Università degli Studi di Salerno

Relazione Situation Awareness

Data stream processing and Complex Event Processing (CEP) for situation identification

Giulia Apicella: 0622701187
Gianluca D'Amore: 0622701169
Vincenzo Noviello: 0622701270
Emilio Sorrentino: 0622701199
Luca Giuseppe Tramuto: 0622701170

Indice

Capitolo 1

Introduzione

Per *Situation Awareness* si intende la consapevolezza della situazione circostante, la comprensione di cosa sta succedendo in un preciso momento in base alla percezione e agli stimoli del mondo esterno, l'elaborazione e la proiezione in un prossimo futuro per effettuare delle azioni. Una definizione formale è "la percezione degli elementi in un ambiente all'interno di un volume di spazio e tempo, la comprensione del loro significato e la proiezione del loro stato nell'immediato futuro" (Endsley, 1988). Essa è dunque formata da 3 livelli: percezione, comprensione e proiezione.

1.1 Dominio di Interesse

Il dominio sul quale si focalizza il progetto in esame è quello marittimo, più precisamente si parla di Maritime Situational Awareness.

Il concetto di Maritime Situational Awareness può essere declinato in una capacità "abilitante" tesa al raggiungimento della necessaria superiorità informativa nell'ambiente marittimo al fine della più completa comprensione delle dinamiche della situazione in atto nello scenario operativo e allo scopo di incrementare l'efficacia nella pianificazione e condotta delle operazioni marittime. Il dominio marittimo è definito come tutte le aree adiacenti o confinanti con un mare, un oceano o altre vie navigabili, comprese tutte le attività, infrastrutture, persone, merci, navi e altri mezzi di trasporto.

Con l'aumentare del traffico marittimo è diventato sempre più difficile, per i sistemi di monitoraggio marittimo, sorvegliare tutte le attività che avvengono in mare. Nasce la necessità di utilizzare un sistema di data processing che effettui l'analisi dei dati in real-time in modo tale da garantire la navigazione sicura, rilevando, in tempo reale, le attività navali pericolose, sospette e illegali. Questi sistemi di monitoraggio tipicamente fanno uso di un sistema di identificazione automatica (AIS), ovvero uno standard internazionale di identificazione delle navi che consente ad esse di trasmettere e ricevere informazioni sulle navi nelle vicinanze. L'AIS integra un ricetrasmettitore VHF con un dispositivo di posizionamento, come il GPS, e altri sensori elettronici di navigazione, come un indicatore della velocità di virata oppure la girobussola, producendo così dati molto importanti e preziosi che sono raccolti dalle stazioni base AIS lungo le coste, o anche dai satelliti, quando sono fuori dalla portata delle reti terrestri.

L' obiettivo del progetto è la realizzazione di un *data stream processing* che consenta di ottenere dati dai messaggi AIS: tramite essi rilevare eventi semplici e rilevare situazioni come eventi complessi (composti da una sequenza di eventi semplici), al fine di acquisire piena consapevolezza dello scenario operativo. In particolare, si vuole identificare l'evento della pesca a strascico (trawling) utilizzando regole di velocità e direzionalità, al fine di consentire, al sistema di monitoraggio marittimo una migliore regolamentazione e monitoraggio delle attività di pesca. Ciò può essere a supporto di studi che si occupano dell'impatto del trawling sulla specie, sull'habitat e sull'ecosistema.

1.2 Descrizione del problema

Si vogliono rilevare in tempo reale eventi semplici e eventi complessi svolti dalle navi. In particolare, nel progetto in questione si vuole rilevare l'evento di *trawling* (pesca a strascico). Nell'immagine in basso è possibile notare che tale evento risulta essere un evento complesso, ovvero una combinazione di più eventi semplici quali *WithinArea*, *TrawlSpeed* e *trawlingMovement*.

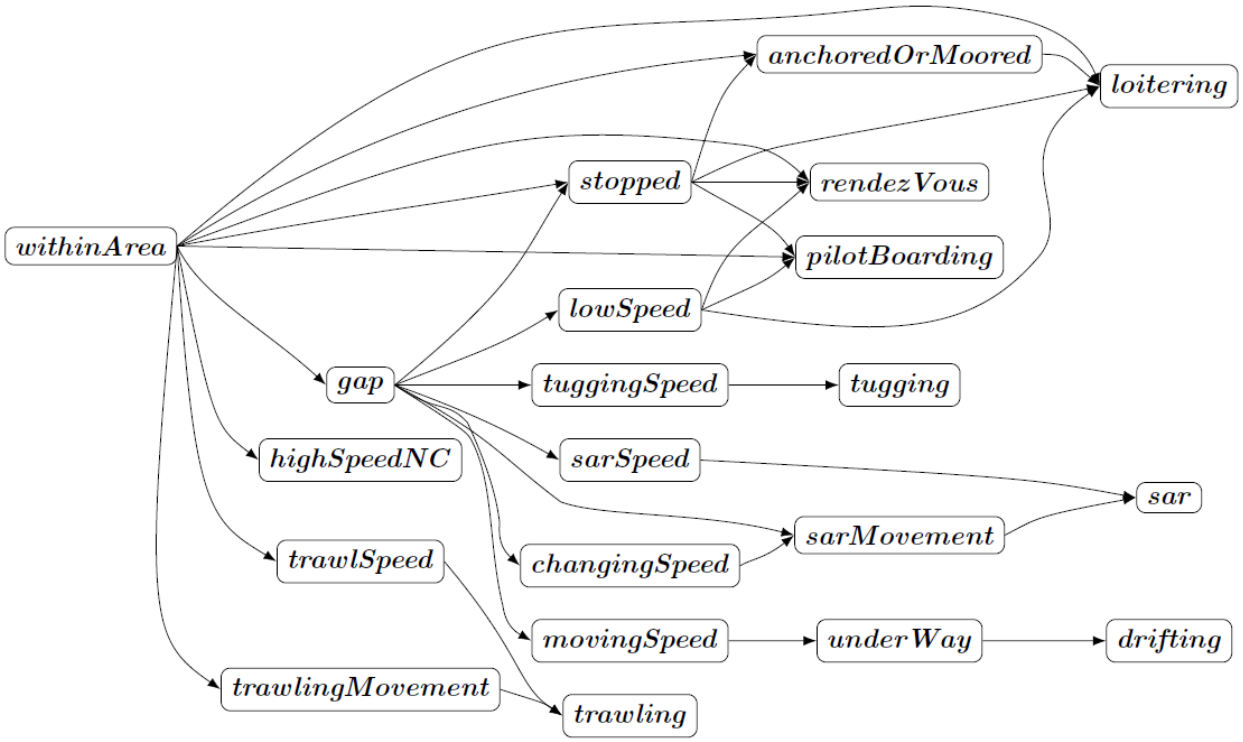

Figura 1.1: Composite Events

La pesca a strascico è un metodo di pesca comune che coinvolge un peschereccio da traino, che tira una rete da pesca. Il peschereccio, durante la fase di pesca a strascico, ha una velocità costante (trawlSpeed), e un'ampia distribuzione dell'angolo di prua.

TrawlingMovement è un evento composto dai due eventi semplici *withinArea* e *headingChange* [10]. Per definizione, esso è vero fin quando l'imbarcazione da pesca in questione si trova in una zona di pesca (*withinArea*) ed esegue una sequenza di cambi di rotta (*headingChange*), ognuno dei quali avviene al più 10 minuti dopo il precedente. Quindi un'imbarcazione si dice che sta facendo

trawling se essa è un'imbarcazione da pesca, sta effettuando un *trawlingMovement* e naviga ad una velocità *trawlSpeed* per un periodo pari ad un'ora [10].

