(DynamicShipClass value, Context<DynamicShipClass> ctx) throws Exception {

             for (DynamicShipClass event : ctx.getEventsForPattern( name: "start")) {
                 //calculating percentage increase
                 Double acc = ((value.getSpeed() - event.getSpeed())/(value.getTs()-event.getTs()));
                 if (acc >= 0.25)
                     return true;
             }
             return false;
         }
     });
```

Code11: pattern for detecting when a vessel accelerates

An example of a vessel with accelerating speed travelling away from the shore is shown on the image below.

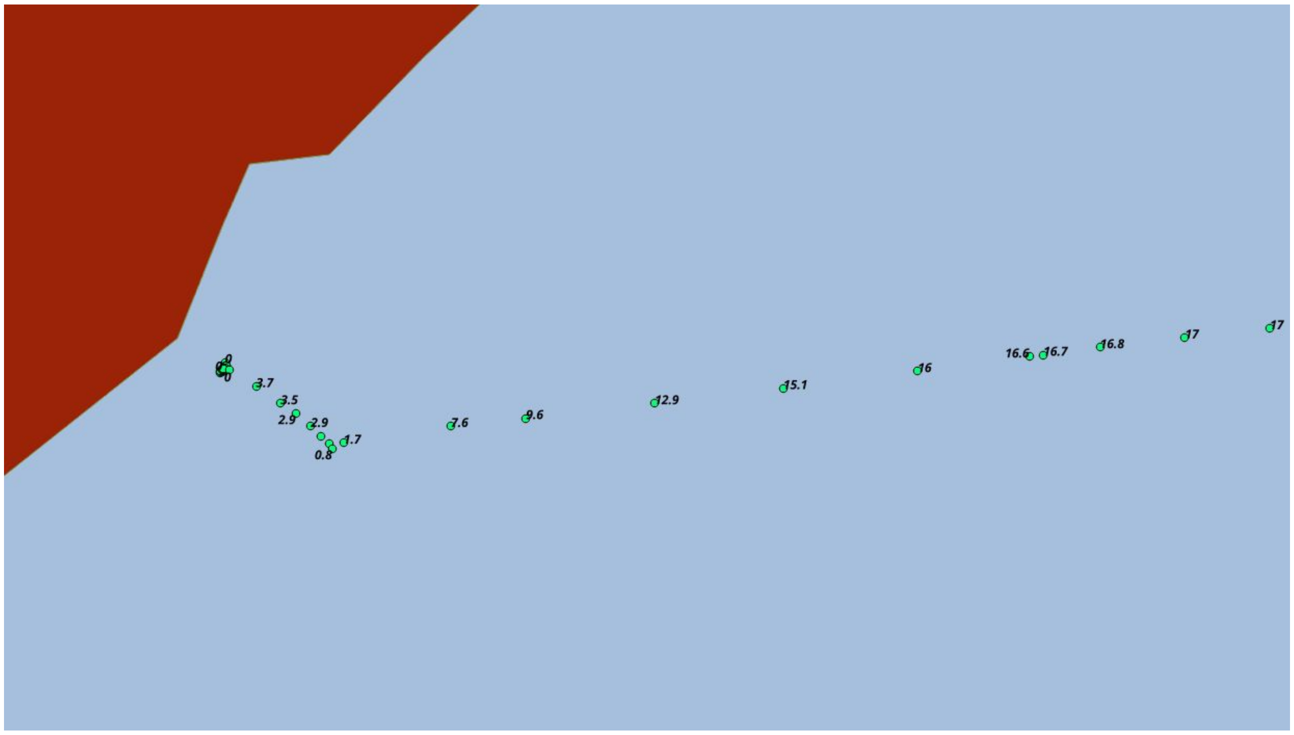
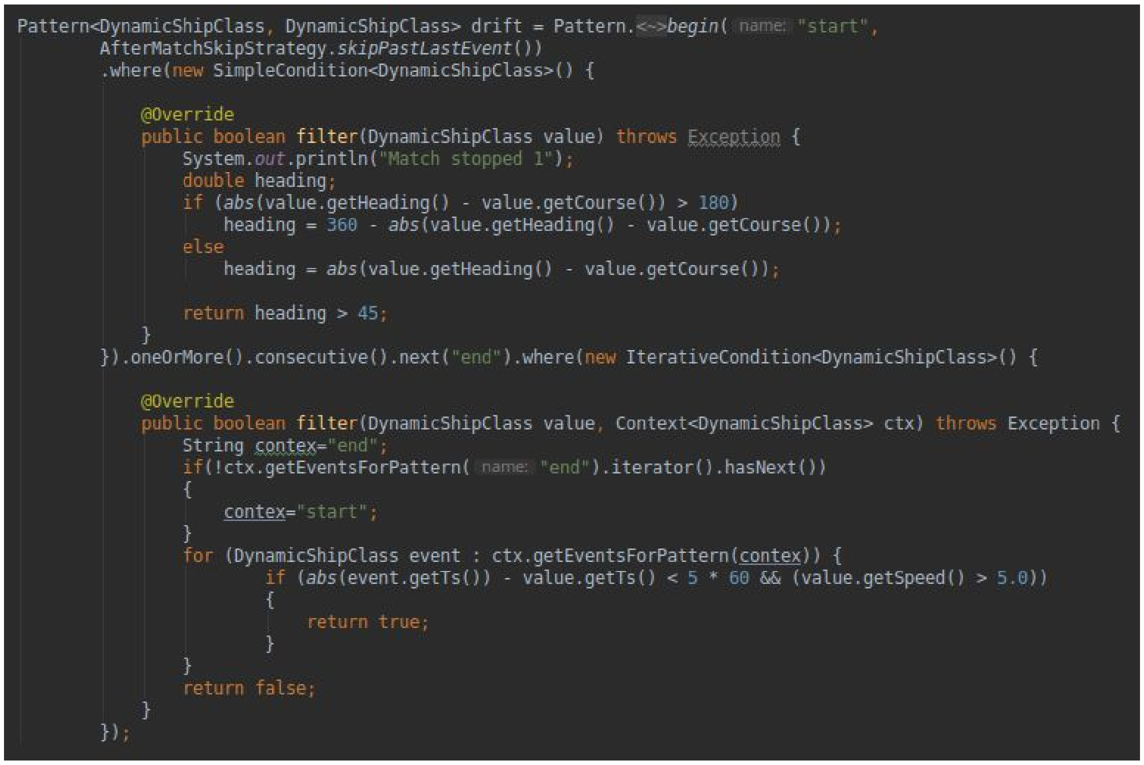


Image11: snapshot of Qgis showing a vessel with accelerated speed moving away from shore

10) Drifting

The meaning of drifting for a vessel, is to oscillate randomly about a position or mode of operation. In the open sea, this can be caused by the wind and the tide and has as a consequence the change in a vessel's course. If the weather conditions are extreme, the drifting may even cause vessels' collision and it is important to gain this kind of knowledge. In our pattern the vessels with a speed of more than 5 knots and a difference in their course and heading of 45 degrees in a period of 5 minutes are detected to suggest that the vessel might be drifting from its normal course.



```
Pattern<DynamicShipClass, DynamicShipClass> drift = Pattern.<~>begin( name: "start",
        AfterMatchSkipStrategy.skipPastLastEvent())
        .where(new SimpleCondition<DynamicShipClass>() {

            @Override
            public boolean filter(DynamicShipClass value) throws Exception {
                System.out.println("Match stopped 1");
                double heading;
                if (abs(value.getHeading() - value.getCourse()) > 180)
                    heading = 360 - abs(value.getHeading() - value.getCourse());
                else
                    heading = abs(value.getHeading() - value.getCourse());

                return heading > 45;
            }
        }).oneOrMore().consecutive().next("end").where(new IterativeCondition<DynamicShipClass>() {

