Applied Data Mining

**Part I: Automata-based CER with FlinkCEP**

Alevizopoulou Sofia 2022201704002

Avgeros Giannis 2022201704003

Tsiatsios George 2022201704024

Athens 2018

Contents

[Contents 1](#_Toc531221334)

[Table of figures 2](#_Toc531221335)

[Introduction 3](#_Toc531221336)

[System Architecture 3](#_Toc531221337)

[System Deployment 3](#_Toc531221338)

[Detecting Trajectory Events 5](#_Toc531221339)

[Online Noise Reduction 5](#_Toc531221340)

[1. Out of order events 5](#_Toc531221341)

[Online Tracking of Moving Vessels 5](#_Toc531221342)

[Grid partitioning 5](#_Toc531221343)

[Trajectory Events 6](#_Toc531221344)

[1. Gaps in communication 7](#_Toc531221345)

[2. Sharp changes on vessel’s heading 9](#_Toc531221346)

[3. Too slow or to too furious movement of a vessel 10](#_Toc531221347)

[4. Course of ground differentiates from the heading of a vessel 11](#_Toc531221348)

5. Long Term Stop....................................................................................................

6. Vessels with Wrong Status..................................................................................

[Complex event 12](#_Toc531221349)

[1. High Acceleration and fast approach 12](#_Toc531221350)

[2. CoTravel for 2 vessels 14](#_Toc531221351)

[3. Fishing Activity 17](#_Toc531221352)

[4. Vessel Rendezvous 19](#_Toc531221353)

[5. Package Picking 19](#_Toc531221353)

[Empirical Evaluation 21](#_Toc531221354)

[Visualization of the detected events 21](#_Toc531221355)

[WATERMARK PATTERN 21](#_Toc531221356)

[Running commands 22](#_Toc531221357)

[Running environment 23](#_Toc531221358)

[References 24](#_Toc531221359)

# Table of figures

[Table 1: Gap Pattern 7](#_Toc531221308)

[Table 2: Sharp heading changes pattern 9](#_Toc531221309)

[Table 3: Too slow or too furious vessel pattern 10](#_Toc531221310)

[Table 4:Heading and CoG pattern 11](#_Toc531221311)

[Table 5: Fast Approach pattern 13](#_Toc531221312)

[Table 6: Cotravel for 2 vessels pattern – trajectory events 15](#_Toc531221313)

[Table 7: Cotravel for 2 vessels pattern – complex events 16](#_Toc531221314)

[Table 8: Fishing pattern 17](#_Toc531221315)

[Table 9: Vessel Rendezvous pattern - trajectory events 19](#_Toc531221316)

[Table 10: Vessel Rendezvous pattern - complex events 20](#_Toc531221317)

# Introduction

A system for online monitoring marine activity over streaming positions from numerous vessels sailing at sea will be presented in this assignment. The system detects abnormal behavior in AIS messages emitted from vessels across time. AIS messages will be used to recognize suspicious events of one or multiple vessels.

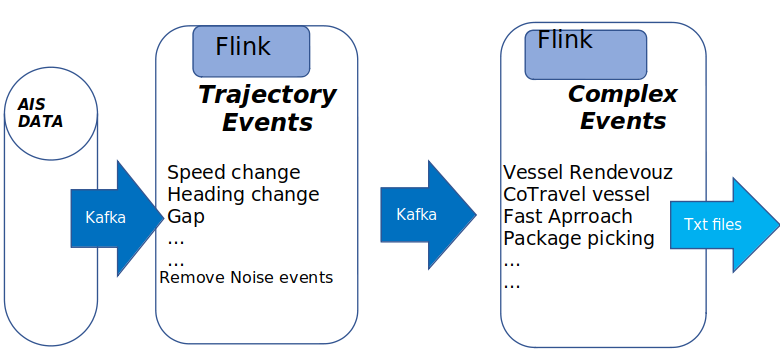
# System Architecture

The online monitoring system is a combination of the following systems: kafka and Apache Flink. Kafka is an open source streaming platform which is used for stream processing and apache Flink is a real time processing engine for stateful computations.

# System Deployment

The system implemented detects complex patterns in a stream of AIS messages. AIS messages are stored in a PSQL database. A python script has been written in order to fetch these messages from database and forward them to a kafka topic named “DEMOCP”.

The system consists of 2 different jobs. The 1st one is used to detect trajectory events for one vessel and the 2nd to detect complex event for more than one vessels based on the trajectory events that have been already detected at the previous step. The 1st project’s “flinkcep” input is what is written at topic DEMOCP (the ais messages fetched from database). The output of this project is forwarded to other kafka topics like DEMO\_CO, DEMOCP2 etc that given as input to the kafka producer of the 2nd project “cep\_flinkcep”. The results of both projects are written in txt and csv files. Each row of the txt/csv files contains information for the detected event. Csv files will be used to make some visualizations.