1.3 Impatto sulla SA

Un ruolo importante nella SA è svolto dalla percezione del tempo in quanto è fondamentale comprendere quanto tempo disponibile rimane fino al verificarsi di un certo evento o prima di intraprendere una determinata azione. Inoltre, la conoscenza del ritmo con cui l'informazione cambia consente di avere una proiezione nel prossimo futuro più precisa. I campi applicativi in cui la SA è basilare sono dinamici: è importante assicurare il costante cambiamento della SA della persona, ogni volta che c'è un cambiamento della situazione, senza automatizzare il processo ed evitando di rendere la SA dell'uomo obsoleta e inesatta.

Il sistema realizzato potrebbe essere a supporto della SA in quanto, potenzialmente, potrebbe fungere da filtro per una mole elevata di informazioni, processate in real-time, che potrebbero comportare un carico cognitivo eccessivo per l'operatore, con conseguenze negativa sulla sua SA e successiva decisione.

Risulta fondamentale, infatti, limitare l'effetto che bias cognitivi e demoni della SA hanno sul processo decisionale degli operatori. A tal proposito, un elevato numero di informazioni si traduce, lato operatore, in uno sforzo cognitivo al fine di percepire e comprendere quanto meglio possibile la situazione corrente. Molto spesso, tale carico, potrebbe essere difficilmente sostenibile, soprattutto se prolungato nel tempo. Si parla, quindi, del demone di Data Overload.

Una possibile soluzione potrebbe essere automatizzare quanto più possibile il processo, limitando l'attenzione dell'operatore agli aspetti maggiormente salienti. Tuttavia, un elevato grado di automatizzazione non è sempre la scelta ideale: al crescere della complessità del sistema, l'operatore potrebbe essere in difficoltà nel capire cosa sta succedendo al sistema stesso, se esso procede e si evolve in completa autonomia. In questi casi si parla di Out-of-the loop demon.

Il sistema prodotto ha l'obiettivo di filtrare il numero di elementi su cui l'operatore deve prestare attenzione, limitandosi, in particolare, a quelle imbarcazioni che presentano i tratti distintivi di eventi di trawling. Sarà, quindi, compito dell'esperto, una volta percepito e compreso le informazioni, prendere le corrette decisioni, rimandendo, di fatto, parte attiva del processo.

Capitolo 2

Tecnologie

La fase di sviluppo e testing software sono state eseguite su ambiente Linux Ubuntu 18.04.05 LTS, con Processore Intel Core i7-6700HQ e 8GB RAM.

La scelta di un sistema Linux-Based è stata effettuata sulla base di alcuni bug noti che la versione Windows di Kafka presenta in merito alla gestione di cartelle.

2.1 Kafka 2.8

La tecnologia utilizzata per assicurare il processamento dei dati in real time è **Apache Kafka**[5]. Essa è una piattaforma di streaming di eventi distribuita open source. Kafka è un sistema distribuito composto da **server e client** che comunicano tramite un protocollo di rete TCP ad alte prestazioni. Kafka viene eseguito come un cluster di uno o più server che possono estendersi su più datacenter. Alcuni di questi server formano il livello di applicazione chiamato **broker**.

Altri server, invece, eseguono **Kafka Connect** per importare ed esportare continuamente dati come flussi di eventi per integrare Kafka con i sistemi esistenti, ad esempio database relazionali o altri cluster Kafka. Inoltre, un cluster Kafka è altamente scalabile e tollerante ai guasti quindi se uno dei server si guasta, gli altri server si occuperanno del loro lavoro per garantire operazioni continue senza alcuna perdita di dati. I client Kafka consentono di scrivere applicazioni distribuite e microservizi che leggono, scrivono ed elaborano flussi di eventi in parallelo, su larga scala e con tolleranza di errore anche in caso di problemi di rete o guasti delle macchine.

Kafka si basa sul concetto di **producer-consumer**, i producer sono quelle applicazioni client che pubblicano (scrivono) gli eventi in Kafka e i consumer sono quelli che sottoscrivono (leggono ed elaborano) questi eventi. Producer e consumer sono completamente disaccoppiati e indipendenti l'uno dall'altro. Gli eventi sono organizzati e archiviati in modo duraturo in **topic**. I topic in Kafka sono sempre multi-producer e multi-consumer: un topic può avere zero, uno o molti producer che vi scrivono eventi, così come zero, uno o molti consumer che si iscrivono a questi eventi. Inoltre, tramite un'impostazione di configurazione, può essere definito anche per quanto tempo Kafka deve conservare gli eventi, dopodiché i vecchi eventi verranno eliminati.

I topic sono **partizionati**, questo significa che un topic è distribuito su un numero di "bucket" situati su diversi broker Kafka. Questo è molto importante per la scalabilità perché consente alle applicazioni client di leggere e scrivere i dati da più broker contemporaneamente. Quando un

nuovo evento viene pubblicato su un topic, viene effettivamente aggiunto a una delle partizioni del topic, gli eventi con la stessa chiave evento, vengono scritti sulla stessa partizione e Kafka garantisce che qualsiasi consumer iscritto al topic, otterrà sempre gli eventi di quella partizione esattamente nello stesso ordine in cui sono stati scritti.

2.1.1 Applicazione di Kafka nel progetto

Per lo scopo del progetto in questione sono state utilizzate le librerie *Kafka Streams* e *Kafka Clients*. Kafka Streams è una libreria client per la creazione di applicazioni e microservizi, in cui i dati di input e output sono archiviati in un cluster Apache Kafka, fornisce funzioni di livello superiore per elaborare flussi di eventi, incluse trasformazioni, join, aggregazioni, finestre e altro ancora. l'input viene letto da uno o più topic per generare un output su uno o più topic, trasformando in maniere efficiente i flussi di input in flussi di output.

2.2 Librerie

2.2.1 Pandas

Pandas è una libreria open source software scritta per il linguaggio di programmazione Python per la manipolazione e l'analisi dei dati. In particolare, offre strutture dati e operazioni per manipolare tabelle numeriche e serie temporali [8].

In particolare, Pandas è stato utilizzato per poter manipolare i dataset di modo da poter agevolare l'analisi dei dati per la topologia kafka. Di fatti, tale libreria permette una manipolazione veloce ed efficace dei dati sotto forma di **DataFrame**: questo ci ha permesso di effettuare slicing del dataset e query di analisi.

2.2.2 Matplotlib

Matplotlib è una libreria *open source* per la creazione di grafici per il linguaggio di programmazione *Python* e la libreria matematica *NumPy* [7].

Questo ci ha permesso di realizzare i grafici delle traiettorie rappresentate sull'area marittima di Brest.

2.2.3 GeoTools

GeoTools è una libreria Java, distribuita con licenza LGPL, per leggere, manipolare, visualizzare ed analizzare dati geografici.

In quanto libreria, essa è concepita per rispondere a criteri di riusabilità, generalità e flessibilità. Per questa ragione prevede implementazioni alternative di uno stesso gruppo di interfacce con obiettivi diversi (diversi trade-off fra dimensione del codice, velocità, uso di memoria, ecc.).

