Combining Ship Trajectories and Semantics with the Simple Event Model (SEM)

Willem Robert van Hage
Web & Media Group
VU University Amsterdam
wrvhage@few.vu.nl

Véronique Malaisé
Web & Media Group
VU University Amsterdam
vmalaise@few.vu.nl

Gerben de Vries
HCS Group
University of Amsterdam
G.K.D.deVries@uva.nl

Guus Schreiber
Web & Media Group
VU University Amsterdam
schreiber@cs.vu.nl

Maarten van Someren
HCS Group
University of Amsterdam
M.W.vanSomeren@uva.nl

ABSTRACT

Bridging the gap between low-level features and semantics is a problem commonly acknowledged in the Multimedia community. Event modeling can fill the gap. In this paper we present the Simple Event Model (SEM) and its application in a Maritime Safety and Security use case about Situational Awareness. We show how we abstract over low-level features, recognize simple behavior events using a Piecewise Linear Segmentation algorithm, and model the events as instances of SEM. We apply deduction rules, spatial proximity reasoning, and semantic web reasoning in SWI-Prolog to derive abstract events from the recognized simple events. The use case described in this paper come from the Dutch Poseidon project.

Categories and Subject Descriptors

E.0.e [**General**]: Knowledge and data engineering tools and techniques; I.2.1 [**Artificial Intelligence**]: Applications and Expert Knowledge-Intensive Systems; I.2.3.f [**Artificial Intelligence**]: Deduction and Theorem Proving and Knowledge Processing—*Logic Programming*; I.2.6.f [**Artificial Intelligence**]: Learning—*Knowledge acquisition*

General Terms

Event modeling, Piecewise linear segmentation, Prolog, Semantic web, Maritime safery and security, Situational awareness

1. INTRODUCTION

The notion of "bridging the gap" [13] is well known in the Multimedia field: the missing chain link between low-level data (*e.g.* features extracted from a video, or in the case of this paper, ship movement) and semantics. Event modeling can fill the gap. We show how the Simple Event Model (SEM) can be used as a semantic layer over abstractions derived from domain-level raw data. The domain-level data are Marine Automatic Identification System (AIS)[1] messages, sent by ships at a regular interval to receivers. AIS messages post the ship's navigation parameters. We describe a method to recognize meaningful events in this ship behavior data, and to model them as instances of SEM. We write rules in SWI-Prolog [16] that define the semantics of these events, and integrate them with GeoNames[2] concepts. We use the rules to classify the behavior of ships and derive new events, like ferry trips.

We present related work in section 2, before introducing SEM itself in section 3. We continue with the description of our use case: the automatic generation of SEM Events from AIS messages for Situational Awareness in section 4. We conclude and discuss future work in section 5.

2. RELATED WORK

Different models have been proposed to bridge the gap between domain-level features and the *semantic* level. Part of the MPEG-7 [6] Multimedia Description Scheme, for example, contains the two aspects. The model is complex and linking the low-level to semantics via MPEG-7 *itself* is hardly ever done. The usual approach is to combine MPEG-7 with an ontology [4, 14]. COMM [1] allows combination of descriptions from MPEG-7 with a semantic description based on the DOLCE [7] and its extension, the Description and Situation pattern. COMM leaves the choice of the semantic description model to the user. It provides a place holder for semantic descriptions that can be filled by either a single item or a complex description, for example, using the *Event Ontology* [10]. SEM proposes an extension to this ontology with additional Role and (external) Type classes. This last point is where SEM differs from other "class-based" Event models (*cf.* [10] and [2]): models that describe events with classes and properties. These models can be extended by creating subclasses. In SEM, we allow (and encourage!) the use of *external type* definitions, from third-party vocabularies as Types for our core classes, instead of this subclassing mechanism. This third solution stands inbetween

[1] `http://en.wikipedia.org/wiki/Automatic_Identification_System`
[2] `http://www.geonames.org/`

"class-based" and "property-based" event models, like the model behind the CultureSampo portal [12]. Ruotsalo et al. model events with properties between classes from external vocabularies, mapped to a common upper level ontology, but do not define event classes *as such*.