# Detecting Trajectory Events

Trajectory Detection is the main module of the project since the Complex Event Recognition module uses its outcome to compute the patterns that satisfy the conditions predefined. Trajectory Movement can be vessel Speed Change Events, gap in communication and generally events that related with the movement and the behavior of one vessel.

## Online Noise Reduction

AIS dataset, is a dataset that contains contains data which can be noisy, and as a result, difficult to be processed. Aiming to have better and more accurate results in the monitoring system, we should remove these noisy messages. In this implementation critical conditions have been added at the flinkcep patterns in order to identify noisy AIS messages, for examples emitted messages in a wrong time order. These conditions are checked firstly in the patterns in order to improve the performance of the system.

### Out of order events

As it is known, transmission delays may frequently occur between the original message and its arrival. Successive positional messages from a single vessel may often arrive intermingled at a distorted order. In our case the ais messages are stored in database so we assume that there is no delay between the original message and when this message arrives on the system. If this system was in a real world at which we would have delays, we should take care of messages that arrived out of order. We decided to exclude these kinds of messages by adding condition that checks the timestamp of each arrived messages. If the timestamp of the ais message for a specific vessel is smaller than the timestamp of the previous ais message, then this ais message will be ignored.

## Online Tracking of Moving Vessels

### Grid partitioning

In order to use the coordinates of a vessel and check how far is from an area or how far is from other vessels *geohash*, a grid portioning method has been used. Geohash is a geocoding system based on a hierarchical spatial data structure which subdivides space into buckets of grid (github.com/davidmoten/geo).

Each cell is labeled using a geohash which is of user-definable precision:

● High precision geohash have a long string length and represent cells that cover only a small area.

● Low precision geohash have a short string length and represent cells that each cover a large area.

GeoHash, can have a choice of precision between 1 and 12. As a consequence of the gradual precision degradation, nearby places will often present similar prefixes. The longer a shared prefix is, the closer the two places are. In the current implementation the below cell dimensions have been used.

|  |  |
| --- | --- |
| **Precision** | **Cell dimension** |
| 4 | 39,1 km x 19,5 km |
| 5 | 4,9 km x 4,9 km |
| 6 | 1,2 km x 600,4m |
| 7 | 152,9km x 152.4m |

### Trajectory Events

At this phase system deduce instantaneousevents by examining the trace of each vessel alone. This system consumes a stream of AIS tracking messages from vessels and continuously detects important patterns that characterize their movement.

A sample ais message that is consumed by the patterns written for trajectory detection is:

*"lat":2.541122,"lon":3.90484,"mmsi":14,"status":7,"speed":30,"turn":,"heading":36,"course":13,1, "t": 1443650402*

Trajectory detection creates a stream of important pattern of events that will be used from the complex event recognition system. Each such event is accompanied by the coordinates or some other characteristics of each corresponding vessel. At each class of trajectory events there is a message serializer class that serializes the accepted events and create another stream. These streams either will be used from the 2nd complex event recognition system or they will be written on txt/csv files if there is no need to proceed to the 2nd monitoring system.

Below are described the trajectory events we have implemented.

#### Gaps in communication

System detects vessels which has communication gaps on sending ais messages. We can define a gap as “*the absence of emitted AIS messages from a specific vessel for a specific period*”. In our approach this time is 600 secs. (120 secs?)[1]. So, if the system won’t receive an ais message from a vessel between 600 secs an its far away from a port (grid 4.9km x 4.9 km), then this vessel tagged as suspicious. This pattern characterized by the critical events “gap\_start” – “gap\_end”.

The flinkcep pattern is written bellow:. As time window we have used 3600 sec because

Table 1: Gap Pattern

|  |
| --- |
| **public** **static** Pattern<AisMessage, ?> patternGap(){  Pattern<AisMessage, ?> rendezvouzPattern = Pattern.<AisMessage>*begin*("gap\_start", AfterMatchSkipStrategy.*skipPastLastEvent*())  .followedBy("gap\_end")  .where(**new** IterativeCondition<AisMessage>()  @Override  **public** **boolean** filter(AisMessage event,Context<AisMessage> ctx) **throws** Exception { **for** (AisMessage ev : ctx.getEventsForPattern("gap\_start")) {  **if** ((event.getT() - ev.getT()) > *gapTime* && (event.getT() - ev.getT()) > 0  && *listOfPorts*.contains(GeoHash.*encodeHash*(event.getLat(), event.getLon(), *geoHashLen*)) == **false**)  {  **return** **true**;  } **else** **return** **false**;  }  **return** **false**;}  }).within(Time.*seconds*(3600));  **return** rendezvouzPattern;  } |

#### Sharp changes on vessel’s heading

First system detects vessels which have sharp changes in their headings. Sharp can be considered a change of value that exceeds 60 degrees. Vessels normally don’t have big changes in heading while they are moving [1] the combination of continuous changes of the heading with a gap in communication can be considered as an alert sign for illegal fishing. This is the complex event at which this trajectory event will be used.