È in grado di lavorare con gli standard OpenGIS per ogni componente di cui sia disponibile una specifica. La sua struttura modulare consente di selezionare i jar che di volta in volta si ha bisogno per la creazione di una nuova applicazione o libreria [3].

Capitolo 3

Pre elaborazione dei dati

Per evitare situazioni di congestione della macchina e di rallentamenti eccessivi, sono state introdotte tecniche di pre-elaborazione dei dati e di prefiltraggio con il fine di rendere meno oneroso il carico computazione della macchina.

3.1 Dataset Utilizzati

Il dataset principale utilizzato è stato preso dal progetto datAcron. Tale progetto, finanziato dall'UE, ha l'obiettivo di gestire e integrare tutti i dati-at-rest (dati prelevati in situazioni stazionarie) e dati-in-motion (dati prelevati in streaming), in modo da migliorare significativamente le capacità dei sistemi di monitoraggio marittimo.

In particolare, i dataset presi in considerazioni sono stati:

- $ais\_brest\_synopses$: contiene le sinossi di traiettorie a partire dai dati forniti dal NARI (Institut de Recherche de l'École Navale), ovvero i messaggi cinematici AIS delle navi che navigano nell'Oceano Atlantico intorno al porto di Brest, Bretagna e Francia, per un periodo che va dal 1 ottobre 2015 al 31 marzo 2016. Contiene informazioni circa l'identificativo della nave (MMSI) , il timestamp (in millisecondi, è il tempo di ricezione dell'AIS), longitudine, latitudine, *annotation* (caratterizzazione della locazione), velocità della nave. Tale dataset è stato utilizzato per riconoscere gli eventi semplici e costruire l'evento complesso di trawling. Le annotation sono caratterizzata su una stringa di 8 bit:
  - 00000001: stop_start
  - 00000010: stop_end
  - 00000100: slow_motion_start
  - 00001000: slow_motion_end
  - 00010000: gap_end
  - 00100000: change_in_heading
  - 01000000: change_in_speed_start
  - 10000000: change_in_speed_end

Nel caso in esame, vengono prese in considerazione le navi che hanno l'annotation di *change_in_heading*.

- *anfr*: contiene una lista delle navi registrate dall'ANFR (French Frequencies Agency). Tale dataset contiene, oltre alle informazioni riguardanti l'aria marittima, nome, lunghezza e identificativo della nave, anche il tipo della nave.

- *v_recode_fish_area_clean*: é uno *shapefile* contenente le zone di pesca stimate dalla commissione europea, con latitudine compresa tra -10 e 0 e longitudine tra 45 e 51. E' stato utile per ricavare le zone di pesca nelle zone di interesse di studio (area di Brest), per avere una maggiore precisione nel filtrare l'evento *withinarea*.

3.2 Unione dei dataset

Si è scelto di utilizzare il dataset *ais_brest_synopses* in quanto forniva un accesso diretto, senza elaborazioni, ai record contenenti eventi di heading-change, tramite il campo annotation. L'informazione mancante sulla tipologia di nave è stata recuperata dal dataset *anfr*, effettuando un merge, tramite mmsi.

Da tale operazione è stato generato il dataset *ais_data_types.csv*.

Capitolo 4

Design e Sviluppo

4.1 Topic

Il primo step per la risoluzione del problema è stato la fase di setup della piattaforma Kafka. Inizialmente sono stati definiti e configurati tutti i topic con i seguenti parametri:

- replication-factor=1
- partitions = 1
- message.timestamp.type=CreateTime

Con l'ultimo parametro si assicura che il timestamp nel messaggio coincida con l'ora di creazione del messaggio, ovvero con il timestamp prelevato dal Producer.

I topic generati sono:

- *ais-topic*: viene simulato l'invio dei messaggi AIS
- *change-heading*: vengono pubblicati gli id delle navi e i timestamp di inizio e fine della finestra riguardante l'evento di heading change.
- *within-area*: vengono pubblicati gli id delle navi e i timestamp di inizio e fine della finestra riguardante l'evento di within area, all'interno delle aree marcate come zone di pesca.
- *trawling-movement*: vengono pubblicati gli id delle navi e i timestamp di inizio e fine della finestra riguardante l'evento di trawling movement.
- *trawling-speed*: vengono pubblicati gli id delle navi e i timestamp di inizio e fine della finestra riguardante l'evento di trawl speed.
- *output*: la combinazione degli output dei topic precedenti fornisce l'evento di trawling, su tale topic vengono, quindi, pubblicati gli id delle navi e i timestamp di inizio e fine dell'evento di trawling.

4.2 Classe AISMessage

Kafka permette la creazione di tipi custom per i messaggi che viaggiono sui topic [1]. Sfruttando tale potenzialità è stata definita la classe AISMessage, contenente tutte le informazioni utili, ottenute tramite lettura del dataset, e le interfacce di Serializzazione[11] e Deserializzazione[2] per l'invio dei messaggi, in formato JSON, sul topic.

Tale scelta progettuale concorre alla modularità del sistema: il formato json, infatti, rende semplificata la modifica dei campi e il loro monitoraggio.

4.3 Timestamp Extractor

Per effettuare analisi in tempo reale su dati storici è sorta la necessità di impostare un timestamp dei messaggi differente da quello di sistema, come implementato di default dal broker Kafka, e che rispettasse in maniera fedele l'ordine di arrivo dei messaggi nella realtà. A tale fine, è stato implementato un **TimestampExtractor** [13]: dato un record di input, contenente il timestamp reale del messaggio, estraendo tale campo è possibile settarlo come timestamp di pubblicazione.

```
7   public class CustomExtractor implements TimestampExtractor {
8       @Override
9       public long extract(ConsumerRecord<Object, Object> consumerRecord, long l) {
10
11          AISMessage s = (AISMessage) consumerRecord.value();
12
13          return Long.parseLong(s.getTimestamp());
14      }
15  }
```

Figura 4.1: Timestamp Extractor

4.4 Topologia

Fondamentale per l'analisi di dati in tempo reale è la costruzione di una topologia: la topologia è un grafico aciclico diretto di nodi di elaborazione del flusso che rappresenta la logica di elaborazione del flusso di un'applicazione Kafka Streams.

4.4.1 Prototipo

La prima topologia Kafka implementata ha considerato:

- *AISProducer*: ad intervalli regolari si occupa di leggere dal file *ais_brest_synopses.csv* tutti i record con stesso timestamp, settare il timestamp dei messaggi attraverso l'interfaccia *TimestampExtractor* e inviare i record estratti sul topic *ais-topic*. Si è cercato di simulare la generazione reale dei dati in un sistema AIS.

- *processor_typeShip*: nodo di filtraggio per tipologia nave che, consumando le informazioni prodotte dal Producer, inoltra ai KStream successivi solamente i dati di imbarcazioni da pesca.

- *processor_whithinArea*: KStream di processamento dell'evento within-area; filtrando esclusivamente le imbarcazioni presenti nelle zone di pesca, i messaggi, organizzati per MMSI, vengono finestrati mediante *SessionWindow* e viene fornito in output il risultato della finestratura.

- *processor_heading*: KStream di processamento dell'evento *heading-change*; effettua un'operazione di filtraggio della imbarcazioni che effettuano *heading-change*; si procede alla fase di finestratura, rispettando il vincolo di avere un heading-change al massimo ogni 10 minuti.