3. SEM: SIMPLE EVENT MODEL

The purpose of SEM is to provide the minimal set of classes to describe events, but with two specific requirements: the possibility to associate roles and (possibly external) types to these classes. We chose to have as few constrains as possible for the sake of compatibility with (1) different, possibly more constrained, event models or ontologies and (2) with fuzzy real world data, where the semantics is not as well defined as in formal models.

SEM has five core classes: Event, Actor, Object, Place and Role. We modeled the core entities as Classes, which are related together with the properties shown in figure 1. What is not displayed on the figure is the fact that all five core Classes can have a Role, not only the Actor. These Classes are sub-classes of TimeStampedEntity: their existence can be bounded in time, but this temporal definition is not mandatory. Another important part of SEM is the different Types corresponding to the core Classes: EventType, ActorType, ObjectType, PlaceType and RoleType. We also modeled them as Classes. Types are not time-stamped: they are generic notions which instances are mobilized in the context of an event description. The instances of the Type classes can be taken from third-party ontologies and can be either instances or classes in those ontologies. In the following paragraphs we present the five core classes, SEM's Type system and the way we model time: as a datatype.

Event. The *Event* is the class meant to describe "what" is happening: being anchored, for example. Events are related to Actors via the generic property hasParticipant or its inverse property participatesIn: a further specification of *how* an Actor is participating in an Event is made via the specification of the Actor's Role (see *Role* below). An Object is involvedIn an Event if it is not an active participant; things (even physical objects like a ship) are Actors if they are directly participating in the Event. The kind of an Event (standing still) can be denoted with an instance of an EventType. Modeling via EventType (see the *Type system* paragraph) is recommended but is not enforced in SEM. EventType instances can be borrowed from external vocabularies. Events can be time-stamped with the timeStampedAt datatype property and associated to a Place with the inPlace property. See the following *Time* and *Place* paragraphs for a more elaborate description.

Events correspond to entities defined primarily via their temporal aspect, like the Perdurants/Occurents of DOLCE [7], the Process class of SUMO [9] and the Situation of CYC [8]. Dublin Core defines an Event class only in the DCMI Type category, as a recommended value for a document's genre. No further constrain is given. In the CultureSampo [12] knowledge representation, the Event class itself is not defined within their model: they capture the relations between an Event form an external vocabulary to other components of the event, like the agent, time, place. As mentioned above, in SEM, an external vocabulary describing the Event is linked via the EventType. The Event Ontology[3] defines an Event class, which can have other Events as parts: we created the same "part-of" property, but in our case it can be applied to any of our classes. We followed the Event Ontolgoy by not restricting this class. In this way, we stay as broad in scope and as generic as possible. If constrains are necessary for give use cases, they can be added as local extensions. The same goes for the mapping to a higher level ontology: we provide links to concpets from some upper ontologies but do not enforce any particular high level semantic choice in SEM.

Actor. The *Actor* is the class meant to describe "who" is doing something (and on whom it is happening): a ship, for example. Actors are related to Events via the generic property participatesIn: a further specification of *how* an Actor is participating in an Event is made via the specification of the Actor's Role (see *Role* below). The kind of an Actor (tanker, passenger vessel) can be denoted with an instance of an ActorType, which can be borrowed from external vocabularies. Actors can be time-stamped with the timeStampedAt datatype property and associated to a Place with the inPlace property.

Actor corresponds to a class of entities defined primarily through their spatial dimension[4], like the Endurants / Continuants from DOLCE [7]. This class corresponds to Agent in SUMO [9] and Agent-Generic or SomethingExisting in CYC [8], as basically anything that is not an event can be an Actor. Dublin Core has the two classes Agent and AgentClass in the dcterms vocabulary to specify Actors from an Event. In the model of the CultureSampo Portal, three properties correspond to the link between our Event and Actor classes: the generic *participant*, *agent* and *patient*. In SEM we have only the generic property; the fact that an Actor is participating in an Event as an agent, patient or any other variation is modeled via the Role and RoleType classes (see below). In the Event Ontology, the suggested value corresponding to our Actor class is the Agent class from the FOAF vocabulary[5], which also corresponds to our view: a "person, group, software or physical artifact".