            @Override
            public boolean filter(DynamicShipClass value, Context<DynamicShipClass> ctx) throws Exception {
                String contex="end";
                if(!ctx.getEventsForPattern( name: "end").iterator().hasNext())
                {
                    contex="start";
                }
                for (DynamicShipClass event : ctx.getEventsForPattern(contex)) {
                        if (abs(event.getTs()) - value.getTs() < 5 * 60 && (value.getSpeed() > 5.0))
                        {
                            return true;
                        }
                }
                return false;
            }
        });
```

Code12: pattern for detecting the drifting of a vessel

The images below show the difference between course and heading in the same snapshot of a vessel moving with speed between 4 to 5 knots.

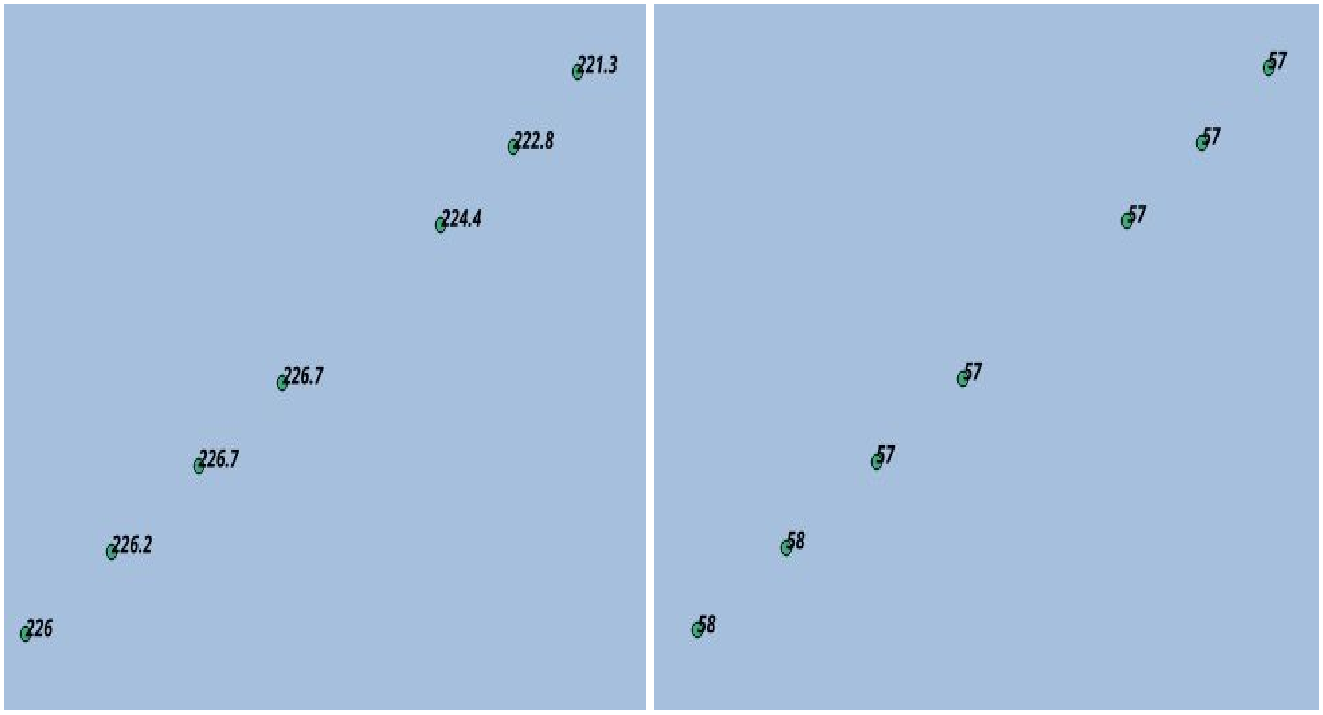
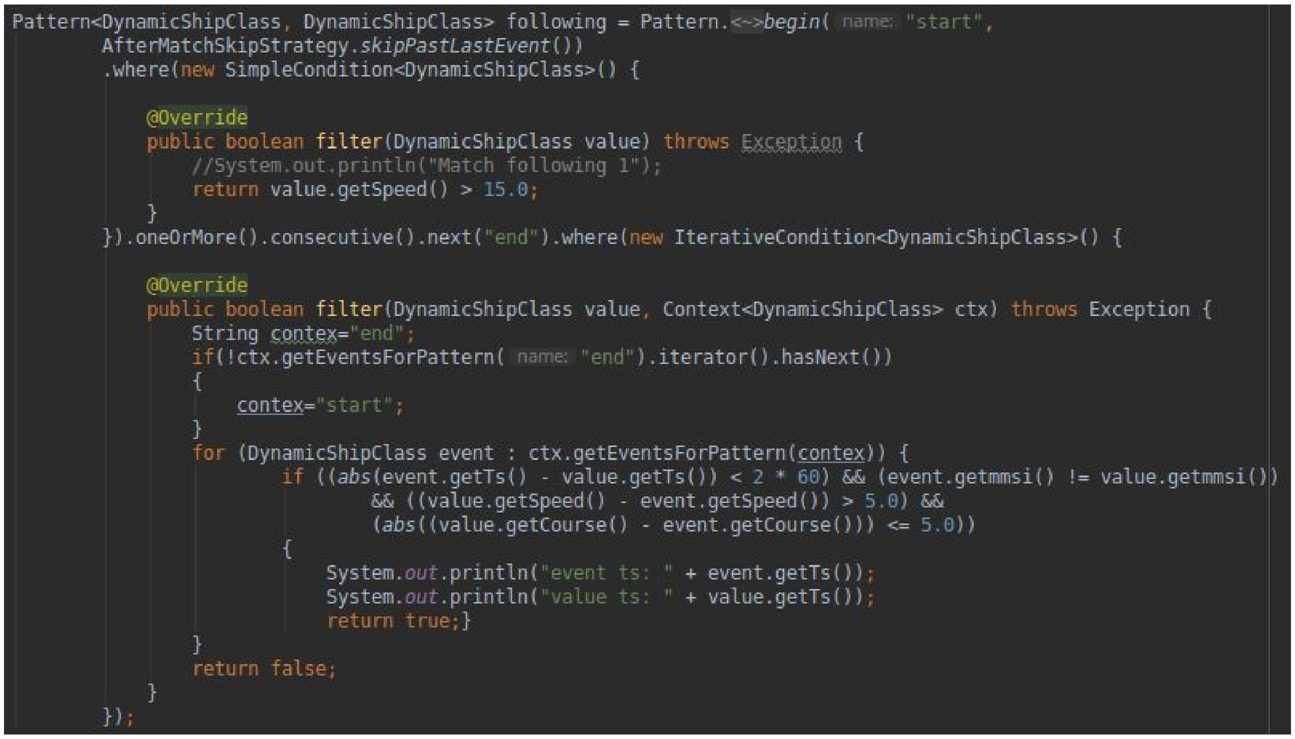


Image12: snapshots of Qgis showing the difference in the course over ground and the true heading of a vessel travelling with speed between 4 to 5 knots

11) Fast Approach

The case of a vessel following another vessel with high speed must be investigated, because it could suggest that something illegal is going to happen, like imminent piracy of the followed vessel. Of course, this could not be the case in our dataset, because there are no incidents of piracy in the wide marine area of Brittany, but it could be useful for monitoring the vessels in other, more dangerous areas, like the marine area of Somalia in the Indian Ocean.

So, the following pattern takes a stream of messages, keeps the vessels that are travelling with speed higher than 15 knots and with the use of the KeyBy operator, it creates logical and disjoint partitions of the stream according to the grid id. Then the pattern tries to detect whether there are two different vessels in the same grid id, in a period of 2 minutes. Then their difference in speed should be more than 5 knots so as for the second to be approaching the first and their course should be the same.



```java
Pattern<DynamicShipClass, DynamicShipClass> following = Pattern.<~>begin( name: "start",
        AfterMatchSkipStrategy.skipPastLastEvent())
        .where(new SimpleCondition<DynamicShipClass>() {

            @Override
            public boolean filter(DynamicShipClass value) throws Exception {
                //System.out.println("Match following 1");
                return value.getSpeed() > 15.0;
            }
        }).oneOrMore().consecutive().next("end").where(new IterativeCondition<DynamicShipClass>() {