- *Trawling-join*: KStream di combinazione tra gli output di heading-change e within-area per l'ottenimento dell'evento composto trawling-movement. In particolare, viene effettuato un join per chiave (MMSI) e, attraverso la finestratatura, si impone il vincolo di durata di almeno 10 minuti.

- *processor_speed*: KStream per il filtraggio e la rilevazione dell'evento di TrawlingSpeed, imponendo un intervallo valido di velocità ([1,9] nodi).

- *trawling_result*: KStream di combinazione, mediante join, tra gli eventi di *trawling-movement* e *trawling-speed*.

In una prima fase di sviluppo nei nodi processori sono state utilizzate SessionWindow [4], organizzate per MMSI, messe a disposizione dalla libreria Kafka-Stream. L'idea alla base è quella di estendere la finestra di un determinato evento fintanto che esso si presenta con una certa regolarità temporale e chiudere la finestra se, per un fissato arco temporale definito gap, l'evento non riceve aggiornamenti.

4.4.1.1 Problemi riscontrati

Il limite di tale approccio è duale ed è conseguenza della topologia utilizzata e del funzionamento delle *SessionWindow*. Considerando e processando esclusivamente gli eventi atomici asseriti (imbarcazione in una data zona, imbarcazione che effettua entro un certo limite di tempo heading change, etc...) in uscita dai nodi processori di filtraggio, non vengono considerati eventuali interruzioni di tali comportamenti, portando, di fatto, a una prosecuzione della finestra stessa, piuttosto che a una sua interruzione e conseguente generazione di una nuova. In aggiunta, per definizione di SessionWIndow, non vi è alcun meccanismo che fornisce un modo per forzare la terminazione di una finestra (in questo caso necessaria, se arriva un messaggio che nega un evento) e la generazione di una successiva.

4.5 Topologia aggiornata

Per la risoluzione delle problematiche sopra elencate si procede alla progettazione di una nuova topologia e all'introduzione di meccanismi, offerti dalla libreria Kafka-Stream, differenti dalla semplice finestratura.

La nuova architettura prevede la conservazione dei KStream Producer e del filtraggio per tipologia nave, precedentemente definiti: il primo è necessario alla simulazione dell'invio dei messaggi; il secondo in quanto, per definizione, condizione necessaria affinché una nave faccia trawling è che sia da pesca.

Per i KStream successivi sono state implementate le interfacce *Transformer*[14] e *TransformerSupplier*[15] per l'applicazione di trasformazioni *ad hoc* sugli input. Attraverso la gestione di un meccanismo *Stateful*[12], realizzato attraverso una struttura *KeyValueStore*[6], è stato possibile realizzare un meccanismo paragonabile alla finestratura in cui, al riconoscimento di un preciso evento all'interno di un messaggio, sono state implementate logiche di gestione della stessa, in modo coerente con quanto dichiarato nel documento *Composite event for maritime monitoring*[10].

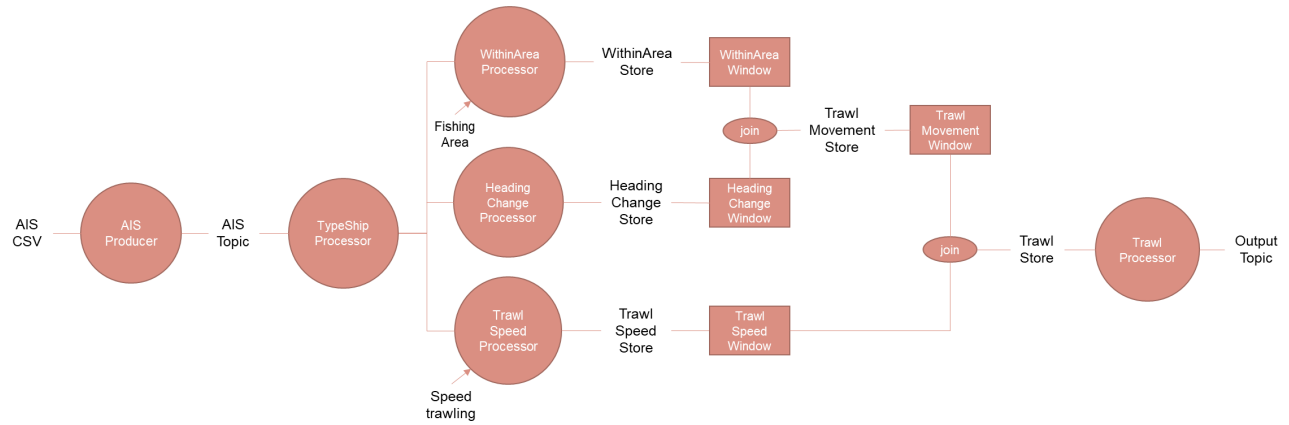

Figura 4.2: Topology

4.5.1 Heading Change

Il vincolo fondamentale per tale evento è l'arrivo di un messaggio, con distanza temporale massima rispetto al messaggio precedente di 10 minuti. Quindi, funzionalmente, il Trasformer implementato segue la logica:

- All'arrivo di un messaggio, se l'MMSI della nave non è presente nel KeyValueStore, esso inserisce la tupla (MMSI, Timestamp del messaggio), segnando, di fatto, l'inizio della finestra.
- Se esiste già una entry per MMSI:
  - se da tale messaggio viene estratta l'informazione di heading-change in corso (nel campo *annotation* è presente un 1 in terza posizione):

```
if (this.kvStore.get(mmsi)==null){

    if (annotation.charAt(2) == '1'){
        new_value = String.format(value.getTimestamp() +','+ value.getTimestamp());
        kvStore.put(id, new_value);
        return new KeyValue<String,String>(mmsi,new_value);
    }

} else {
    final String[] kvStoreData = kvStore.get(mmsi).split(",");
    Long startTimestamp = Long.parseLong(kvStoreData[0]);
    Long precTimestamp = Long.parseLong(kvStoreData[1]);
    Long currTimestamp = Long.parseLong(value.getTimestamp());
    Long difference = (currTimestamp - startTimestamp)/(60 * 1000);
    if (annotation.charAt(2) == '1'){
        if (currTimestamp -precTimestamp <= Duration.ofMinutes(10).toMillis()){

            String new_value_insert =String.format(startTimestamp.toString() + ','+ currTimestamp.toString()+','+Long.toString(difference));
            kvStore.put(mmsi,new_value_insert);
            return new KeyValue<String,String>(mmsi,new_value_insert);
        }
        else{
            new_value =String.format(currTimestamp.toString() + ','+ currTimestamp.toString());
            kvStore.put(mmsi,new_value);

            return new KeyValue<String,String>(mmsi,new_value);
        }
    }else{
        kvStore.delete(mmsi);
        return new KeyValue<String,String>(mmsi,String.format(startTimestamp.toString() + ','+
            currTimestamp.toString()+','+Long.toString(difference)));
    }
}
```

Figura 4.3: Heading Change Transformer

* Se dalla entry precedentemente inserita nel KeyValueStore e il messaggio corrente sono trascorsi meno di 10 minuti, viene estesa la finestra, inserendo la tupla (MMSI, (TimestampDiInizio, TimestampCorrente))
* Altrimenti, viene fatta partire una nuova finestra, inserendo la tupla (MMSI, TimestampCorrente)

– Altrimenti, viene cancellata la tupla corrispondente all'MMSI e viene ritornata la tupla (MMSI, (TimestampDiInizio, TimestampCorrente))