Object. The *Object* is the class meant to describe "with what" is something being done: this can range from physical objects like a boat compass to animate entities when they are involved in a situation as instrument, in extreme cases. Objects are related to Events via the generic property involvedIn: a further specification of *how* an Object is participating in an Event is made via the specification of the Objects Role (see *Role* below). The kind of an Object (navigation instrument) can be denoted with an instance of an ObjectType, which can be borrowed from external vocabularies. Objects can be time-stamped with the timeStampedAt datatype property and associated to a Place with the

[3]`http://motools.sourceforge.net/event/event.html`

[4]Although an Actor can also be a fictional character or a legal entity, all of the parts of the character exist at all of the moments of its existence, unlike events.

[5]The FOAF language is defined as "a dictionary of named properties and classes", `http://xmlns.com/foaf/spec/`

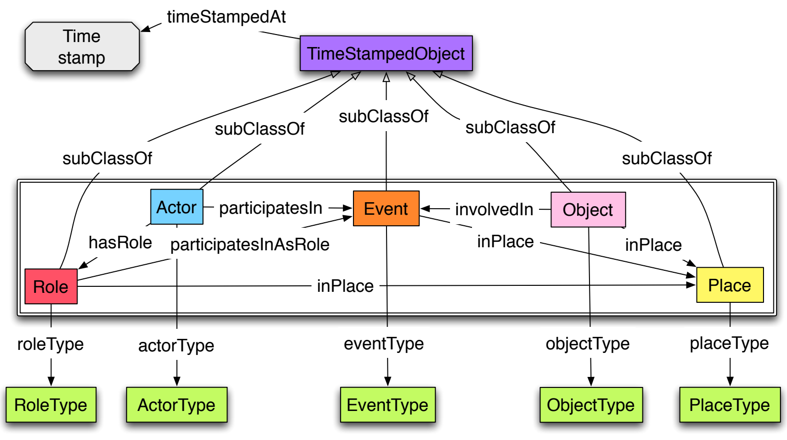

Figure 1: The Simple Event Model, SEM. The core of SEM is constituted by the 5 classes: Event, Actor, Role, Object, Place. These are sub-classes of TimeStampedEntity, from which they inherit the possibility to be time stamped. TimeStampedEntity does not have instances. The boxes in green represent the different typing classes. They are not mandatory, but they are provided as anchors for external vocabularies.

inPlace property.

Like Actor, Object corresponds to a class of entities defined primarily through their spatial dimension, like the Endurants / Continuants from DOLCE [7]. In SUMO [9], our notion of Object is also represented by Object; it corresponds to SomethingExisting in CYC. Dublin Core has two classes for our notion of Object: PhysicalResource and PhysicalObject. In CultureSampo and in the Event Ontology, different views on the objects are taken into account: instrument and goal in the former, event:Factor and event:Product in the latter. These distinctions, although represented in a different ways (respectively as properties and classes) can be modeled in SEM in the same manner: via the Role and RoleType of the Object.

Place. The *Place* is the class meant to describe "where' is something happening: this can range from concepts (like the ones from GeoNames[6]) to strings (like Dublin Core's DCMI Box[7] or Point[8] scheme) or simple geographical coordinates (like the ones defined in WGS84[9] or GeoRSS and GML[10], for example). Places are related to Events via the generic property inPlace: every class used as range for this property is classified as a Place. It can be a physical object (in the harbor) or an imaginary place (Neverland), etc. The kind of a Place (port, harbor) can be denoted with an instance of a PlaceType, which can be borrowed from external vocabularies. Places can be time-stamped with the timeStampedAt datatype property and associated to an Object and a Role with the inPlace property.

The Place is modeled via a complex pattern in DOLCE [7]: it is a SpatialRegion related to the SpatialQuality of a Class. This pattern is derived from the way Natural Language is used to refer to Place. Although it is conceptually less precise than DOLCE, a simple triple involving the classes of Event and the Place is sufficient for our needs, so we opted for a simpler representation in SEM. This simple representation of Place as a single class is also commonly accepted (GeographicThing class in SUMO and EnduringThing-Localized in CYC). Dublin Core proposes two mechanisms: the class Location in dcterms, and a string composed of defined attributes: the DCMI Period[11]. The CultureSampo model defines one property (place), which suggests that they also vote for the option of the simple triple to represent places, although they use DOLCE as a reference to align different vocabularies. The Event Ontology recommends the use of GeoNames as a value for the Place. We subsume this choice, enabling also for locations with unclear boundaries, or boundaries changing over time (a coastline), imaginary ones or any kind of entity that is witnessing an event: a coastguard station for example.