            @Override
            public boolean filter(DynamicShipClass value, Context<DynamicShipClass> ctx) throws Exception {
                String contex="end";
                if(!ctx.getEventsForPattern( name: "end").iterator().hasNext())
                {
                    contex="start";
                }
                for (DynamicShipClass event : ctx.getEventsForPattern(contex)) {
                        if ((abs(event.getTs() - value.getTs()) < 2 * 60) && (event.getmmsi() != value.getmmsi())
                                && ((value.getSpeed() - event.getSpeed()) > 5.0) &&
                                (abs((value.getCourse() - event.getCourse())) <= 5.0))
                        {
                            System.out.println("event ts: " + event.getTs());
                            System.out.println("value ts: " + value.getTs());
                            return true;}
                }
                return false;
            }
        });
```

Code13: pattern for detecting a vessel being followed by another vessel

The image below shows two pairs of ships that the first seems to be following the second. Of course, this might be a coincidence, but it has to be further investigated. The same pattern can be modified to detect vessels that approaching one another with high speed, by changing the course factor to be counter-diametric instead of almost the same.

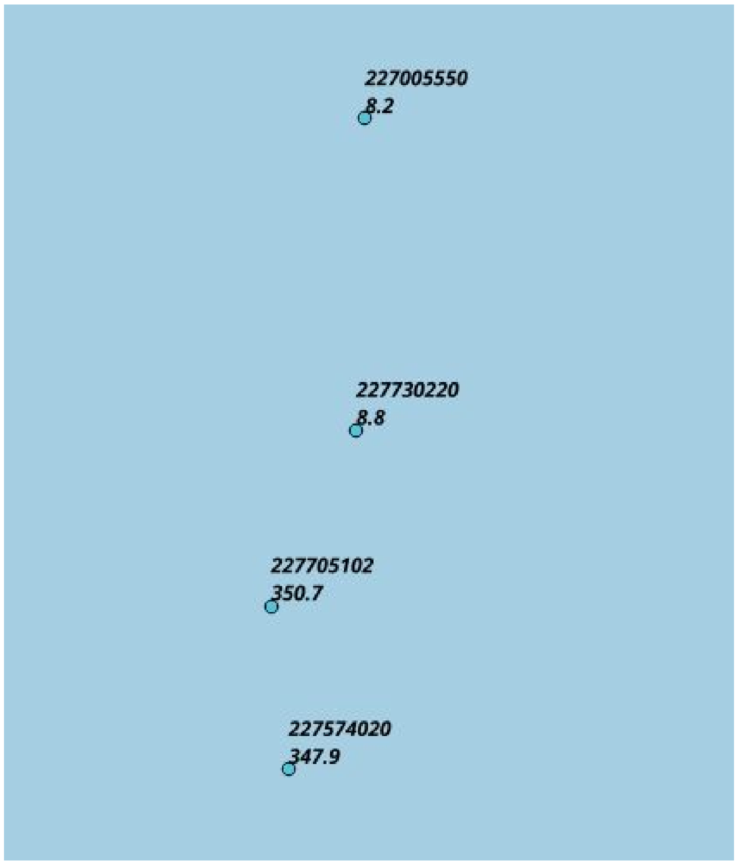
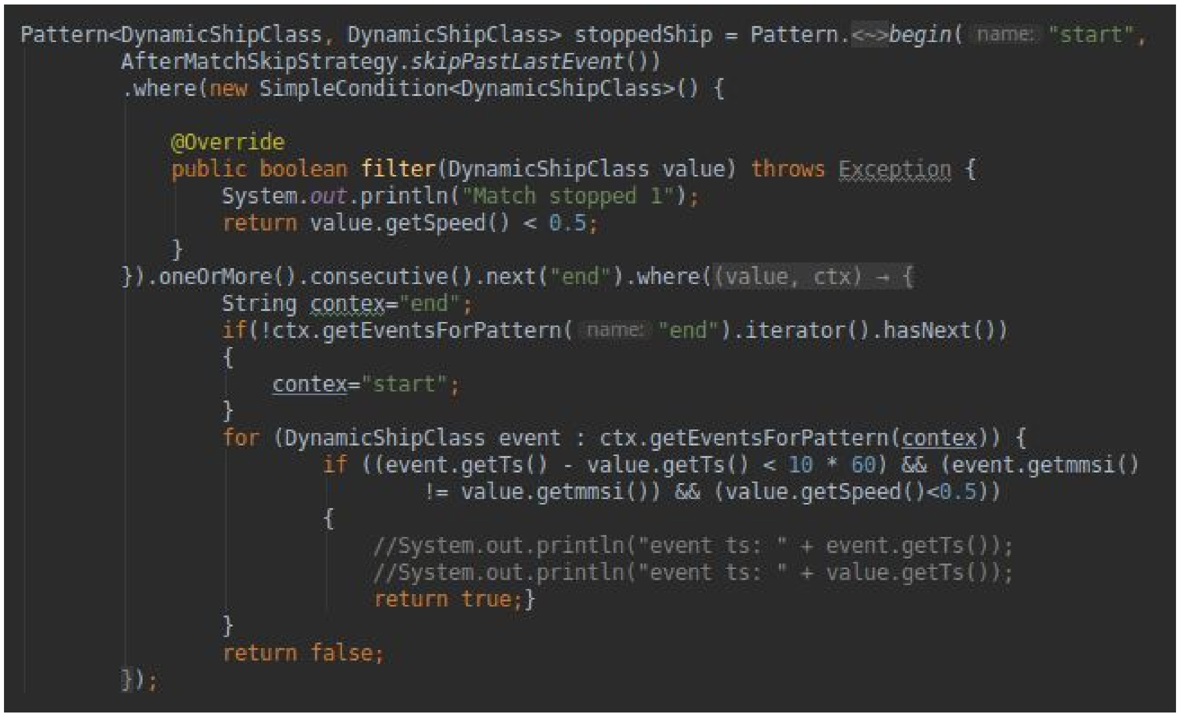


Image13: snapshots of Qgis showing two pairs of vessels detected from the following complex pattern. The first label is their mmsi showing they are different vessels and the second their course, which is almost the same

12) Rendez-vous

Another case of interest is the rendez-vous of two vessels in the open sea or even close to the shore, where no port exists. This is the case where something illegal might happen, like exchange of goods, petroleum selling or even illegal immigration.

The following pattern ingests a same to the previous stream (keyed by grid id) and tries to detect two different vessels in stopped (speed less than 0.5 knots) in the same grid id, in a period less than 10 minutes.



```
Pattern<DynamicShipClass, DynamicShipClass> stoppedShip = Pattern.<~>begin( name: "start",
        AfterMatchSkipStrategy.skipPastLastEvent())
        .where(new SimpleCondition<DynamicShipClass>() {

            @Override
            public boolean filter(DynamicShipClass value) throws Exception {
                System.out.println("Match stopped 1");
                return value.getSpeed() < 0.5;
            }
        }).oneOrMore().consecutive().next("end").where((value, ctx) → {
                String contex="end";
                if(!ctx.getEventsForPattern( name: "end").iterator().hasNext())
                {
                    contex="start";
                }
                for (DynamicShipClass event : ctx.getEventsForPattern(contex)) {
                        if ((event.getTs() - value.getTs() < 10 * 60) && (event.getmmsi()
                                != value.getmmsi()) && (value.getSpeed()<0.5))
                        {
                            //System.out.println("event ts: " + event.getTs());
                            //System.out.println("event ts: " + value.getTs());
                            return true;}
                }
                return false;
        });
```

Code14: pattern for detecting the rendez-vous of two vessels

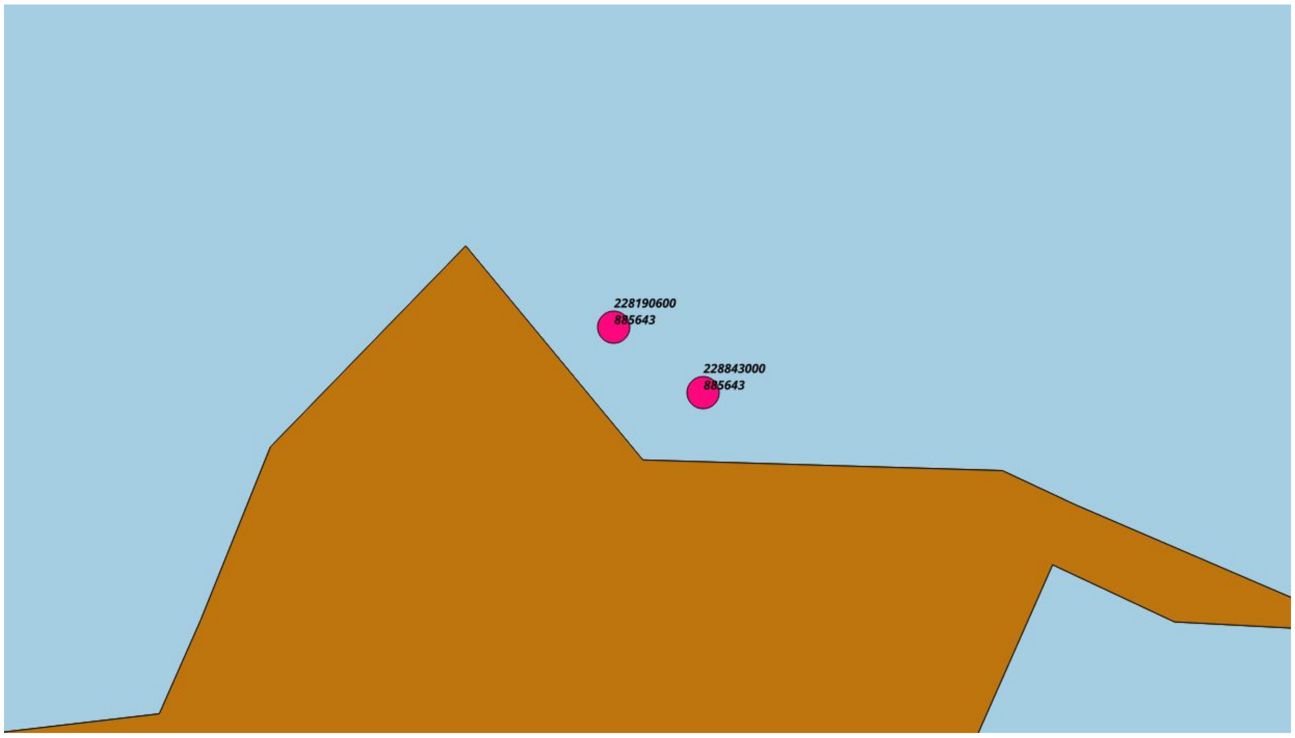
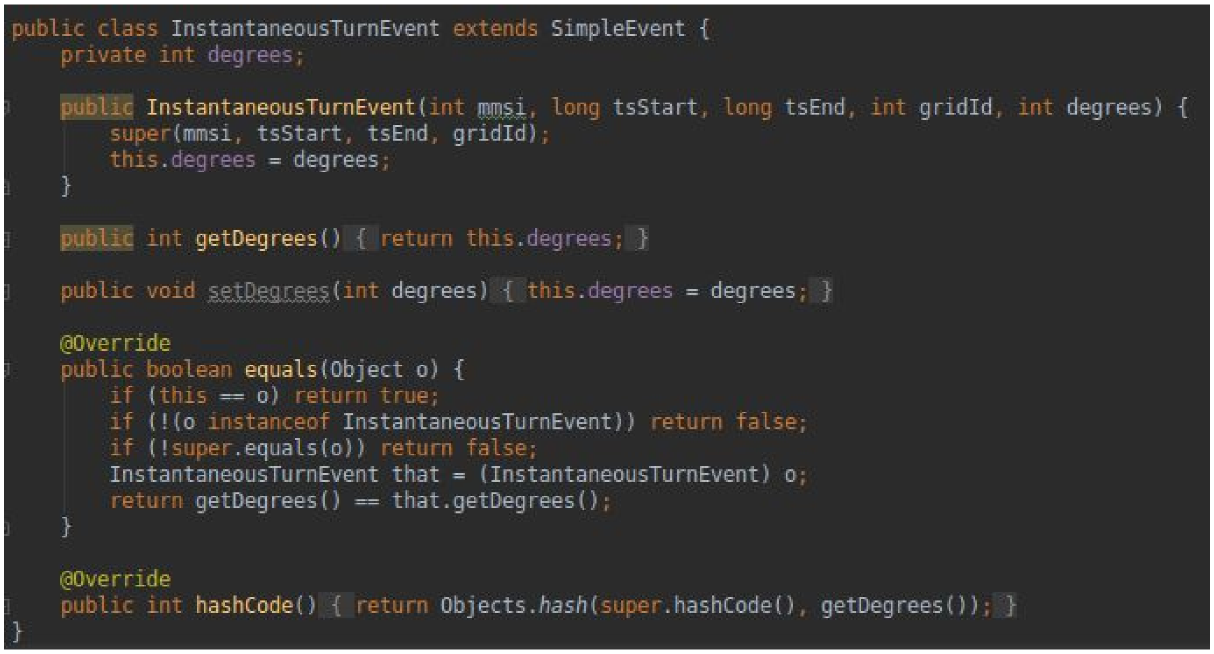


Image14: snapshots of Qgis showing two vessels in the same grid having zero speed during a period of 10 minutes.

Complex Event Processing

FlinkCEP allows us to create more complex patterns, by unifying two or more simple pattern matchings. In order to achieve the implementation of a complex event, the needed classes to describe the type of events - critical points that are extracted and used. Each one of these extra classes, in which the simple events are stored, contain the needed information of the matching and additionally, an override to the implementations of *equals* and *hashcode* in order to achieve a custom equality mechanism.

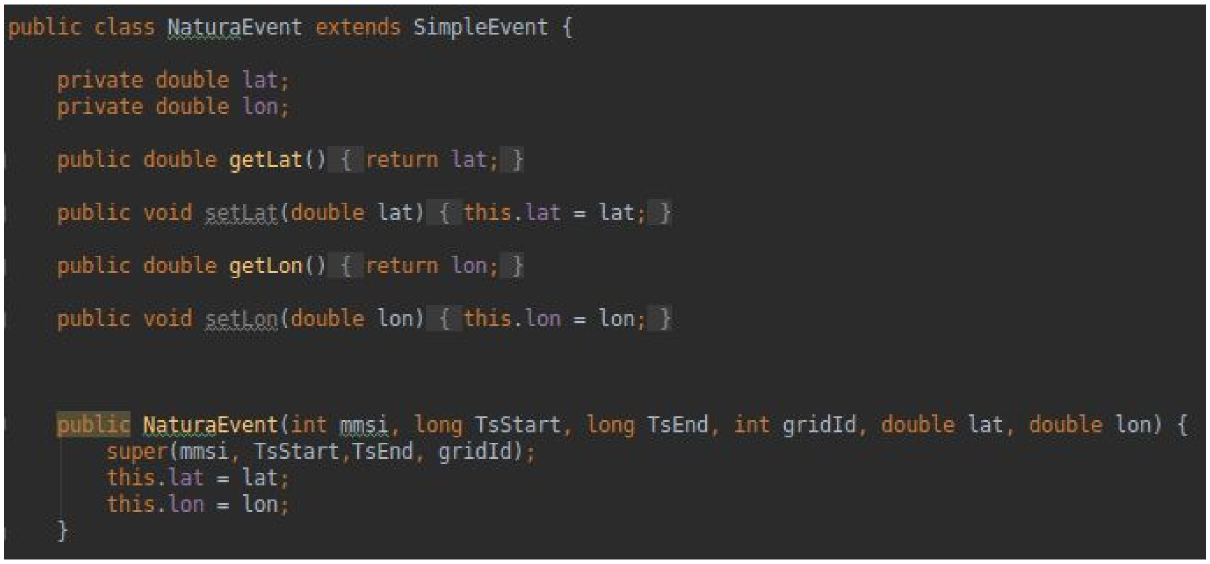
So, the class used to detect complex events that contain an instantaneous turning event is shown below. It contains the mmsi of the vessel found, the two timestamps (for the start timestamp and the end timestamp), the grid id and the degrees of the turn.