4.5.2 Within Area

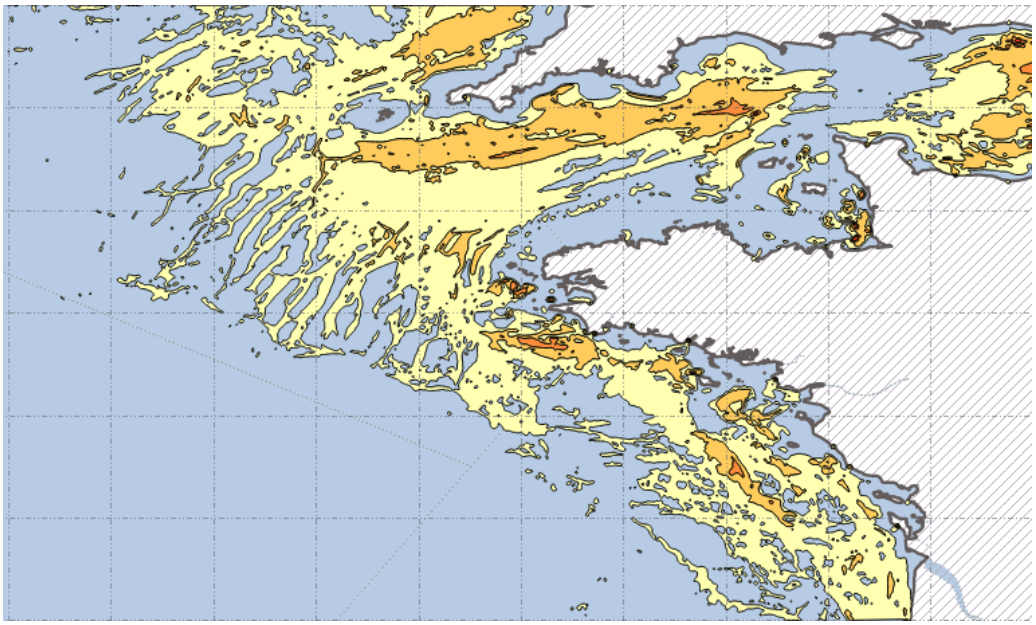

Figura 4.4: Fishing Area

```
String id = value.getId();
String new_value;
if (this.kvStore.get(id)==null){
    if (this.FishingArea.is_in_FishingArea(value)){
        new_value = String.format(value.getTimestamp() +','+ value.getTimestamp());
        kvStore.put(id, new_value);
        return new KeyValue<String,String>(id,new_value);
    }

} else {
    final String[] kvStoreData = kvStore.get(id).split(",");
    Long startTimestamp = Long.parseLong(kvStoreData[0]);
    Long precTimestamp = Long.parseLong(kvStoreData[1]);
    Long currTimestamp = Long.parseLong(value.getTimestamp());
    Long difference = (currTimestamp - startTimestamp)/(60 * 1000);


    if (this.FishingArea.is_in_FishingArea(value)){
        if (currTimestamp -precTimestamp <= Duration.ofMinutes(10).toMillis()){
            String new_value_insert =String.format(startTimestamp.toString() + ','+ currTimestamp.toString()+','+
                Long.toString(difference));
            kvStore.put(id,new_value_insert);
            return new KeyValue<String,String>(id,new_value_insert);
        }
        else{
            new_value =String.format(currTimestamp.toString() + ','+ currTimestamp.toString());
            kvStore.put(id,new_value);

            return new KeyValue<String,String>(id,new_value);
        }
    }else{
        kvStore.delete(id);
        return new KeyValue<String,String>(id,String.format(startTimestamp.toString() + ','+
            currTimestamp.toString()+','+Long.toString(difference)));
    }
}
```

Figura 4.5: Within Area Transformer

I vincoli fondamentali per tale evento sono: l'arrivo di un messaggio al massimo ogni 10 minuti (di modo da rispettare le condizioni di GAP definite dai ricercatori del progetto datAcron [9]) e che la nave si trova in un'area di pesca.

Per verificare se una nave si trova in un'area di pesca è stato utilizzato uno *shapefile* che identifica le zone di pesca nell'area di Brest. In particolare, ogni area è identificata da coordinate geografiche, espresse in termini di longitudine e latitudine, organizzate in poligoni. A tal fine è stata creata una classe *FishingArea* che verifica se un punto costituito dai valori di longitudine e latitudine di un *AIS_Message* è contenuto in uno dei poligoni definiti dallo shapefile. Se ciò si verifica, si ha che la nave che ha inviato quel particolare AIS_Message si trova in un'area di pesca.

Tale classe viene istanziata nel Transformer che segue tale logica:

- All'arrivo di un messaggio, se l'MMSI della nave non è presente nel KeyValueStore, esso inserisce la tupla (MMSI, Timestamp del messaggio), segnando, di fatto, l'inizio della finestra.
- Se esiste già una entry per MMSI:
  - se la nave si trova nell'area di pesca:
    - Se dalla entry precedentemente inserita nel KeyValueStore e il messaggio corrente sono trascorsi meno di 10 minuti, viene estesa la finestra, inserendo la tupla (MMSI, (TimestampDiInizio, TimestampCorrente))
    - Altrimenti, viene fatta partire una nuova finestra, inserendo la tupla (MMSI, TimestampCorrente)
  - Altrimenti, viene cancellata la tupla corrispondente all'MMSI e viene ritornata la tupla (MMSI, (TimestampDiInizio, TimestampCorrente))

4.5.3 Trawling Speed

L'evento di trawling speed è definito come un *fluent event* [10]. Tuttavia, è stato introdotto una finestratura in modo tale da poterlo confrontare con gli altri eventi per poterne studiare la contemporaneità e il conseguente effetto sull'evento composto *trawling*. La logica alla base è la seguente:

```
@Override
public KeyValue<String, String> transform(String key, AISMessage value) {

    String mmsi = value.getId();
    String new_value;
    if (this.kvStore.get(mmsi)==null){
        if (TrawlSpeedCheck(value, WindowTransfTrawlSpeed.SPEED_MIN, WindowTransfTrawlSpeed.SPEED_MAX)){
            new_value = String.format(value.getTimestamp() +','+ value.getTimestamp());
            kvStore.put(mmsi, new_value);
            return new KeyValue<String,String>(mmsi,new_value);
        }

    } else {
        final String[] kvStoreData = kvStore.get(mmsi).split(",");
        Long startTimestamp = Long.parseLong(kvStoreData[0]);
        Long currTimestamp = Long.parseLong(value.getTimestamp());
        Long difference = (currTimestamp - startTimestamp)/(60 * 1000);

        if (TrawlSpeedCheck(value, WindowTransfTrawlSpeed.SPEED_MIN, WindowTransfTrawlSpeed.SPEED_MAX)){

            String new_value_insert = String.format(startTimestamp.toString() + ','+ currTimestamp.toString());
            kvStore.put(mmsi,new_value_insert);
            return new KeyValue<String,String>(mmsi,new_value_insert+","+Long.toString(difference));

        }else{
            kvStore.delete(mmsi);

            return new KeyValue<String,String>(mmsi, String.format(startTimestamp.toString() + ','+
                currTimestamp.toString()+","+Long.toString(difference)));
        }
    }
```