Role. The *Role* represents the function that is played by an instance of one of SEM's core classes, in the context of a given event. Roles are time-stamped: this way it is possible to represent the fact that the classes' instance and its role have a different time span. For example, a ship's Captain has a birth date that does not coincide with the date at which he started his role as captain. In the same fashion, Amsterdam is a city since the late 12th century, but is the capital of the Netherlands only since 1815; an object can have a ritual function only at a given moment in time, during a religious ceremony for example. Instances can have different Roles, depending on the Event in which they participate: the ship's Captain is a husband at home, each Role having its own independent time-span. Roles are directly related to Places: this allows to describe that a person with the Role of Captain is in the command cabine, and participates in the Event "Ferry Trip" in the Netherlands. The

[6]`http://www.geonames.org/`
[7]`http://dublincore.org/documents/dcmi-box/`
[8]`http://dublincore.org/documents/dcmi-point/`
[9]`http://www.w3.org/2003/01/geo`
[10]`http://georss.org/gml`
[11]`http://dublincore.org/documents/dcmi-period/`

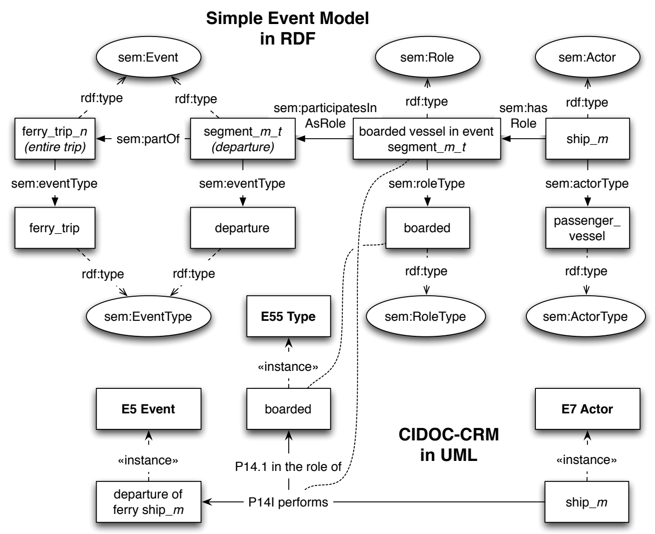

Figure 2: The modeling of Roles in SEM compared with their modeling in CIDOC-CRM

notion of Role is modeled in CIDOC-CRM as a reification of the property of an Actor participating in an Event: we use the same modeling principle, with the Role concept at the place of the reification. Figure 2 shows the similarity in the modeling, despite the fact that different representation paradigms are being used for the two examples.

Type system. One of the design strength of SEM is to allow the connection of the core classes to Types that can be defined by a third party. Although it can seem a costly modeling option, this solution is actually a generalization over Alan Rector's patterns for Value Partition and Value Sets[12], in the W3C Best Practices group. This way, we can use types from vocabularies that are designed as Value Sets as well as Value Partition. Figure3 shows Rector's example patterns along with the representation in SEM for this use case. There are two possible ways of relating the core classes to their corresponding Types: by using rdf:type or via our custom properties (eventType, actorType, roleType, objectType and placeType). The reason to have created such sub-properties is that some types can be domain-specific, and thus can convey a very different meaning than more "generic" types. For example, an actor Ship can have an actorType "Oil Tanker" and have an rdf:type PhysicalEntity. Intuitively, the scope of these two types is different. Types are not time-stamped, and their use is recommended but not mandatory.

Time. The *Time* is the datatype meant to describe "when" is something happening: this can range from precise dates (Thursday October the 12th at 14h30) to fuzzy (Estimated Time of Arrival) or even imaginary time spans. TimeStampedEntities are related to any kind of XML Schema datatype representing time-stamps via the datatype property timeStampedAt.