```
public class InstantaneousTurnEvent extends SimpleEvent {
    private int degrees;

    public InstantaneousTurnEvent(int mmsi, long tsStart, long tsEnd, int gridId, int degrees) {
        super(mmsi, tsStart, tsEnd, gridId);
        this.degrees = degrees;
    }

    public int getDegrees() { return this.degrees; }

    public void setDegrees(int degrees) { this.degrees = degrees; }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof InstantaneousTurnEvent)) return false;
        if (!super.equals(o)) return false;
        InstantaneousTurnEvent that = (InstantaneousTurnEvent) o;
        return getDegrees() == that.getDegrees();
    }

    @Override
    public int hashCode() { return Objects.hash(super.hashCode(), getDegrees()); }
}
```

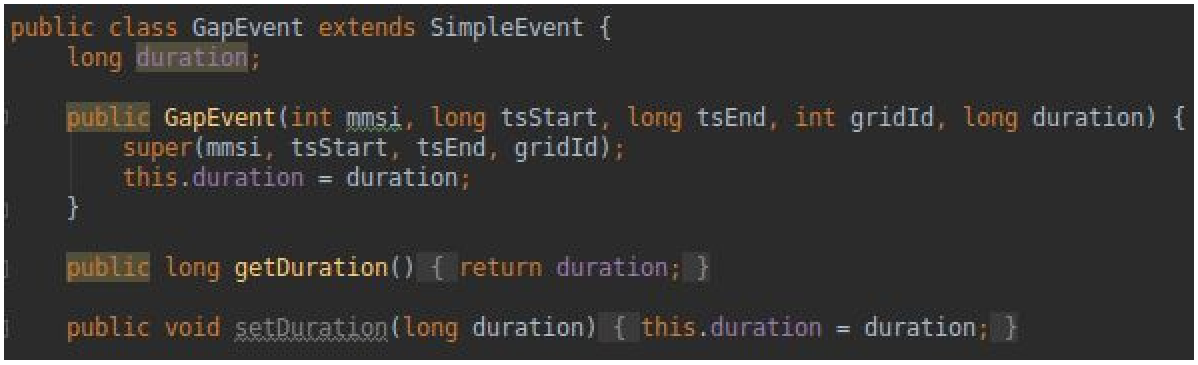
Code15: class for storing the simple instantaneous turn events of a vessel

Respectively the classes for natura events, communication gap events, low speed events and stopped events are shown below.



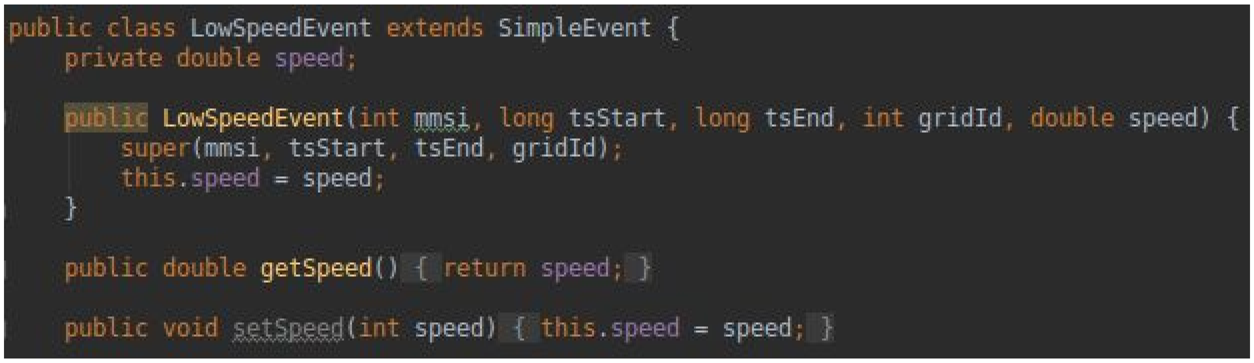
```java
public class NaturaEvent extends SimpleEvent {

    private double lat;
    private double lon;

    public double getLat() { return lat; }

    public void setLat(double lat) { this.lat = lat; }

    public double getLon() { return lon; }

    public void setLon(double lon) { this.lon = lon; }



    public NaturaEvent(int mmsi, long TsStart, long TsEnd, int gridId, double lat, double lon) {
        super(mmsi, TsStart,TsEnd, gridId);
        this.lat = lat;
        this.lon = lon;
    }
```

Code16: class for storing the simple travelling in natura area events of a vessel



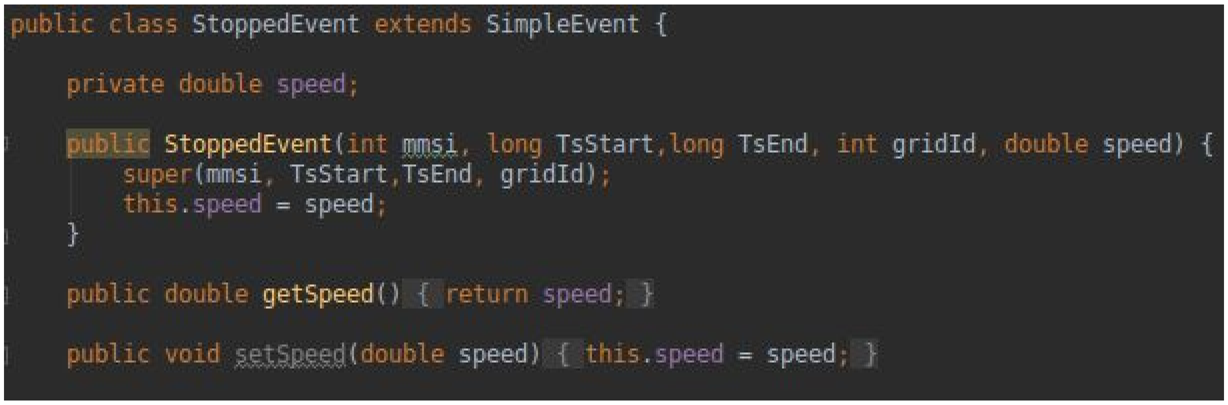
```java
public class GapEvent extends SimpleEvent {
    long duration;

    public GapEvent(int mmsi, long tsStart, long tsEnd, int gridId, long duration) {
        super(mmsi, tsStart, tsEnd, gridId);
        this.duration = duration;
    }

    public long getDuration() { return duration; }

    public void setDuration(long duration) { this.duration = duration; }
```

Code17: class for storing the simple communication gap events of a vessel