The flinkcep pattern is written bellow. As time window we have 10 secs. It will be suspicious it is happened for 60 secs at least.

Table 2: Sharp heading changes pattern

|  |
| --- |
| **public** **static** Pattern<AisMessage, ?> patternFishing(){  Pattern<AisMessage, ?> fishingPattern = Pattern.<AisMessage>*begin*("start")  .followedBy("gap\_start")  .where(**new** IterativeCondition<AisMessage>() {  @Override  **public** **boolean** filter(AisMessage event,Context<AisMessage> ctx) **throws** Exception {  f**or** (AisMessage ev : ctx.getEventsForPattern("start")) {  **if**(Math.*abs*(ev.getHeading()event.getHeading())>*headingChange*)  {  **return** **true**;  }**else**{  **return** **false**;  }  }  **return** **false**;  }}).within(Time.*seconds*(3600));  } |

#### Too slow or to too furious movement of a vessel

At this case system detects vessels which have very small or very big speed for the type being. Each vessel type has its own min and max speeds defined so in any case that a vessel that overcomes these limits can be considered as an alert sign. In order to have the information about the ship type of each vessel we have created a csv file which contains all vessel types and all the mmsis for each type (these data fetched from nari\_static table on database). For the maximum and minimum speed of each vessel type, there is also a csv file that contains max and min speed per vessel type. This file created from information that is available on web.

The flinkcep pattern is written bellow. As time window we have 10 secs. It will be suspicious it is happened for 30 secs at least.

Table 3: Too slow or too furious vessel pattern

|  |
| --- |
| Pattern<AisMessage, ?> spaciousSpeedPattern = Pattern.<AisMessage>begin("speed\_spacicious\_start", AfterMatchSkipStrategy.skipPastLastEvent())  .followedBy("speed\_spacicious\_stop")  .where(new IterativeCondition<AisMessage>() {  public boolean filter(AisMessage event, Context<AisMessage> ctx) throws Exception {  for (AisMessage ev : ctx.getEventsForPattern("suspicious\_heading\_start")) {  String mmsi\_= String.valueOf(event.getMmsi());  String type="";  for (Object o : listOfVesselsType.keySet()) {  if (Arrays.asList(listOfVesselsType.get(o)).contains(mmsi\_))  {  type = o.toString();  }}    if (type.equals("")==false) {  String speed [] = listOfVesselsMaxMinSpeed.get(type);  if ( ((event.getSpeed()<Float.valueOf(speed[0])) || (event.getSpeed()>Float.valueOf(speed[1]))) && (event.getT() - ev.getT()) > 0) {  return true;  } else {return false;  }}else {  return false; }}return false;  }}).  within(Time.seconds(30)); |

#### Course of ground differentiates from the heading of a vessel

At this case system detects vessels with different heading and course. This can happened when there are bad weather condition and vessel cant follow its actual route. Heading describes the direction that a vessel is pointed at any time relative to the magnetic north pole or geographic north pole. As such, a stationary vessel (ex. a vessel which has been tied to a dock) will have a heading associated with the vessel's orientation. Course Over Ground (COG) describes the direction of motion with respect to the ground that a vessel has moved relative to the magnetic north pole or geographic north pole. An alert sign could be sent in this case.

The flinkcep pattern is written bellow. As time window we have 10 secs. It will be suspicious it is happened for 30 secs at least.

Table 4:Heading and CoG pattern

|  |
| --- |
| **Pattern<AisMessage, ?> spaciousHeading = Pattern.<AisMessage>begin("suspicious\_heading\_start", AfterMatchSkipStrategy.skipPastLastEvent())**  **.followedBy("suspicious\_heading\_stop")**  **.where(new IterativeCondition<AisMessage>() {**  **@Override**  **public boolean filter(AisMessage event, Context<AisMessage> ctx) throws Exception {**  **for (AisMessage ev : ctx.getEventsForPattern("suspicious\_heading\_start")) {**  **if ((Math.abs(ev.getHeading()-ev.getCourse())> 10) && (Math.abs(event.getHeading()-event.getCourse()))>10 && (event.getT() - ev.getT()) > 0) {**  **return true;**  **} else {return false; }**  **}return false;**  **}}).within(Time.seconds(30));**    **}** |

#### 

#### Long Term Stop

Long-Term stop is only fired if the vessel is noiticed to move slow after a pause. If M messages (after 60mins ) are with in a predefined area ( here I use geohash with length 6) , a long term stop is identified.