[12]http://www.w3.org/TR/swbp-specified-values/

DOLCE's pattern for Time is similar to the pattern used to describe a Place: a TemporalRegion as value for a TemporalQuality related to an Entity. This pattern is also derived from the way Natural Language is used to refer to Time: people can refer to the quality of time as well as to some region in the temporal space. Here again, our needs are fulfilled by a simpler representation. Unlike SUMO, CYC and the Event Ontology which require or recommend classes as value for Time (the W3C Time Ontology[13] for the latter), we do not need more than an XML Literal representation of a time-stamp. Indeed, we do not state anything about Time itself, only about TemporalEntities, hence we do not require a class. Modeling via a datatype value makes SEM simpler. Dublin Core defined one pattern to express time points as strings: conforming to the DCMI Period syntax allows to be inter-operable with other systems. We recommend the use of TIMEX2[14] or W3CDTF[15] as value to represent time-stamps. They can both be specified as a schema in the DCMI Period schema, and they subsume a controlled vocabulary to express Time (ISO 8601). They allows to represent expressions as precise as complex dates and as fuzzy as periods. It is even possible to represent mythical and relative time with this format. If a use case requires it, it is also possible to extend SEM with a sub-property and use a system based on classes (like the W3C Time Ontology) instead of our default datatype system.

3.1 Accessibility and Extension

SEM is accessible online at the URL: `http://semanticweb.cs.vu.nl/2009/04/event/`. It is informally mapped to a set of event models (Event Ontology, CultureSampo, Dublin Core, CIDOC-CRM) and of commonly used upper level ontologies: DOLCE, SUMO and CYC. This set of proposed mappings has been modeled in SKOS[16]; the extended SEM is available online[17].

4. USE CASE: MARITIME SITUATIONAL AWARENESS

We describe a Semantic Web application in which we automatically recognize events in domain-level data representing ship trajectories. From these atomic events, modeled as SEM instances, we derive ship behavior types (slowing down, speeding up, anchored) to reason about patterns: ship maneuvering when approaching an anchorage.

We transform the sensor-data trajectories into movement predicates using a compression algorithm. We use GeoNames[18] as an ontology of maritime geographical concepts and represent the movement predicates with SEM. We define rules over these two sources of knowledge, that allow us high-level reasoning about the behavior of ships. These rules, movement predicates and ontology are all integrated in SWI-Prolog [16] using its Semantic Web libraries[19] and

[13]http://www.w3.org/TR/owl-time/
[14]http://fofoca.mitre.org/
[15]http://www.w3.org/TR/NOTE-datetime
[16]http://www.w3.org/2004/02/skos/
[17]http://semanticweb.cs.vu.nl/2009/04/eventExtended/
[18]http://www.geonames.org
[19]http://www.swi-prolog.org/pldoc/package/semweb.html

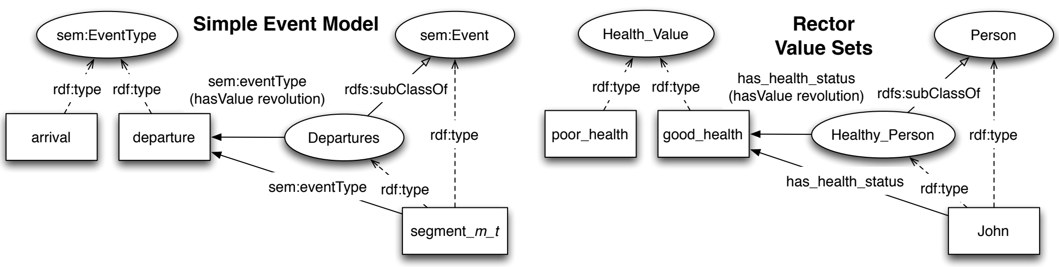

Figure 3: The modeling of Types in SEM as value-sets (on the left), compared with their modeling in W3C Best Practices by Alan Rector (on the right)

the spatial index that we have developed.

4.1 Abstraction of Trajectory Data

Trajectories. The ship trajectory data in our application comes from the Automatic Identification System (AIS). Each commercial vessel larger than 300 tons carries an AIS transponder. This transponder sends updates at regular intervals (in the order of seconds) about, among other things, the ship's location, speed over ground and course over ground. We will use these three time-series to extract movement predicates. Let $T_l = ((x_1, y_1), t_1), \dots, ((x_n, y_n), t_n)$ be a ship's location time-series, $T_s = (s_1, t_1), \dots, (s_n, t_n)$ its speed over ground time-series and $T_c = (c_1, t_1), \dots, (c_n, t_n)$ its course over ground time-series. The three time-series have samples at the same time-points, however, the AIS samplerate is not fixed.