```java
public class LowSpeedEvent extends SimpleEvent {
    private double speed;

    public LowSpeedEvent(int mmsi, long tsStart, long tsEnd, int gridId, double speed) {
        super(mmsi, tsStart, tsEnd, gridId);
        this.speed = speed;
    }

    public double getSpeed() { return speed; }

    public void setSpeed(int speed) { this.speed = speed; }
```

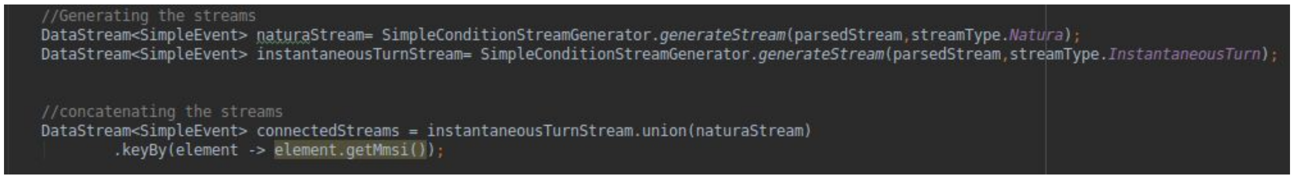
Code18: class for storing the simple low speed events of a vessel



```
public class StoppedEvent extends SimpleEvent {

    private double speed;

    public StoppedEvent(int mmsi, long TsStart,long TsEnd, int gridId, double speed) {
        super(mmsi, TsStart,TsEnd, gridId);
        this.speed = speed;
    }

    public double getSpeed() { return speed; }

    public void setSpeed(double speed) { this.speed = speed; }
```

Code19: class for storing the simple stopped events of a vessel

In order to process more complex events we have to feed our pattern with a datastream containing the above simple events. After the critical points recognition, we end up with many datastreams, which we want to concatenate using the union operator. For example the line below shows the union of two simple event types (instantaneous turn and natura events) in one datastream.



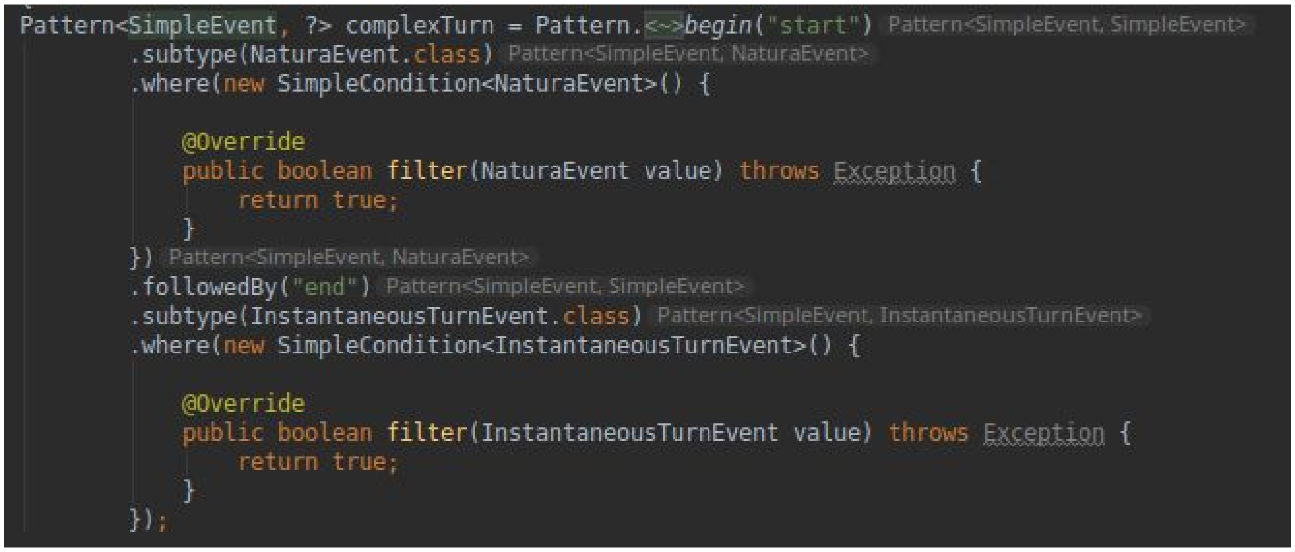
```
    //Generating the streams
    DataStream<SimpleEvent> naturaStream= SimpleConditionStreamGenerator.generateStream(parsedStream,streamType.Natura);
    DataStream<SimpleEvent> instantaneousTurnStream= SimpleConditionStreamGenerator.generateStream(parsedStream,streamType.InstantaneousTurn);


    //concatenating the streams
    DataStream<SimpleEvent> connectedStreams = instantaneousTurnStream.union(naturaStream)
            .keyBy(element -> element.getMmsi());
```

Code20: code for unifying simple event types into one datastream

Then the unified datastream is fed into the complex event patterns to proceed with the matching phase. The same applies to all of the complex event patterns, each time using the needed unified datastream.

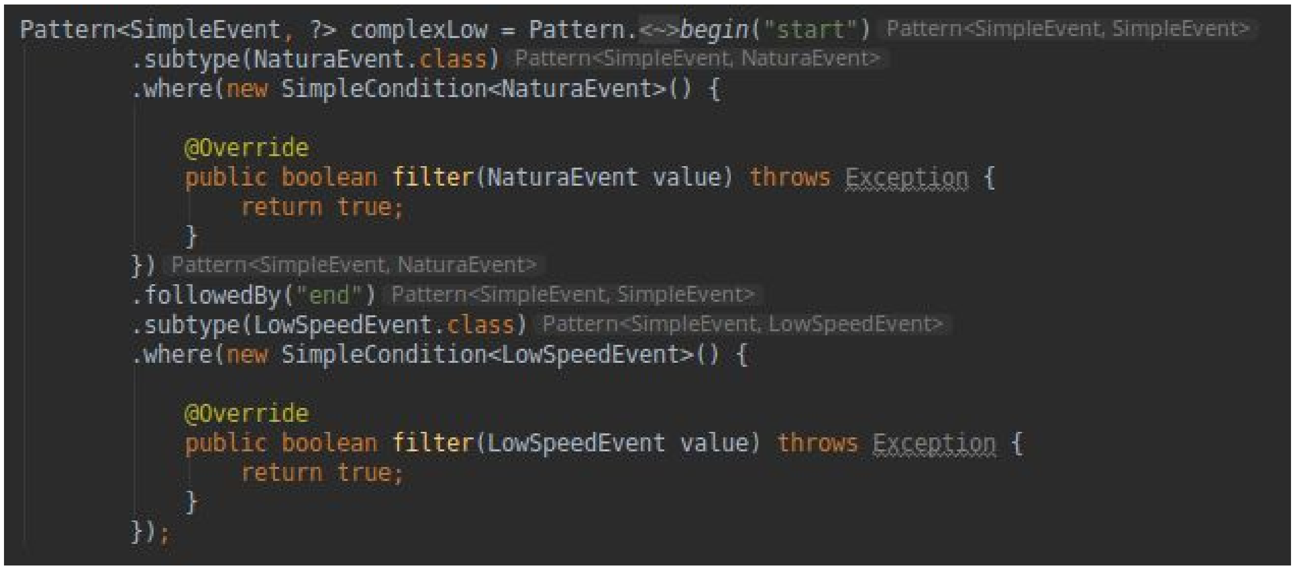
1) Illegal Fishing

We can say that Illegal fishing occurs when vessels are in protected areas and we can find that there existed a communication gap event or a low speed event or a turn event. To recognise these moves we created the patterns below.



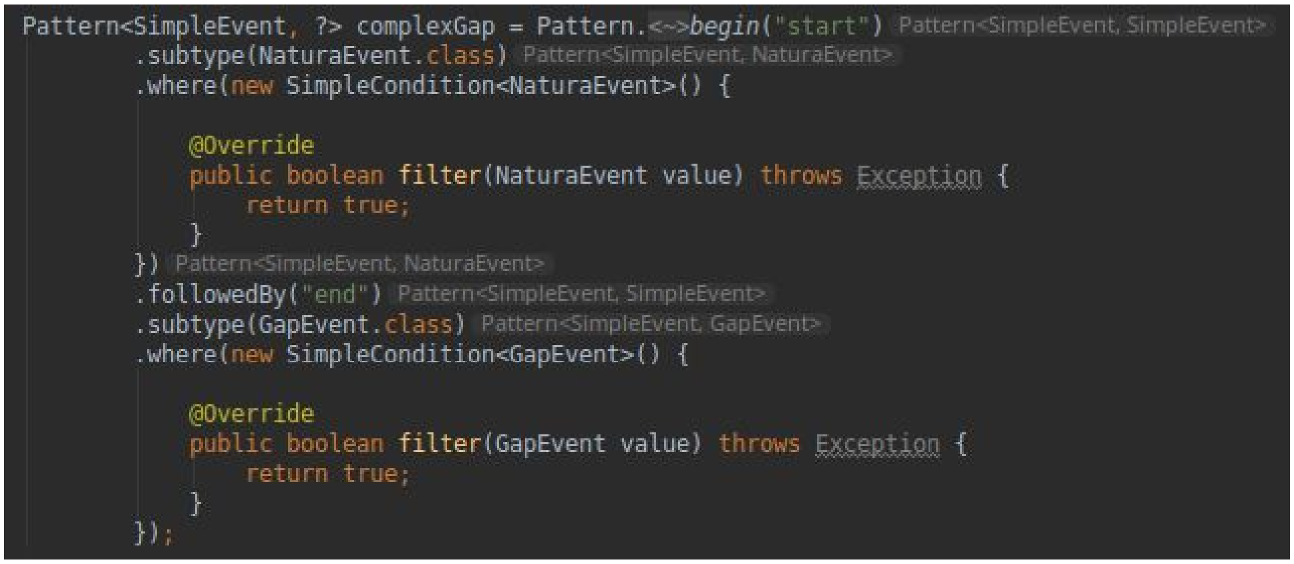
```
Pattern<SimpleEvent, ?> complexTurn = Pattern.<~>begin("start") Pattern<SimpleEvent, SimpleEvent>
        .subtype(NaturaEvent.class) Pattern<SimpleEvent, NaturaEvent>
        .where(new SimpleCondition<NaturaEvent>() {