Figura 4.6: Trawling Speed Transformer

- All'arrivo di un messaggio, se l'MMSI della nave non è presente nel KeyValueStore, esso inserisce la tupla (MMSI, Timestamp del messaggio), segnando, di fatto, l'inizio della finestra.
- Se esiste già una entry per MMSI:
  - se vengono rispettati i vincoli sulla velocità (velocità minima di 1 nodo e massima di 9 nodi) si procede all'estensione della finestra, inserendo la tupla (MMSI, (TimestampDiInizio, TimestampCorrente))
  - Altrimenti, viene cancellata la tupla corrispondente all'MMSI e viene ritornata la tupla (MMSI, (TimestampDiInizio, TimestampCorrente))

4.5.4 Trawling Movement

TrawlingMovement è un evento composto dai due eventi semplici *withinArea* e *headingChange*, infatti esso è vero fin quando l'imbarcazione da pesca in questione si trova in una zona di pesca (*withinArea*) ed esegue una sequenza di cambi di rotta (*headingChange*) ognuno dei quali avviene al più 10 minuti dopo il precedente. Tale evento quindi è il risultato del join dei due eventi semplici.

```
//TRAWILING MOVEMENT JOIN
KStream<String, String> trawlingMovement = whitinArea.join(headingChange,windowJoiner,
JoinWindows.of(Duration.ofMillis(100)),StreamJoined.with(Serdes.String(),Serdes.String(),Serdes.String()).withName
("Trawling-movement-join"));

//FILTER TRAWILING MOVEMENT WINDOW WITH LESS THAN 10 MINUTE
KStream<String, String> trawlingMovement_filter = trawlingMovement.filter(new Predicate<String,String>(){
    @Override
    public boolean test(String key, String value) {
        final String[] data = value.split(",");
        Long startTimestamp = Long.parseLong(data[0]);
        Long endTimestamp = Long.parseLong(data[1]);
        if(endTimestamp-startTimestamp>= Duration.ofMinutes(10).toMillis())
            return true;
        return false;
    }
});
```

Figura 4.7: Trawling Movement

4.5.5 Trawling

Trawling è un evento ottenuto dalla combinazione di un evento composto *TrawlingMovement* e un evento semplice *Trawling Speed*, infatti esso è vero fin quando l'imbarcazione da pesca in questione si trova in una zona di pesca e esegue una sequenza di cambi di rotta (*TrawlingMovement*) e mantiene una velocità nel range identificato dall'evento (*Trawling Speed*).

```
//TRAWLING JOIN
KStream<String, String> trawling = trawlSpeed.join(trawlingMovement_filter,windowJoiner,
JoinWindows.of(Duration.ofMillis(100)),StreamJoined.with(Serdes.String(),Serdes.String(),Serdes.String()).withName
("Trawling-join"));

trawling_result.filter((k,v) -> v != null).filter(new Predicate<String,String>(){
    @Override
    public boolean test(String key, String value) {
        final String[] data = value.split(",");
        Long startTimestamp = Long.parseLong(data[0]);
        Long endTimestamp = Long.parseLong(data[1]);
        if(endTimestamp-startTimestamp>= Duration.ofMinutes(15).toMillis())
            return true;
        return false;
    }
}).to(Application.TRAWLING_RESULT_TOPIC, Produced.with(Serdes.String(), Serdes.String()));
```

Figura 4.8: Trawling

Capitolo 5

Funzionamento del sistema

Di seguito viene riportato un esempio comparativo tra l'analisi sviluppata e i risultati emersi dal documento *Composite Event Recognition for Maritime Monitoring* [10].

Colore	Evento
Blu	Evento riconosciuto con durata superiore a 15 minuti
Rosso	Traiettoria intermedia tra due eventi di trawling
Giallo	Partenza e ritorno dal porto

Tabella 5.1: Legenda Figura 5.1 e Figura 5.2

5.1 Analisi 1

Si prende in considerazione la nave con questo MMSI 228109000. In particolare, si prende in esame il periodo di tempo tra Giovedi 22 Ottobre 2015 alle ore 5:55:29 AM (in millisecondi, 1445493329000), data di partenza dal porto di Brest, e Lunedi 26 Ottobre 2015 10:32:14 AM (in millisecondi, 1445855534000): si osserva la presenza di 27 eventi di trawling di duranta variabile, tutti al di sopra dei 15 minuti (eventi a durata inferiore sono stati ignorati). Il primo evento di trawling è stato registrato in data Venerdi 23 Ottobre 2015 alle ore 09:35:51 AM (in millisecondi, 1445592951000) e l'ultimo in data Domenica 25 Ottobre 2015 alle ore 9:58:04 PM (in millisecondi, 1445767084000).

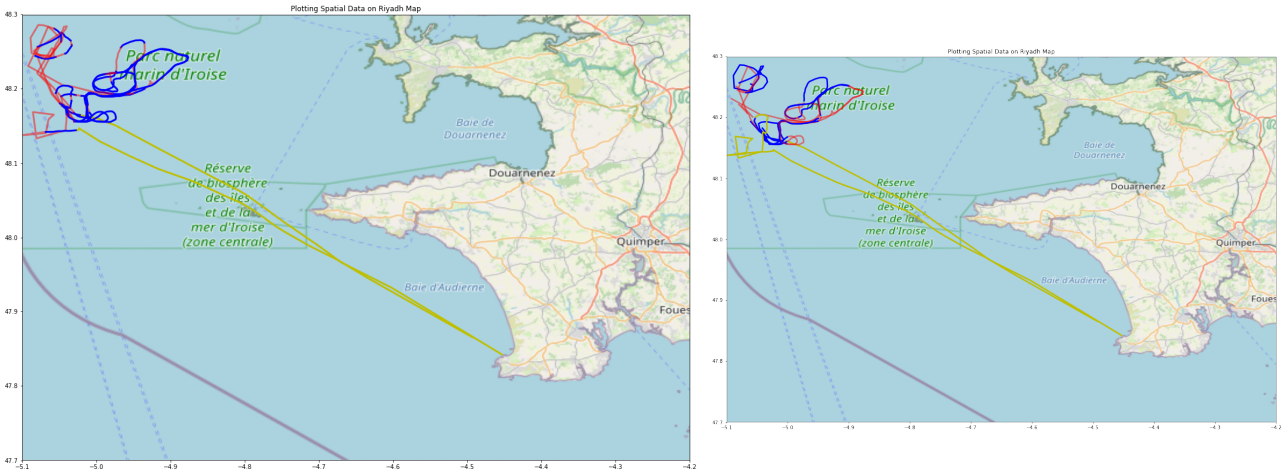

Figura 5.1: Risultati ottenuti

Figura 5.2: Risultati datACRON

Va precisato, quindi, che le traiettorie segnate in rosso in Figura 5.2, potrebbero contenere fenomeni di trawling di durata inferiore ai 15 minuti, impostati come soglia di sbarramento.

event	id	argume…	value	timestamp1	timestamp2
trawling	228109000		true	1445592951000	1445597411000
trawling	228109000		true	1445601571000	1445612201000
trawling	228109000		true	1445619190000	1445625251000
trawling	228109000		true	1445628651000	1445636271000
trawling	228109000		true	1445636282000	1445637602000
trawling	228109000		true	1445646702000	1445651101000
trawling	228109000		true	1445651351000	1445659916000
trawling	228109000		true	1445669590000	1445673342000
trawling	228109000		true	1445677060000	1445689731000
trawling	228109000		true	1445702961000	1445708080000
trawling	228109000		true	1445716621000	1445721441000
trawling	228109000		true	1445740231000	1445746633000
trawling	228109000		true	1445752380000	1445756255000
trawling	228109000		true	1445761820000	1445767084000