As these time-series are from ships, they describe movements of relatively large objects. Such large objects are constrained in possible trajectories, e.g. large objects do not jump around, nor turn and accelerate very fast. In a sense, this type of movement data is highly regular and is quite predictable.

Piecewise Linear Segmentation. The above mentioned regularity of the time-series suggests that they can be compressed quite well using a piecewise linear representation. As we will see, such a compression is also easily convertible to a predicate representation. Thus, as a first step to creating movement predicates we compress the trajectory time-series using an algorithm that creates a piecewise linear segmentation. As is mentioned in [5] the variant that we use goes by many names. Especially the two-dimensional variant is well known, it is called the Douglas-Peucker algorithm [3] in carthography and Ramer's algorithm [11] in image processing. In machine learning it is also known as "iterative end-points fit" [5]. This algorithm compresses a time-series into linear segments by recursively keeping the points that have maximum error higher than a fixed threshold. In the form that we have defined it in algorithm 1, the algorithm does not return the value time-point pairs, but just the time-points, since we only need those in the next step.

The function $\mathbf{error}((v_i, t_i), ((v_1, t_1), (v_n, t_n)))$ computes the error that is produced when the points (v_1, t_1) and (v_n, t_n) are used to (linearly) represent the time-series between t_1 and t_n and the point (v_i, t_i) is left out. For the location time-series this function is defined as follows:

Algorithm 1: **PLS**(T,ϵ)

```
Data: A time-series: $T = (v_1, t_1), \dots, (v_n, t_n)$, and the
      maximum allowed error: $\epsilon$.
Result: A list of time-points that are kept: $T^c$.

$d_{max} = 0$, $i_{max} = 0$
for $i = 2$ to $n - 1$ do
    $d = \mathbf{error}((v_i, t_i), ((v_1, t_1), (v_n, t_n)))$
    if $d > d_{max}$ then
        $i_{max} = i$
        $d_{max} = d$
    end
end
if $d_{max} \geq \epsilon$ then                    // recursive calls
    $T^A = \mathbf{PLS}((v_1, t_1), \dots, (v_{i_{max}}, t_{i_{max}}), \epsilon)$
    $T^B = \mathbf{PLS}((v_{i_{max}}, t_{i_{max}}), \dots, (v_n, t_n), \epsilon)$
    $T^c = t^A_1, \dots, t^A_{m-1}, t^B_1, \dots, t^B_k$
else
    $T^c = t_1, t_n$
end
return $T^c$
```

$$\mathbf{error}(((x_i, y_i), t_i), (((x_1, y_1), t_1), ((x_n, y_n), t_n))) = \sqrt{(x_i - x'_i)^2 + (y_i - y'_i)^2} \ ,$$

$$\text{where } (x'_i, y'_i) \text{ is the orthogonal projection of } (x_i, y_i) \text{ onto the line through } (x_1, y_1) \text{ and } (x_n, y_n) \ . \quad (1)$$

In other words, we compute the orthogonal distance between the point (x_i, y_i) and the line given by (x_1, y_1) and (x_n, y_n)[20]. This is the error measure that is also used in the Douglas-Peucker algorithm [3].

For the speed over ground time-series we define this function as:

$$\mathbf{error}((s_i, t_i), ((s_1, t_1), (s_n, t_n))) = \|s_i - s'_i\| \ ,$$

$$\text{where } s'_i = s_1 + \frac{t_i - t_1}{t_n - t_1}(s_n - s_1) \ . \quad (2)$$

This error is the difference between the actual speed over ground value (s_i) at t_i and the linearly interpolated value

[20]In this case the location time-series is treated as a two-dimensional line, i.e. the time dimension is ignored.

(s_i') based on the values (s_1 and s_n) at t_1 and t_n.

The error function for the course over ground time-series is defined analogously to the one for the speed over ground time-series.