|  |
| --- |
| int longterm = 200;  Pattern<AisMessage, ?> LongTermStopPattern = Pattern.<AisMessage>*begin*("stop",AfterMatchSkipStrategy.*skipPastLastEvent*())  .subtype(AisMessage.class)  .where(new SimpleCondition<AisMessage>() {  @Override  public boolean filter(AisMessage event) throws Exception {  boolean near\_ports = false;  for(String str: Ports) {  String ship\_geohash = GeoHash.*encodeHash*(event.getLat(),event.getLon(),6);  if(str.equals(ship\_geohash))  near\_ports = true;  //System.out.printf("Ship near Port\n");   }  //System.out.printf("ships: %d %d %f\n", event.getMmsi(), event.getT(),event.getSpeed());   if((event.getSpeed() <1 && near\_ports == false)){  //low speed away from ports ---> pause  //System.out.printf("Pause by ship %s %d %f\n", event.getMmsi(), event.getT(),event.getSpeed());    return true;  }  return false;  }  })  //.oneOrMore()  //.timesOrMore(2)  .followedByAny("stop\_ends")  .where(new IterativeCondition<AisMessage>() {   @Override  public boolean filter(AisMessage event, Context<AisMessage> ctx) throws Exception {  boolean near\_ports = false;  for (AisMessage ev : ctx.getEventsForPattern("stop")) {  String geoHash1=GeoHash.*encodeHash*(event.getLat(),event.getLon(),6);    if ( ev.getMmsi() == event.getMmsi()) {   String geoHash2=GeoHash.*encodeHash*(ev.getLat(),ev.getLon(),6);    if((geoHash1.equals(geoHash2)) && (event.getSpeed()<2) && ev.getSpeed()<2 && (Math.*abs*(event.getT() - ev.getT()) > longterm)){  System.*out*.printf("LongStop %s %d %f\n?", event.getMmsi(), event.getT(),event.getSpeed());   return true;  }  else{  return false;  }  }  else {  return false;  }   }  return false;  }})  .within(Time.*seconds*(1800)); |

1. Vessels with false type

The dataset contains vessels with wrong status for example vessels with speed and status moored or at anchor. Those ships could be identified.

|  |
| --- |
| Pattern<AisMessage, AisMessage> alarmPattern = Pattern.<AisMessage>*begin*("first")  .where(new SimpleCondition<AisMessage>(){  @Override  public boolean filter(AisMessage event) {   if ((event.getStatus() != 1 || event.getStatus() != 5)) {  if(event.getSpeed() >5) {  return true;  }  else  return false;  //Ports.contains(GeoHash.encodeHash(event.getLat(), event.getLon(), 5)) == false   } else {  return false;  }  }   })  .within(Time.*seconds*(100)); |
|  |

### Complex event

Complex event Recognition module consumes the output of the Trajectory detection module so as to process the results and recognize in real time potentially complex maritime situations. Below will be described some patterns used for complex event recognition. For some of them, two monitoring systems needed. The 1st system for the trajectory events that are analyzed at the previous section and the 2nd one for combining these events to detect more complex patterns.

#### 1. High Acceleration and fast approach

First system detects vessels that have high acceleration at the open sea, this is a trajectory event. Acceleration is the temporal rate of change in velocity (speed) of a vessel. Suspicious accelerated considered each vessel that has accelerated more than 25%, more specifically speed has changed 20 KNOTS at less than 80 secs. [1]

a=Δυ/Δt0.25=20/ΔtΔt=80 sec

In order to check if the vessel is not near a port, the coordinates of Brittany’s ports have been used (the dataset includes messages from vessels near the area of France). A csv file is read and in the following step, we check whether the geohash of the vessel equals to the geohash of the ports. We have used precision 6(substring of the original geohash sized 6 is kept) which results in a grid 1,2 km x 620,4m. We want to verify that vessel is far away from a port for tagging its acceleration as suspicious. High acceleration near ports is normal. This pattern characterized by the critical events “acceleration\_start” – “acceleration\_end”.

The next step is to check if the corresponding vessel changes its speed while being at an open sea (not near ports), the new speed is above 20 knots in a small period of time , and there is at least one other nearby vessel toward. In other words, fast approach is a dangerous situation, which may arise when a vessel is rapidly moving towards some other vessel(s). We used again the geohash grid with precision 6 (1,2 km x 620,4m) to check if the geohash of the vessel equals to the geohash of other vessels. At this point will be checked all the vessels that have sent AIS messages until that moment. This precision is used as we would like to inform on time the other vessels that they are in danger.

At that point the system will check just the events that have been characterized by the critical tag “acceleration\_start” – “speed\_change” - “acceleration\_end” in order to detect the fast approach event.