Figura 5.3: trawling risultanti datACRON

Confrontando gli studi condotti dal team a supporto del progetto datACRON e i risultati ottenuti si evince un comportamento parzialmente sovrapponibile. In particolare, la sostanziale differenza è che datACRON tende a unificare trawling temporalmente ravvicinati, mentre le analisi portate

event	id	start	stop
trawling	228109000	1445593999000	1445595590000
trawling	228109000	1445603780000	1445605280000
trawling	228109000	1445619189000	1445620491000
trawling	228109000	1445621280000	1445623109000
trawling	228109000	1445624010000	1445625289000
trawling	228109000	1445632710000	1445634769000
trawling	228109000	1445637940000	1445646530000
trawling	228109000	1445647931000	1445655059000
trawling	228109000	1445655201000	1445663389000
trawling	228109000	1445666170000	1445671280000
trawling	228109000	1445672040000	1445674780000
trawling	228109000	1445674840000	1445678839000
trawling	228109000	1445681479000	1445682410000
trawling	228109000	1445683070000	1445684120000
trawling	228109000	1445684900000	1445686899000
trawling	228109000	1445702311000	1445703800000
trawling	228109000	1445704640000	1445705950000
trawling	228109000	1445706620000	1445708079000
trawling	228109000	1445719190000	1445721440000
trawling	228109000	1445730660000	1445734861000
trawling	228109000	1445736390000	1445740879000
trawling	228109000	1445741660000	1445753530000
trawling	228109000	1445754199000	1445765690000
trawling	228109000	1445766711000	1445769239000
trawling	228109000	1445769891000	1445770910000
trawling	228109000	1445771330000	1445772359000
trawling	228109000	1445790709000	1445793249000

Figura 5.4: trawling risultanti dell'analisi

avanti durante gli sviluppi tendono a frammentare eventi di trawling che si presentano in un arco temporale ristretto.

Confrontando, infine, timestamp per timestamp, gli eventi di trawling rilevati da datAcron con quelli rilevati nel corso dell'analisi, si evince come questi ultimi siano sottoinsieme dei primi.

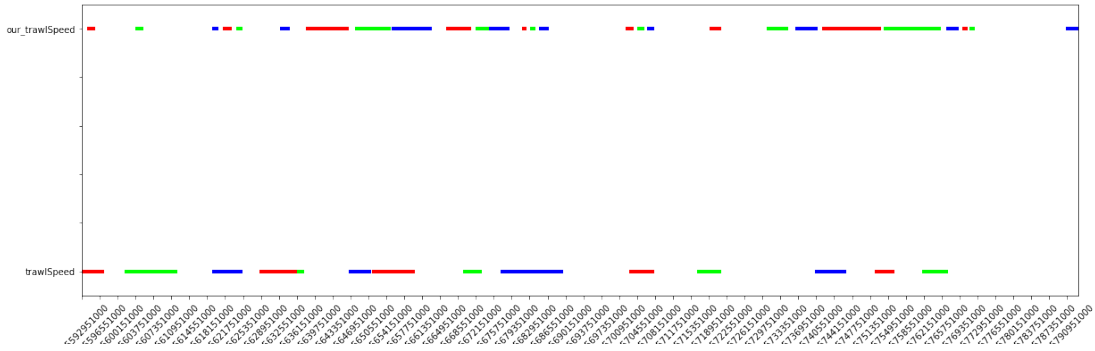

Figura 5.5: Confronto per timestamp

5.2 Analisi 2

Si considera la medesima imbarcazione nel caso precedente, ma nella battuta di pesca compresa tra Giovedi 1 Ottobre 2015 alle 12:14:42 PM (in millisecondi, 1443701682000) e Lunedi 5 Ottobre 2015 alle 10:08:12 AM (in millisecondi, 1444039692000). In Figura 5.6 è riportato il risultato dell'analisi. Si nota, nella traiettoria di ritorno al porto di Brest, un tratto segnalato in blu: seppur tale tratto è considerato un evento di trawling, in realtà esso potrebbe essere una semplice manovra di traiettoria. In particolare, l'imbarcazione:

- è una nave di pesca
- è posizionata in un'area di pesca
- sta effettuando *heading-change*
- ha una velocità compresa tra 1 e 9 nodi

Tuttavia, la forma della traiettoria descritta non sembra rimandare a un tipico movimento di pesca.

Questo esempio proposto ha un duplice significato. Da un lato dimostra come il sistema suggerisca all'operatore le presunte traiettorie di trawling, filtrando le informazioni necessarie alla sua individuazione. D'altro canto viene sottolineato come il sistema non sia in grado di sostituire completamente l'operatore, ma, al contrario, esso è parte integrante del loop di controllo ed è demandata all'operatore stesso la decisione riguardante la classificazione di tali eventi.