            @Override
            public boolean filter(NaturaEvent value) throws Exception {
                return true;
            }
        }) Pattern<SimpleEvent, NaturaEvent>
        .followedBy("end") Pattern<SimpleEvent, SimpleEvent>
        .subtype(InstantaneousTurnEvent.class) Pattern<SimpleEvent, InstantaneousTurnEvent>
        .where(new SimpleCondition<InstantaneousTurnEvent>() {

            @Override
            public boolean filter(InstantaneousTurnEvent value) throws Exception {
                return true;
            }
        });
```

Code21: pattern for detecting an illegal fishing event (1st way)



```
Pattern<SimpleEvent, ?> complexLow = Pattern.<~>begin("start") Pattern<SimpleEvent, SimpleEvent>
        .subtype(NaturaEvent.class) Pattern<SimpleEvent, NaturaEvent>
        .where(new SimpleCondition<NaturaEvent>() {

            @Override
            public boolean filter(NaturaEvent value) throws Exception {
                return true;
            }
        }) Pattern<SimpleEvent, NaturaEvent>
        .followedBy("end") Pattern<SimpleEvent, SimpleEvent>
        .subtype(LowSpeedEvent.class) Pattern<SimpleEvent, LowSpeedEvent>
        .where(new SimpleCondition<LowSpeedEvent>() {

            @Override
            public boolean filter(LowSpeedEvent value) throws Exception {
                return true;
            }
        });
```

Code22: pattern for detecting an illegal fishing event (2nd way)



```
Pattern<SimpleEvent, ?> complexGap = Pattern.<~>begin("start") Pattern<SimpleEvent, SimpleEvent>
        .subtype(NaturaEvent.class) Pattern<SimpleEvent, NaturaEvent>
        .where(new SimpleCondition<NaturaEvent>() {

            @Override
            public boolean filter(NaturaEvent value) throws Exception {
                return true;
            }
        }) Pattern<SimpleEvent, NaturaEvent>
        .followedBy("end") Pattern<SimpleEvent, SimpleEvent>
        .subtype(GapEvent.class) Pattern<SimpleEvent, GapEvent>
        .where(new SimpleCondition<GapEvent>() {

            @Override
            public boolean filter(GapEvent value) throws Exception {
                return true;
            }
        });
```

Code23: pattern for detecting an illegal fishing event (3rd way)

The output of those patterns contains the results in which a vessel is into a Natura area and lost the communication for more than 10 minutes or made an instantaneous turn more than 15 degrees or its speed is between 0.5 and 5 knots.

The image below shows indicatively one of the above cases. A vessel inside a natura area that has low speed.

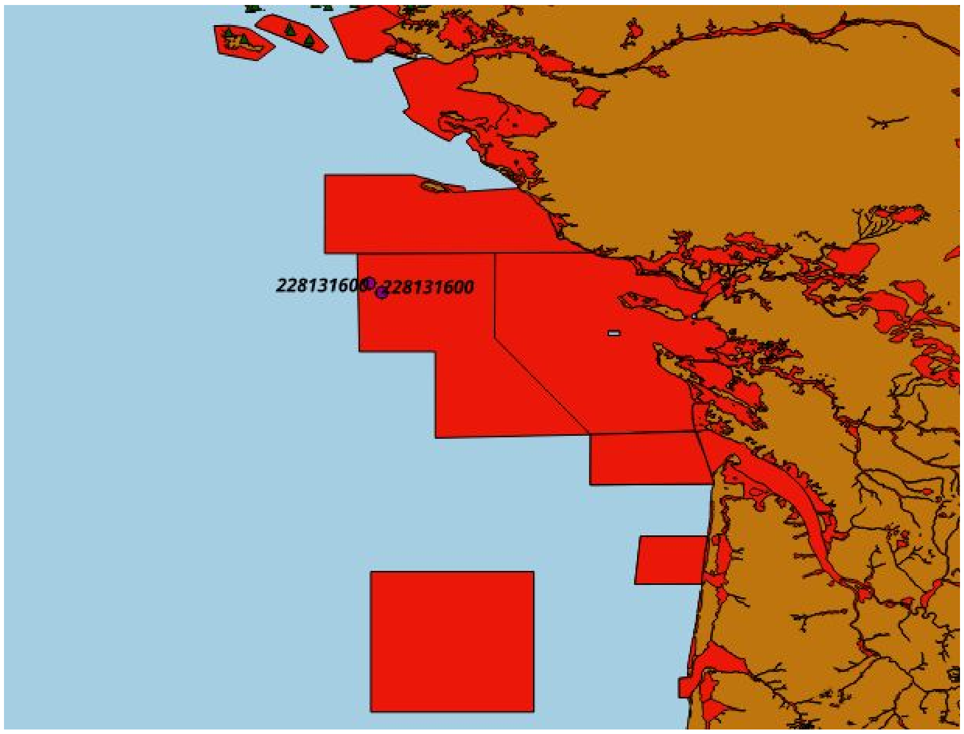
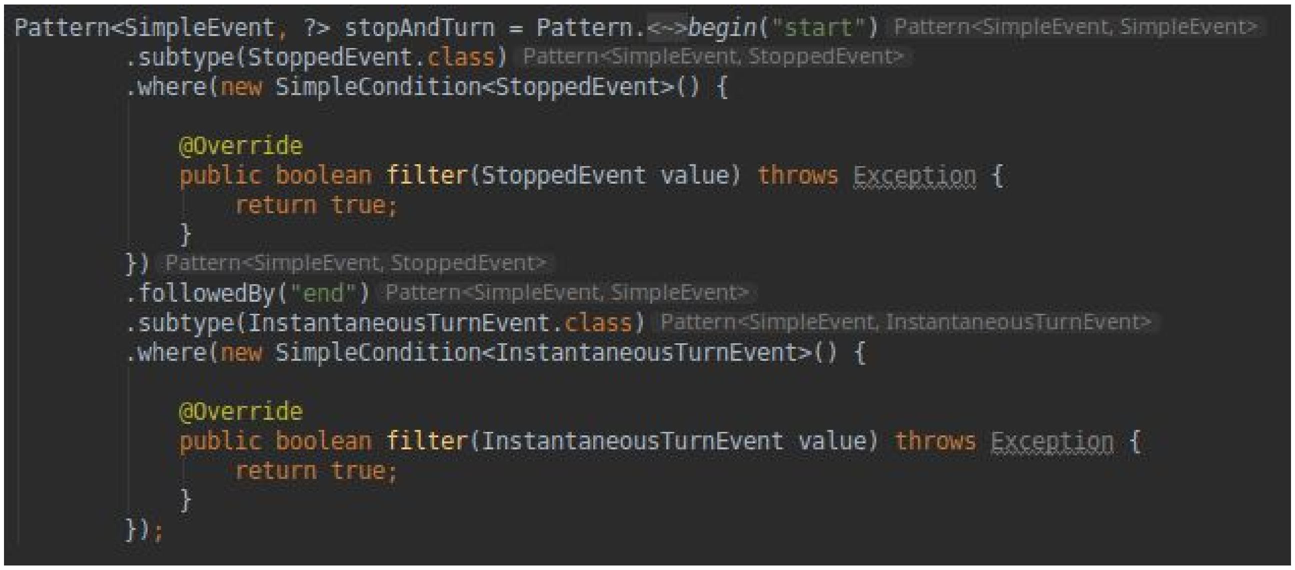


Image15: snapshots of Qgis showing a vessel into a protected area travelling with low speed

2) Stop and turn

Another interesting complex event would be the one that a vessel suddenly stops in the open sea and changes its course, thus taking an instantaneous turn. This might indicate that there is an emergency situation causing the vessel to act this way, like the appearance of bad weather conditions ahead or an obstacle of other kind. It might also imply that the vessel is suddenly recruited to help another vessel in danger.

With the pattern below, this kind of complex event is trying to be detected, by checking whether there is a stopped event followed by an instantaneous turn event.