The flinkcep pattern is written bellow. As time window we have used 10 sec because

Table 5: Fast Approach pattern

|  |
| --- |
| **public** **static** Pattern<AisMessage, ?> patternAcceleration(){  Pattern<AisMessage, ?> fastForwardPattern = Pattern.<AisMessage>*begin*("accelaration\_start")  .oneOrMore()  .followedBy("speed change")  .where(**new** IterativeCondition<AisMessage>() {  @Override  **public** **boolean** filter(AisMessage event, Context<AisMessage> ctx) **throws** Exception {  **if**(ctx!=**null**) {  **if** (ctx.getEventsForPattern("start") != **null**) {  **for** (AisMessage ev : ctx.getEventsForPattern("accelaration\_start")) {  **if** ((event.getSpeed() - ev.getSpeed()) >= *maxSpeed* && (event.getT() - ev.getT()) < *accelerationTime* && (event.getT() - ev.getT()) > 0 && *listOfPorts*.contains(GeoHash.*encodeHash*(event.getLat(), event.getLon(), *indexNearPorts*)) == **false** && ev.getMmsi() == event.getMmsi()) {  **return** **true**;  }}**return** **false**;  } **else** {**return** **false**;  }}**else**{  **return** **false**; }})  .followedBy("accelaration\_end")  .where(**new** IterativeCondition<AisMessage>() { @Override  **public** **boolean** filter(AisMessage event, Context<AisMessage> ctx) **throws** Exception {  ArrayList<AisMessage>aises=Lists.*newArrayList*(ctx.getEventsForPattern("speed change"));  AisMessage ship=**null**;  **for** (AisMessage ev : ctx.getEventsForPattern("accelaration\_start")) {  **if**(((GeoHash.*encodeHash*(ev.getLat(),ev.getLon(),*indexNearVessels*).  equals(GeoHash.*encodeHash*(ship.getLat(),ship.getLon(),*indexNearVessels*)))==**true** && ev.getT()-ship.getT()>0)){  **return** **true**}}**return** **false**;}})  .within(Time.*seconds*(*10*)); |

#### CoTravel for 2 vessels

The pattern, checks that the ais messages from 2 different vessels are between a time period of 10 secs. The pattern will detect events that happened closely in the time dimension. Next thing that will be checked is if the ships are on route and not paused. The speed of the vessel should be bigger than 1KNOT (trajectory event - 1 KNOT is the minimum speed of a vessel that isn’t in anchor). [1] At the end, the geohash of the 2 vessels will be checked in order to ensure that the two vessels were geographically closely. The precision of geohash is 7 (grid: 159,2m x 152.4m ).

This pattern characterized from events “vessel\_1” – “vessel\_2”.

At the end,all the detected patterns will be inserted into a kafka topic and will be given as input stream at the 2nd online monitoring system “cep\_flincep” to detect the desired events.

The format of the messages which kafka producer will receive is:

*MMSI\_1,Lon\_1,lat\_1,MMSI\_2,Lon\_2,lat\_2,time*

In order to make the second job more efficient, we decided to insert the id of the vessel that is the smallest between mmsi 1 and 2 as MMSI\_1 and to collect all the events per vessel. With that, instead of search through all the accepted events we will consider only the pairs that at least one of the MMSI does not change.

The flinkcep patterns are written bellow:. As time window we have used 10 secs for the 1st pattern and 3600 for the 2nd because

The 1st pattern is :

Table 6: Cotravel for 2 vessels pattern – trajectory events

|  |
| --- |
| **public** **static** Pattern<AisMessage, ?> patternCoTravel(){  Pattern<AisMessage, ?> coTravelPattern = Pattern.<AisMessage>*begin*("vessel\_1")  .subtype(AisMessage.**class**)  .followedBy("vessel\_2")  .subtype(AisMessage.**class**)  .where(**new** IterativeCondition<AisMessage>() {  @Override  **public** **boolean** filter(AisMessage event, Context<AisMessage> ctx) **throws** Exception {  **for** (AisMessage ev : ctx.getEventsForPattern("vessel\_1")) {  **if**(ev.getSpeed()>*minSpeed*  && event.getSpeed()>*minSpeed*  && ev.getMmsi()!=event.getMmsi()){  String geoHash1=  GeoHash.*encodeHash*(ev.getLat(),ev.getLon(),*geoHashLength*);  String geoHash2=  GeoHash.*encodeHash*(event.getLat(),event.getLon(),*geoHashLength*);  **if**(geoHash1.equals(geoHash2)==**true**){  **return** **true**;  }  **else**{  **return** **false**;  }  }  **else**{  **return** **false**;  } }  **return** **false**;}})  .within(Time.*seconds*(*10*)); |

The 2nd pattern is:

Table 7: Cotravel for 2 vessels pattern – complex events

|  |
| --- |
| Pattern<CoTravelInfo, ?> coTravelattern = Pattern.<CoTravelInfo>*begin*("msg\_1",AfterMatchSkipStrategy.*skipPastLastEvent*())  .subtype(CoTravelInfo.**class**)  .oneOrMore()  .followedBy("msg\_2")  .where(**new** IterativeCondition<CoTravelInfo>() {  @Override  **public** **boolean** filter(CoTravelInfo event, Context<CoTravelInfo> ctx) **throws** Exception{  **int** base = event.getTimestamp();  **int** currTime = event.getTimestamp();  List<CoTravelInfo> l = Lists.*newArrayList*(ctx.getEventsForPattern("msg\_1"));  **for** (CoTravelInfo ev : Lists.*reverse*(l)) {  **if** ((currTime - ev.getTimestamp()) < *coTravelTime*) {  **if** (event.getMmsi\_2() == ev.getMmsi\_2()) {  **if** ((base - ev.getTimestamp()) > *coTravellingTotalTime*) {  **return** **true**;  } **else** {  currTime = ev.getTimestamp();  }  }  } **else** {  **return** **false**;  }  }  **return** **false**;}})  .within(Time.*seconds*(3600)); |