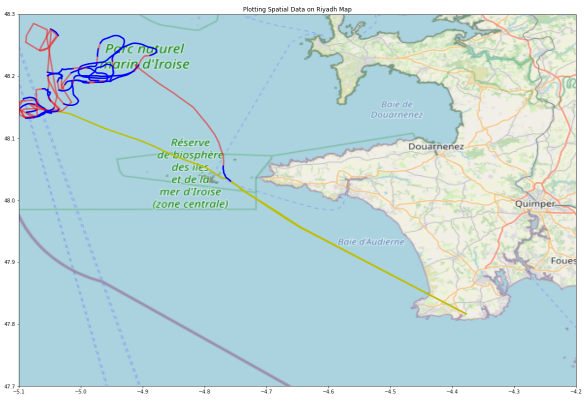

Figura 5.6: Risultati ottenuti

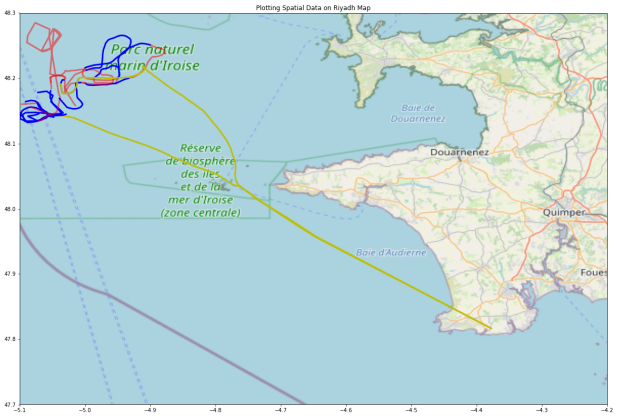

Figura 5.7: Risultati datACRON

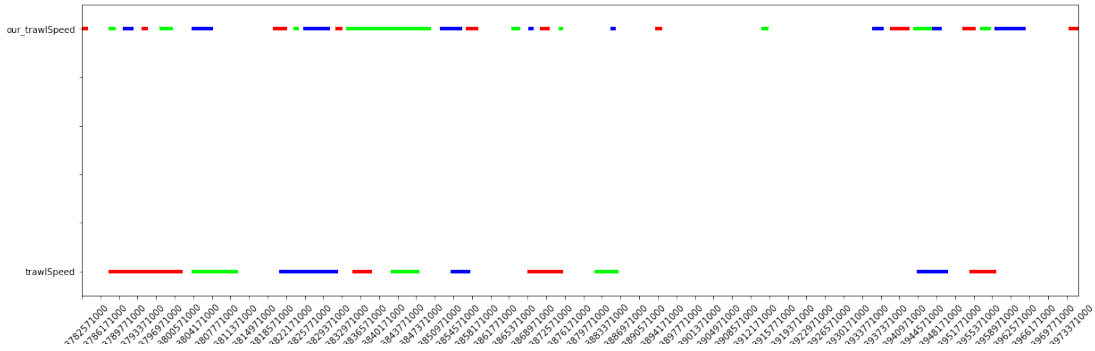

Figura 5.8: Confronto per timestamp

5.3 Analisi 3

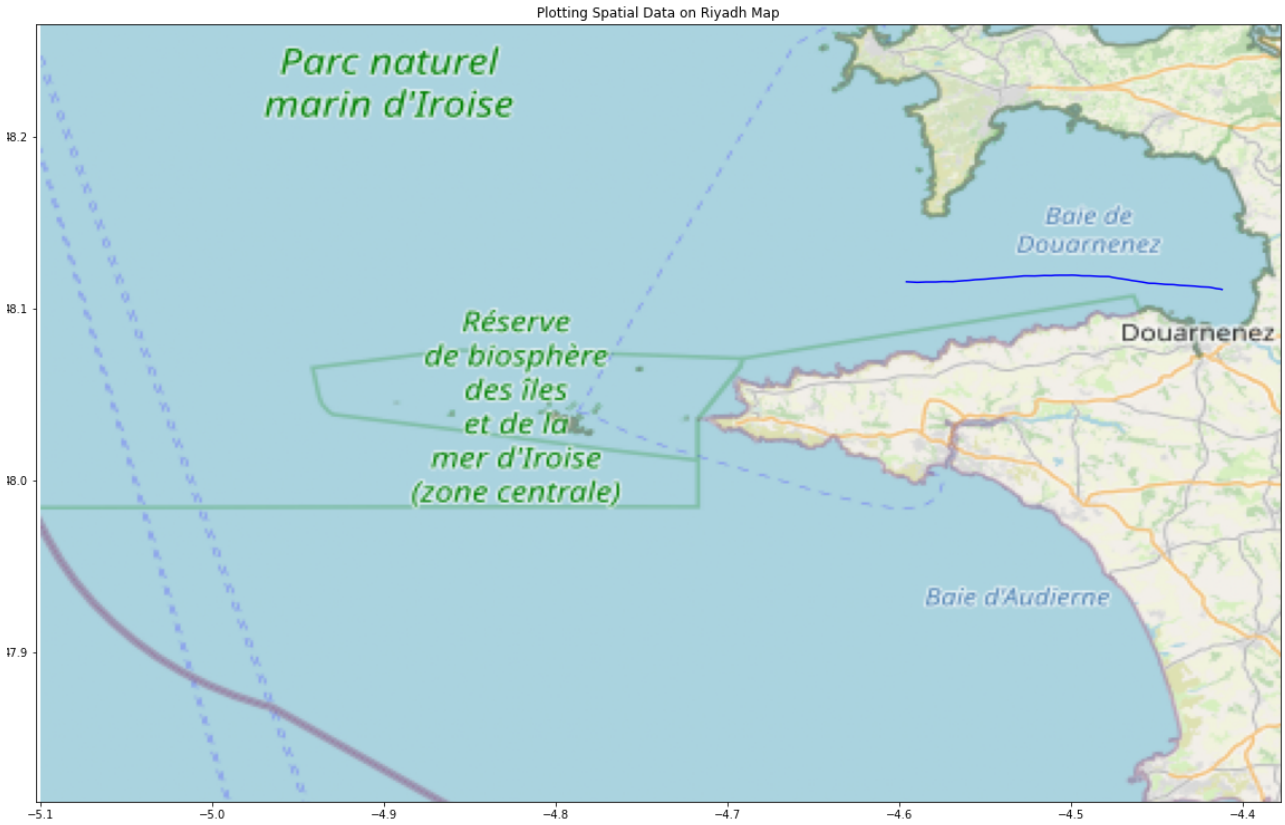

Figura 5.9: MMSI: 228015700

Caso particolare è l'imbarcazione con MMSI: 228015700, nel periodo che intercorre tra Lunedi 5 ottobre 2015 alle ore 4:24:58 AM (in millisecondi, 1444019098000) e Lunedi 5 ottobre 2015 5:31:26 AM (in millisecondi, 1444023086000).

Dalle analisi condotte è emerso che essa rispetta tutte le condizioni necessarie per affermare che essa stia effettuando trawling. Tuttavia, cercando un riscontro nel dataset di datAcron, tale evento risulta essere del tutto assente: si suppone che la traiettoria descritta dall'imbarcazione non venga considerata come facente parte di un'area di pesca, mentre nel dataset *v_recode_fish_area_clean*, utilizzato per le nostre analisi sull'evento *within-area*, tale condizione risulta verificata.

Capitolo 6

Sviluppi futuri

Alcuni sviluppi futuri coinvolgono la creazione di un'interfaccia grafica dinamica. In particolare, si potrebbe realizzare un ulteriore nodo Kafka processore in grado di leggere i messaggi del topic *output* e disegnare le traiettorie che le navi stanno compiendo, man mano che nuovi messaggi vengono ricevuti.

Tale interfaccia deve tener conto delle best-practises e delle metodologie ottimali di progettazione, al fine di supportare in maniera ottimale il decisore.

Ulteriori analisi possono essere condotte sui vincoli imposti per renderli quanto più simili ai vincoli utilizzati nel progetto datAcron.

Bibliografia

[1] *Datatypes*. https://kafka.apache.org/documentation/streams/developer-guide/datatypes.html.

[2] *Deserializer*. https://kafka.apache.org/27/javadoc/org/apache/kafka/common/serialization/Deserializer.html.

[3] *GeoTools*. https://it.wikipedia.org/wiki/GeoTools.

[4] *Kafka documentation*. https://kafka.apache.org/27/documentation/streams/developer-guide/dsl-api.html.

[5] *Kafka key concepts*. https://kafka.apache.org/documentation/.html.

[6] *Key Value Store*. https://docs.confluent.io/platform/current/streams/concepts.html.

[7] *Matplotlib Tool Python*. https://it.wikipedia.org/wiki/Matplotlib.

[8] *Pandas Tool Python*. https://it.wikipedia.org/wiki/Pandas_(software).

[9] Kostas Patroumpas et al. «Online event recognition from moving vessel trajectories». In: *GeoInformatica* 21.2 (2017), pp. 389–427.

[10] Manolis Pitsikalis et al. «Composite event recognition for maritime monitoring». In: *Proceedings of the 13th ACM International Conference on Distributed and Event-based Systems*. 2019, pp. 163–174.

[11] *Serializer*. https://kafka.apache.org/27/javadoc/org/apache/kafka/common/serialization/Serializer.html.

[12] *Stateful Stream Processing*. https://docs.confluent.io/platform/current/streams/concepts.html.

[13] *TimestampExtractor*. https://kafka.apache.org/27/javadoc/org/apache/kafka/streams/processor/TimestampExtractor.html.

[14] *Transformer*. https://kafka.apache.org/27/javadoc/org/apache/kafka/streams/kstream/Transformer.html.

[15] *TransformerSupplier*. https://kafka.apache.org/27/javadoc/org/apache/kafka/streams/kstream/TransformerSupplier.html.