```java
Pattern<SimpleEvent, ?> stopAndTurn = Pattern.<~>begin("start") Pattern<SimpleEvent, SimpleEvent>
        .subtype(StoppedEvent.class) Pattern<SimpleEvent, StoppedEvent>
        .where(new SimpleCondition<StoppedEvent>() {

            @Override
            public boolean filter(StoppedEvent value) throws Exception {
                return true;
            }
        }) Pattern<SimpleEvent, StoppedEvent>
        .followedBy("end") Pattern<SimpleEvent, SimpleEvent>
        .subtype(InstantaneousTurnEvent.class) Pattern<SimpleEvent, InstantaneousTurnEvent>
        .where(new SimpleCondition<InstantaneousTurnEvent>() {

            @Override
            public boolean filter(InstantaneousTurnEvent value) throws Exception {
                return true;
            }
        });
```

Code24: pattern for detecting a stop and turn event

The image below shows a vessel being in the open sea and matching the above pattern. The first event is a stop event and the second an instantaneous turn.

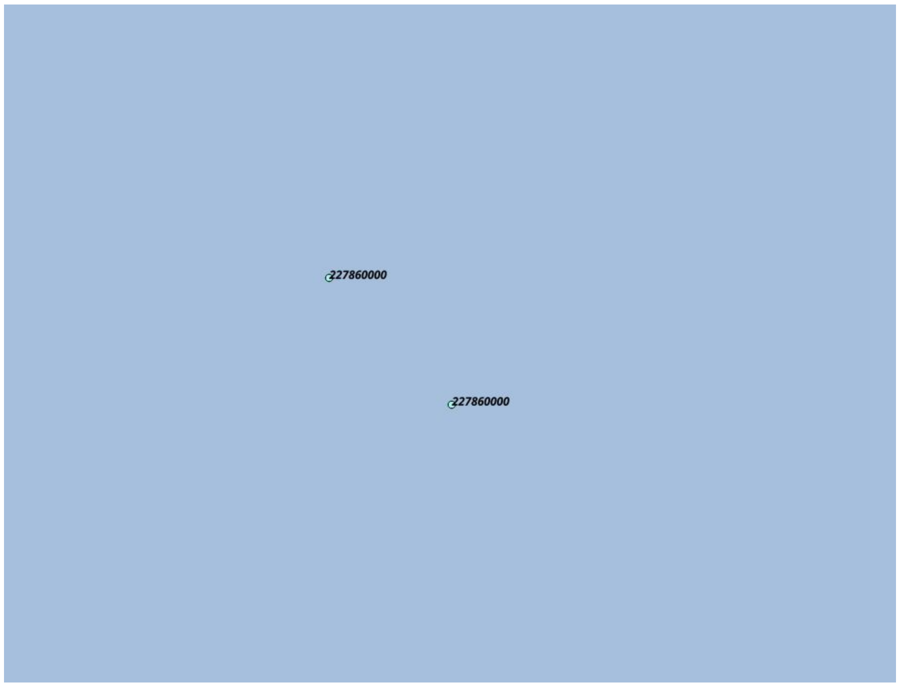
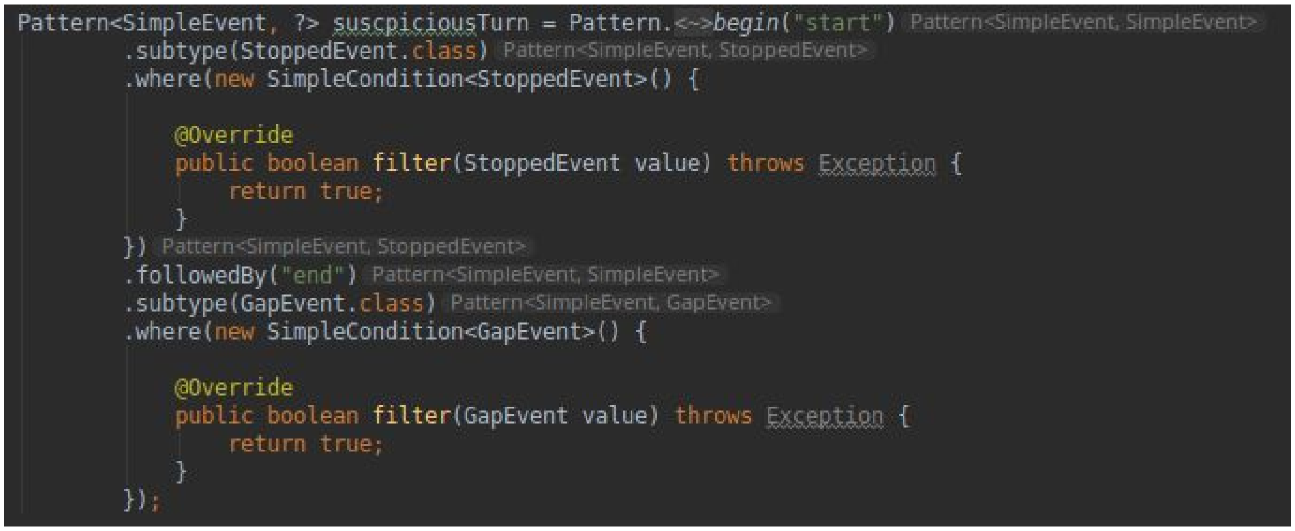


Image16: snapshots of Qgis showing a vessel in a stop and turn CE

3) Suspicious Stop

The notion of suspicious stop is defined as the complex event where a stopped event is followed by a gap event. This is of great importance because the behaviour of a vessel does not normally consists of these consecutive events and it might indicate that something illegal is conducted by the vessel's staff. The following pattern tries to detect this kind of behaviour.



```
Pattern<SimpleEvent, ?> suscpiciousTurn = Pattern.<~>begin("start") Pattern<SimpleEvent, SimpleEvent>
        .subtype(StoppedEvent.class) Pattern<SimpleEvent, StoppedEvent>
        .where(new SimpleCondition<StoppedEvent>() {

            @Override
            public boolean filter(StoppedEvent value) throws Exception {
                return true;
            }
        }) Pattern<SimpleEvent, StoppedEvent>
        .followedBy("end") Pattern<SimpleEvent, SimpleEvent>
        .subtype(GapEvent.class) Pattern<SimpleEvent, GapEvent>
        .where(new SimpleCondition<GapEvent>() {