#### Fishing Activity

This pattern combines two trajectory events in order to detect a complex event. More specifically checks the sequence of continuous changes of the heading, followed by a gap in communication and a bit turn in the end. This can be considered as an alert sign for illegal fishing. The captain checks the area and afterwards closes its GPS. At that point the system will check just the events that have been characterized by the tag “gap\_start” – “gap\_end” in order to detect some more changes in heading of the vessels. The trajectory events of heading change and gap in communication have been already analyzed.

The flinkcep pattern is written bellow. As time window we have used 3600 secs because

Table 8: Fishing pattern

|  |
| --- |
| **public** **static** Pattern<AisMessage, ?> patternFishing(){  Pattern<AisMessage, ?> fishingPattern = Pattern.<AisMessage>*begin*("start")  .subtype(AisMessage.**class**)  .followedBy("gap\_start")  .subtype(AisMessage.**class**)  .where(**new** IterativeCondition<AisMessage>() {  @Override  **public** **boolean** filter(AisMessage event, Context<AisMessage> ctx) **throws** Exception {  **for** (AisMessage ev : ctx.getEventsForPattern("start")) {  i**f**(Math.*abs*(ev.getHeading()-vent.getHeading())>*headingChange*)  {**return** **true**;  }**else**{  **return** **false**;  }}  **return** **false**;}})  .subtype(AisMessage.**class**)  .followedBy("gap\_end")  .subtype(AisMessage.**class**)  .where(**new** IterativeCondition<AisMessage>() {  @Override  **public** **boolean** filter(AisMessage event, Context<AisMessage> ctx) **throws** Exception {  **for** (AisMessage ev : ctx.getEventsForPattern("gap\_start")) {  **if**((event.getT()-ev.getT())>*gapTime*){  **return** **true**;  }**else**{  **return** **false**; }}  **return** **false**;}})  .followedBy("change in heading again")  .subtype(AisMessage.**class**)  .where(**new** IterativeCondition<AisMessage>() {  @Override  **public** **boolean** filter(AisMessage event, Context<AisMessage> ctx) **throws** Exception {  **for** (AisMessage ev : ctx.getEventsForPattern("gap\_end")) { **if**(Math.*abs*(ev.getHeading()-event.getHeading())>*headingChange*){  **return** **true**;  }**else**{  **return** **false**; }}  **return** **false**;}})  .within(Time.*seconds*(3600)); |

#### Vessel Rendezvous

The pattern, checks that the ais messages from 2 different vessels are between a time period of 60 secs. The pattern will detect events that happened closely in the time dimension. Next thing that will be checked is if the that have gap in their communication are in the same geohash grid. The precision of geohash is 5 (grid: 4.9km x 4.9 km ). The information about the geohash of each vessel is given by the previous pattern “gap-commmunication” whose outcome is used as input stream at this pattern. Two vessels seem to have suspicious renevouz when they have both gap in their communication at the same time as long as they are at the same geohash grid.

This pattern characterized from events “vessel\_1” – “vessel\_2”.

The flinkcep patterns are written bellow. As time window we have used 3600 secs for the 1st pattern and 60 for the 2nd because

The 1st pattern is:

Table 9: Vessel Rendezvous pattern - trajectory events

|  |
| --- |
| **public** **static** Pattern<AisMessage, ?> patternGap(){  Pattern<AisMessage, ?> rendezvouzPattern = Pattern.<AisMessage>*begin*("gap\_start", AfterMatchSkipStrategy.*skipPastLastEvent*())  .followedBy("gap\_end")  .where(**new** IterativeCondition<AisMessage>()  @Override  **public** **boolean** filter(AisMessage event,Context<AisMessage> ctx) **throws** Exception { **for** (AisMessage ev : ctx.getEventsForPattern("gap\_start")) {  **if** ((event.getT() - ev.getT()) > *gapTime* && (event.getT() - ev.getT()) > 0  && *listOfPorts*.contains(GeoHash.*encodeHash*(event.getLat(), event.getLon(), *geoHashLen*)) == **false**)  { **return** **true**;  } **else** {**return** **false**;  }**return** **false**;}  }).within(Time.*seconds*(3600)); } |

The 2nd pattern is:

Table 10: Vessel Rendezvous pattern - complex events

|  |
| --- |
| **public** **static** Pattern<GapMessage, ?> patternRendezvouz(){  Pattern<GapMessage, ?> rendevouzPattern = Pattern.<GapMessage>*begin*("Vessel\_1")  .subtype(GapMessage.**class**)  .followedBy("Vessel\_2")  .subtype(GapMessage.**class**)  .where(**new** IterativeCondition<GapMessage>() {  @Override  **public** **boolean** filter(GapMessage event, Context<GapMessage> ctx) **throws** Exception {  **for** (GapMessage ev : ctx.getEventsForPattern("Vessel\_1")) {  **if** ((ev.getGeoHash().equals(event.getGeoHash()) == **true**)  && ev.getMmsi()!=event.getMmsi()) {  **return** **true**;  } **else** { **return** **false**;  }}  **return** **false**;}})  .within(Time.*seconds*(*gapTime*)); |

#### Package Picking

*A possilble iteraction between two vessels is when one of them drops a package at some area and another vessel appears later in order to pick it up. By joining the previous long stop events using geohash area(length 6) we find ships that have long stops in the same area where the package picking is possible. As described before the ships should be away from ports.*

|  |
| --- |
| Pattern<AisMessage, ?> PackagePickingPattern = Pattern.<AisMessage>*begin*("drop\_packet\_vessel")  .subtype(AisMessage.class)  .followedByAny("pause\_vessel")  .subtype(AisMessage.class)  .where(new SimpleCondition<AisMessage>() {  @Override  public boolean filter(AisMessage event) throws Exception {  System.*out*.printf("ships: %d %d %f\n", event.getMmsi(), event.getT(),event.getSpeed());  boolean near\_ports = false;  for(String str: Ports) {  String ship\_geohash = GeoHash.*encodeHash*(event.getLat(),event.getLon(),6);  if(str.equals(ship\_geohash))  near\_ports = true; }  if((event.getSpeed() <1 && near\_ports == false) ) {  //low speed away from ports  return true;  }  return false;  }  })  .subtype(AisMessage.class)  .followedByAny("pick\_packet\_vessel")  .subtype(AisMessage.class)  .where(new IterativeCondition<AisMessage>() {  @Override  public boolean filter(AisMessage event, Context<AisMessage> ctx) throws Exception {  for (AisMessage ev : ctx.getEventsForPattern("pause\_vessel")) {  //Pause to pick   if ( ev.getMmsi() != event.getMmsi()) {    String geoHash1=GeoHash.*encodeHash*(ev.getLat(),ev.getLon(),6);  String geoHash2=GeoHash.*encodeHash*(event.getLat(),event.getLon(),6);   if((geoHash1.equals(geoHash2)) && (event.getSpeed()<1) ){   return true;  }  else{  return false;  }  }  else {  return false;  }   }  return false;  }})  .within(Time.*seconds*(100)); |
|  |

*6.Loitering*

*Loitering is the act of remaining in a particular area for a long period without purpose. Vessels with low speed , anchored or moored must be filtered out. If the messages from a single vessel are in the same area for 60 minutes (loitering time) this vessel is considered r to be loitering*

|  |
| --- |
| Pattern<AisMessage, ?> Loitering = Pattern.<AisMessage>*begin*("stop")  .subtype(AisMessage.class)  .where(new SimpleCondition<AisMessage>() {  @Override  public boolean filter(AisMessage event) throws Exception {  boolean near\_ports = false;  for(String str: Ports) {  String ship\_geohash = GeoHash.*encodeHash*(event.getLat(),event.getLon(),6);  if(str.equals(ship\_geohash))  near\_ports = true;  //System.out.printf("Ship near Port\n");   }  //System.out.printf("ships: %d %d %f\n", event.getMmsi(), event.getT(),event.getSpeed());   if((event.getSpeed() >2.87 && near\_ports == false)){  if(event.getSpeed() < 8){  //low speed away from ports    return true;}  else{  return false;  }  }  return false;  }  })  //.oneOrMore() //times could be also used  .followedByAny("stop\_ends")  .where(new IterativeCondition<AisMessage>() {   @Override  public boolean filter(AisMessage event, Context<AisMessage> ctx) throws Exception {   for (AisMessage ev : ctx.getEventsForPattern("stop")) {  String geoHash1=GeoHash.*encodeHash*(event.getLat(),event.getLon(),6);  // System.out.printf("events ship %s %d %s %f?\n",event.getMmsi(),event.getT(),geoHash1,event.getSpeed());  if ( ev.getMmsi() == event.getMmsi()) {    String geoHash2=GeoHash.*encodeHash*(ev.getLat(),ev.getLon(),6);  //System.out.printf("eventszssss ship %s %d %s %f?\n",ev.getMmsi(),ev.getT(),geoHash2,ev.getSpeed());  // added ev.speed() < 8  if((geoHash1.equals(geoHash2)) && ev.getSpeed()< 8 && (event.getSpeed()< 8 && (event.getT()-ev.getT()>60))){  if(event.getSpeed()>2.87 && ev.getSpeed() > 2.87){  System.*out*.printf("Loitering %s %d %f\n?", event.getMmsi(), event.getT(),event.getSpeed());   return true;  }  else{  return false;  }  }  else{  return false;  }  }  else {  return false;  }   }  return false;  }})  .within(Time.*seconds*(100)); |

# Empirical Evaluation

Changing the values on mobility tracking parameters eg: minimum speed, maximum acceleration etc we can see that the number of events that are detected changes dramatically. Doing some tests we conclude at the below results. We decided to use as parameters values these that were giving us more realistic results.