Conversion to Movement Predicates. Based on the piecewise linear segmentation described above, we create movement predicates that we call segments. Let T_l, T_s and T_c be the three time-series, corresponding to a ship's trajectory, that we mentioned earlier. Then, using algorithm 1 we can construct the set:

$$TP = \mathbf{PLS}(T_l,\epsilon_l) \cup \mathbf{PLS}(T_s,\epsilon_s) \cup \mathbf{PLS}(T_c,\epsilon_c) \quad . \tag{3}$$

The values of ϵ differ per type of time-series.

Essentially, the set TP contains all the time-points that we want to keep. These points represent a change in ship behavior, between two of those points the behavior of a ship is regarded as constant. Based on this idea, using TP, we create segment predicates for a ship trajectory in the following way:

$$\begin{aligned} S = \{ & \mathit{segment}(u, l_i, l_j, s_i, s_j, c_i, c_j, t_i, t_j) \mid t_i, t_j \in TP \\ & \wedge t_i < t_j \wedge \neg\exists t_k (t_i < t_k < t_j) \wedge (l_i, t_i), (l_j, t_j) \in T_l \\ & \wedge (s_i, t_i), (s_j, t_j) \in T_s \wedge (c_i, t_i), (c_j, t_j) \in T_c \} \quad . \end{aligned} \tag{4}$$

Note that l is a short-hand for (x, y). We take two consecutive time-points from TP and take the location, speed and course values corresponding to those time-points from the time-series, together they make up a segment. For each segment we generate a URI, u, based on the ship's unique identifier, the Maritime Mobile Service Identity number, and the start time (t_i), and hence uniquely identifies the segment. This URI, u, is used as the identifier of the event that corresponds to the ship's behavior in the segment.

Segments as Events in SEM. We assign types and SEM properties to the URIs of the segments to define their semantics. Every segment starts out with one type, rdf:type sem:Event. Additional (external) types are added by evaluating Prolog deduction rules that reason about the behavior during the segment. Simple event types are added by attaching an sem:eventType property to the segment that refers to the segment's event type. For example, when a segment has a low start speed and a high end speed, the triple assigning the type "speeding up" to the segment ⟨u, sem:eventType, poseidon:speeding_up⟩ is added into the knowledge base. Complex events, like ferry trips, that aggregate segments, are added as new Events of which the simple events are parts. For example, the event poseidon:ferry_trip_m_n has sem:eventType poseidon:ferry_trip and segments are added to this trip by asserting triples like ⟨u, sem:partOf, poseidon:ferry_trip_m_n ⟩. This is illustrated in figure 4 on line 6. Hierarchical relations between event types are represented with the sem:parentType property. For example, poseidon:anchored has sem:parentType poseidon:stopped. The location of events is attached to the segment using properties from the W3C WGS84 vocabulary. This is illustrated in figure 4 on line 11. Time is represented in TIMEX2 format as an XML Literal attached to the segment with sem:beginsAt and sem:endsAt, both subproperties of sem:timeStampedAt. This is illustrated in figure 4 on line 13 to 16.

```
 1 poseidon:mmsi_xxxxxxxx a sem:Event ;
 2     sem:eventType seg:AISsegment ;
 3     % low-level behavior semantics
 4     sem:eventType poseidon:departure ;
 5     % high-level behavior semantics
 6     sem:partOf poseidon:ferry_trip_xxxxxxxx_6 ;
 7     seg:beginsAtPlace [
 8         a sem:Place ;
 9         % classified as a harbor due to proximity to
10         % geos:2750318, see line number 22
11         wgs84:lat "52.9786" ;  wgs84:long "4.7843" ;
12     ] ;
13     sem:beginsAt "<timex2object>
14         <timex2 VAL="2008-08-04T03:00">
15             2008-08-04T03:00 LT
16         </timex2object>"^^<rdf:parseType="Literal"> ;
17     sem:involves poseidon:ship_xxxxxxxx .
18
19 poseidon:ship_xxxxxxxx a sem:Actor ;
20     sem:actorType poseidon:atype_passenger_vessel ;
21     ais:name "USS Enterprise" ;
22     ... ;
23     ais:mmsi "xxxxxxxx" .
24
25 # matched to the segment location by proximity
26 geos:2750318 a geo:Feature ;
27     geo:name "Nieuwe Haven" ;
28     geo:parentFeature geos:2749879 ;
29     wgs84:lat "52.966" ;  wgs84:long "4.783" ;
30     ... ;
31     geo:featureCode geoo:H.HBR .
```