            @Override
            public boolean filter(GapEvent value) throws Exception {
                return true;
            }
        });
```

Code25: pattern for detecting a suspicious stop event

After applying the above pattern to the stream, no results are produced, which means that in the part of the file that we examine there are no events that could be characterized as suspicious stops.

Part II: Logic-based CER with RTEC

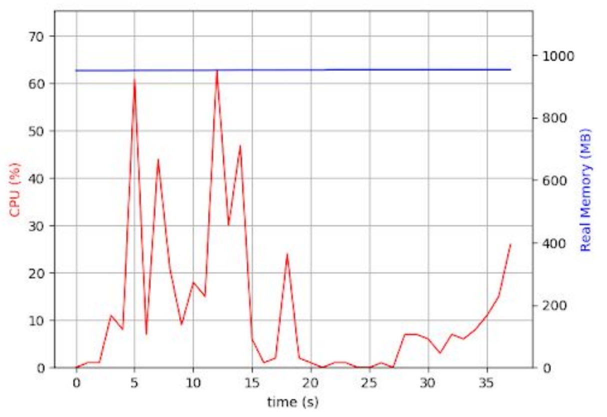
Introduction

The Event Calculus for Run-Time reasoning (RTEC) is an implementation of the Event Calculus, based on logic programming. It is created for complex event (CE) recognition on data streams through continuous narrative queries. RTEC's model of time is linear and uses time points.

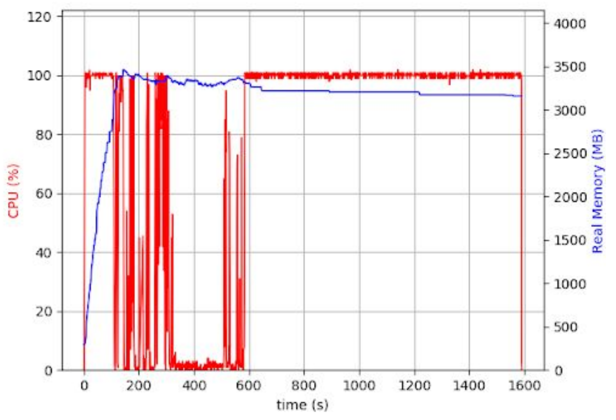
The implementation of RTEC for the maritime monitoring uses a preprocessed file of the events, in which simple fluents have been extracted. In order to compare the two systems, from the preprocessed file we keep the lines which correspond to the same period as the file we used for the Flink implementation. Thus, we choose to keep the lines in which the timestamps are the same as in the previous file.

The goal is to compare the two implementations' performance on the same amount of data. We execute the RTEC files and the corresponding Flink files for the same complex event recognition task. At the same time, we monitor CPU, Real Memory and Relative Execution Time for each of the executions and get the visualizations below.

Low Speed

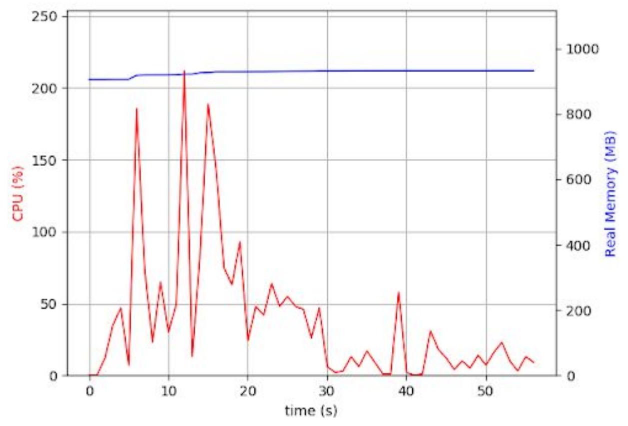


FlinkCEP

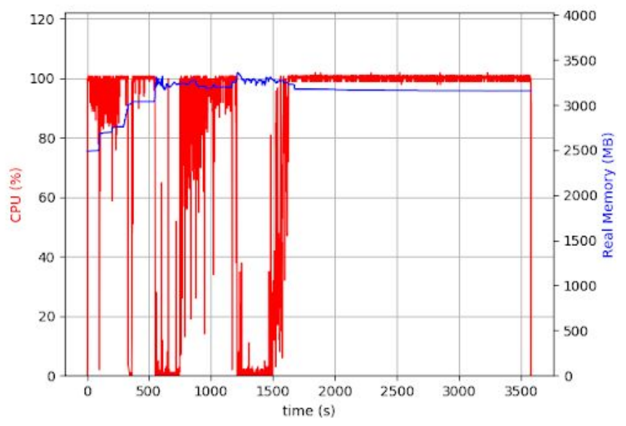


RTEC

Communication Gaps

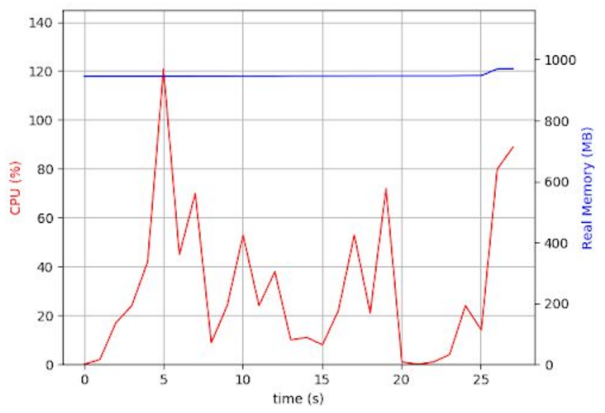


FlinkCEP

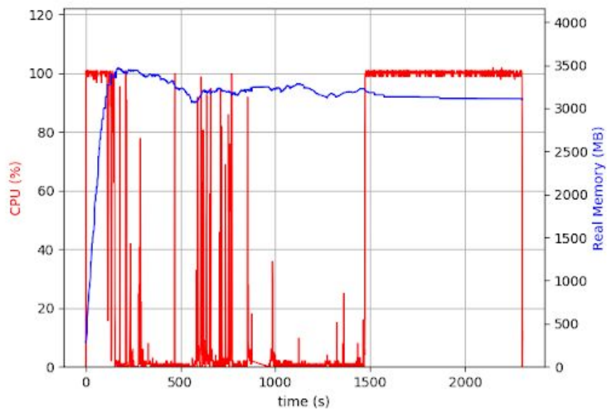


RTEC

Stopped Events

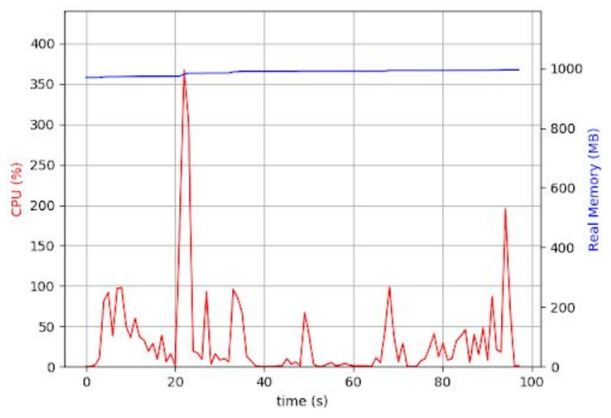


FlinkCEP

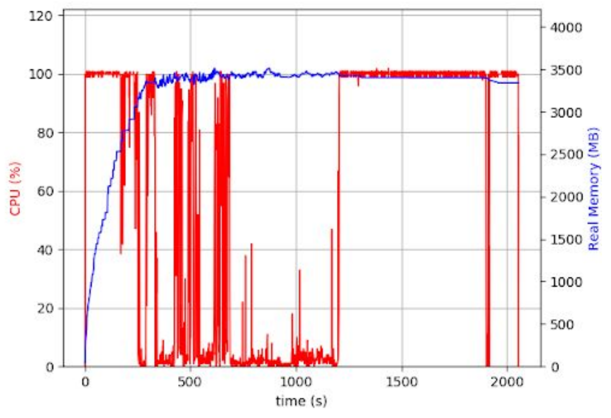


RTEC

Speed Change

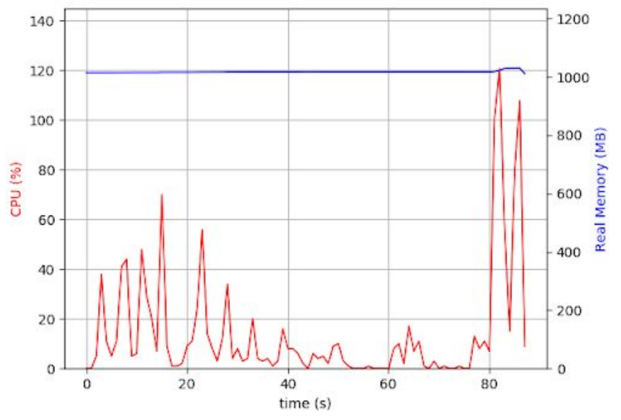


FlinkCEP

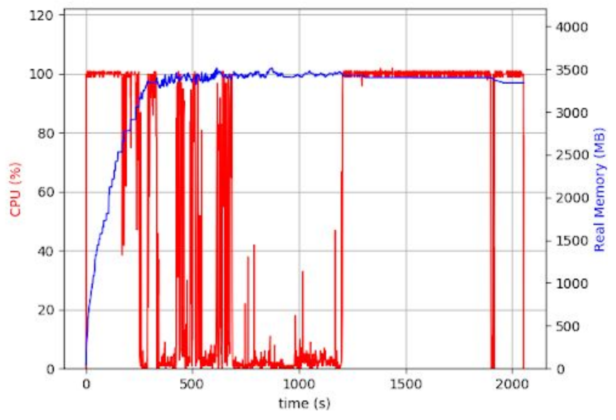


RTEC

Under Way

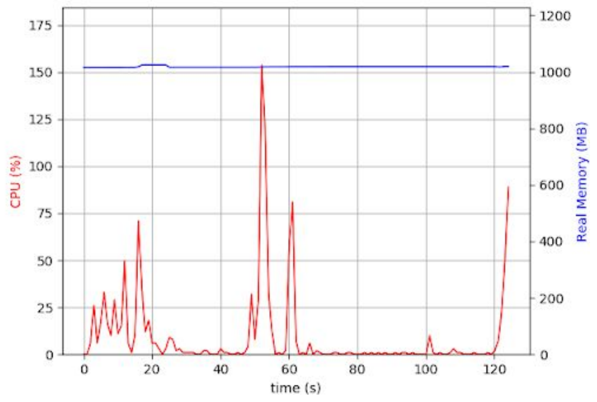


FlinkCEP

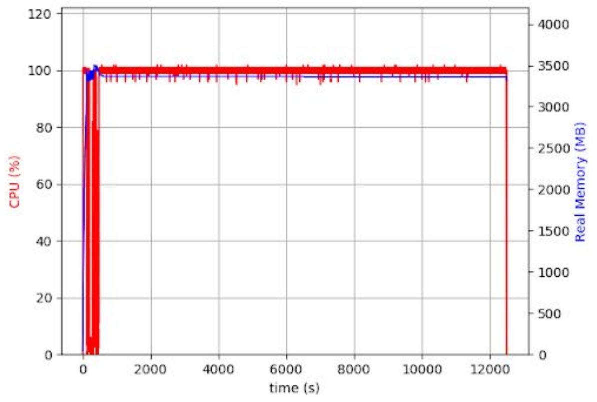


RTEC

Rendez Vous



FlinkCEP



RTEC

So, the datasets examined are not identical, but assumed to contain same amount of information, as the events in them refer to the same time interval. We also cannot conclude anything exact about the execution time because duration of the compared executions have to do with the base they rely on. Nevertheless, FlinkCEP's executions seem to complete significantly faster than RTEC's.

By the above comparison, it seems that RTEC generally reaches higher memory usage, whereas FlinkCEP uses almost constant memory throughout the execution. This might be caused by the fact that it uses Kafka server from where it loads the data. Another observation is that the memory of RTEC increases during execution, but its begin is at a much lower point.

As far as CPU is concerned, peaks are more common for FlinkCEP, whereas RTEC's CPU usage is constant for the same related periods of time. Most of the time, of course, the CPU usage is much higher for the RTEC. Because of the fact that the CPU for FlinkCEP climbs above 100% and RTEC's is not, we can conclude that RTEC utilizes only one CPU, but FlinkCEP uses more than one.

Instructions

- Zookeper and Kafka:
  - Open terminal in folder /home and type command "bin/zookeeper-server-start.sh config/zookeeper.properties" to start zookeper
  - Type "bin/kafka-server-start.sh config/server.properties" to start kafka
  - The topic is already created
- Flink:
  - Open intellij Idea

    - Run FarFromPortsExecutor.java file and Nari_Dynamic_KafkaStream.java to get the input from kafka to create the input stream for the Simple Patterns
    - Run SimplePatternExecutor.java file and then run KafkaStream.java to read the input stream created above
    - Complex patterns take input from csv file, so kafka is not necessary
- Results:
    - Results are saved in folder Results, which contains the output csv files