## Fast Approach

|  |  |
| --- | --- |
| Acceleration | Detected events |
| 15% |  |
| 25% |  |
| 40% |  |

## Gap in communications

|  |  |
| --- | --- |
| Secs of gap | Detected events |
|  |  |
|  |  |
|  |  |

## Time of Cotraveling

|  |  |
| --- | --- |
| Secs of cotraveling | Detected events |
|  |  |
|  |  |
|  |  |

## Rendezvous

|  |  |
| --- | --- |
| Secs of | Detected events |
|  |  |
|  |  |
|  |  |

# Visualization of the detected events

Except from the txt files that are created for each complex event (contains information about the vessels and that take part in a complex event) csv files are also created with the same information. These files are used in QGIS. The below pictures come from QGIS .

# WATERMARK PATTERN

# Running commands

* ***$ flink\_1.6.2:bin/start-cluster.sh***
* start the cluster of flink
* Check that server is running on http://localhost:8081/#/overview
* ***$ kafka2.2:bin/zookeeper-server-start.sh config/zookeeper.properties***
* ***$ kafka2.2:bin/kafka-server-start.sh config/server.properties***
* Start the kafka and zookeeper servers
* ***$ kafka2.2:bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 partitions 1 --topic DEMOCP***
* ***$kafka2.2:bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic DEMOCP2***
* ***$kafka2.2:bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic DEMOCP\_CO***
* ***$kafka2.2:bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic DEMOCP\_ACC***
  + Create the topics on kafka
* ***sudo /home/cer/Downloads/flink-1.6.2/bin/flink run /home/cer/Desktop/cer\_2/flinkcep/cep\_flinkcep/target/flinkicu\_cep-1.0-jar-with-dependencies.jar --topic\_gap DEMOCP2 --bootstrap.servers localhost:9092 --zookeeper.connect localhost:2181 --topic\_co DEMOCP\_CO***
* ***sudo /home/cer/Downloads/flink-1.6.2/bin/flink run /home/cer/Desktop/cer\_2/flinkcep/flinkcep/target/flinkicu-1.0-jar-with-dependencies.jar --topic DEMOCP --bootstrap.servers localhost:9092 --zookeeper.connect localhost:2181 --out /home/cer/Desktop/out.txt --topic\_output\_acc DEMO\_ACC --topic\_output\_gap DEMOCP2 --topic\_output\_co DEMOCP\_CO***
* ***$ Kafka2.2:bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 topic DEMOCP***
* You can check the events that detected at each job at this url. http://localhost:8081/#/overview
* You can see what is send at each topic running the consumer of each topic. For example for topic DEMOCP, running the consumer you will see the ais messages
* ***./ais.py DEMOCP --topic\_output DEMOCP2***
* Run the python script in order to full fill the topic that contains all the ais messages and 1st module starts receiving ais messages
* Producer is inside the project at the path ‘flinkcep/producer’

Running the script suspicious events will start to be detected. All of these will be written at the path *home/cer/Desktop/temp/* as .txt /.csv files. There will be a .txt /.csv file for each trajectory and suspicious event.

# Running environment

|  |  |
| --- | --- |
| OS | PRETTY\_NAME="Ubuntu 18.04.1 LTS"  VERSION\_ID="18.04" |
| Postgres | postgres=# SELECT version();  PostgreSQL 10.5 (Ubuntu 10.5-0ubuntu0.18.04) on x86\_64-pc-linux-gnu, compiled by gcc (Ubuntu 7.3.0-16ubuntu3) 7.3.0, 64-bit |
| QGIS | 2.18.0 |
| Python | Python 2.7.15 |
| Kafka | kafka\_2.11-1.0.0 |
| Flink | flink-1.6.2 |
| Scala | Scala code runner version 2.11.12 -- Copyright 2002-2017, LAMP/E |

## References

[1] Online Event Recognition from Moving Vessel Trajectories Kostas Patroumpas · Elias Alevizos · Alexander Artikis · Marios Vodas · Nikos Pelekis · Yannis Theodoridis.

[2] COMPARISON OF INS HEADING AND GPS COG, R. Michael Reynolds November 6, 20

http://www.rmrco.com/docs/m1227\_Compare-cog-hdg.pdf