Figure 4: A ship behavior segment modeled in SEM and an associated GeoNames Feature. In this case, the ship is at a harbor.

```
% semantic classification of ferry trip behavior
classify_ferry_behavior :-
    % define ferry trip event
    rdf_bnode(Trip),
    rdf_assert(Trip, rdf:type, poseidon:ferry_trip),
    % find instances of ferry trips with ferry_trip/6
    findall(trip(A,B), ferry_trip(A,_,_,_,B,_), Trips),
    forall(member(trip(From, To), Trips),
           % make intermediate segments part of the trip
           (   seg_from_to(From , To, Segments),
               forall(member(S, Segments),
                      rdf_assert(S, sem:partOf, Trip))
           )).

% rule defining ferry behavior semantics
% trip from Harbor H0  to H1 and back
% via Segment S0, S1, and S2
ferry_trip(S0, H0, S1 ,H1, S2, H0) :-
    trip(S0, H0, S1, H1),
    trip(S1, H1, S2, H0).

trip(S0, H0, S1, H1) :-
    % ship is stopped at harbor H0 during S0
    stopped_at_harbor(S0, H0),
    % in successive segments N..M it is moving
    seg_succ(S0, N),
    seg_succ_while(N, M, moving),
    seg_succ(M, S1),
    % ship is stopped at harbor H1 during S1
    stopped_at_harbor(S1, S1).

```

```
32 moving(S) :- not(stopped(S)).
33 stopped(S) :- speed(S, 0).
34 stopped_at_harbor(S, H) :-
35     stopped(S),
36     % fetch location of segment
37     S = segment(_,L,_,_,_,_,_,_),  % trajectory data
38     call(S),
39     % find nearest place within margin
40     nearest(L, H), % calls spatial index
41     rdf(H, geo:featureCode, geo:'H.HBR'),
42     distance(L,H,D),
43     D < 0.175. % scope of being in a harbor
44
45 % trajectory data
46 % segments are successors if begin and end time matches
47 seg_succ(segment(_,_,_,_,_,_,_,T),
48          segment(_,_,_,_,_,_,T,_)).
49
50 % S1 is a valid successor if Pred holds over it
51 seg_succ_if(S0, S1, Pred) :-
52     seg_succ(S0, S1),
53     % construct condition and test it
54     Goal =.. [ Pred, S1 ],
55     Goal.
56
57 % tail recursive definition of transitive segment
58 % successor while condition Pred holds
59 seg_succ_while(S0, SN, Pred) :-
60     seg_succ_if(S0, SN, Pred).
61 seg_succ_while(S0, SN, Pred) :-
62     seg_succ_if(S0, S1, Pred),
63     seg_succ_while(S1, SN, Pred).
64
65 % collect all segments between segment From and To
66 seg_from_to(To, To, [To]).
67 seg_from_to(From, To, [From|ToList]) :-
68     seg_succ(From, Next),
69     seg_from_to(Next, To, ToList).
```

Figure 5: SWI-Prolog rules that link domain-level data to place and behavior semantics. The rules in this example are used to classify ferry behavior.

4.2 Linking to Ontologies

To classify the places at which events happen we use GeoNames Features. We relate the anonymous places (see line number 7 to 12 in figure 4) indicated with latitude and longitude to the typed places in GeoNames by geographical proximity reasoning with the Haversine function.

$$d = R \cdot 2 \arctan^2(\sqrt{a}, \sqrt{1-a})$$

$$a = \sin^2(\delta\mathrm{lat}/2) + \cos(\mathrm{lat}_1) \cdot \cos(\mathrm{lat}_2) \cdot \sin^2(\delta\mathrm{long}/2)$$

where R = the earth's radius, δlat is the difference in latitude and δlong is the difference in longitude. Using a spatial index in Prolog based on an R*-tree implementation from the spatialindex package[21], we can efficiently derive whether a ship is lying still in a harbor, perhaps moored, or at an offshore anchorage or just somewhere out at sea. Ship information, like the callsign, flag, owner, etc. are fetched from various websites[22] and automatically converted to corresponding RDF datatype properties of the ships (Actors in SEM). Ship types mentioned in the MMSI[23] number of the AIS messages[24] are represented as instances of ActorType. This is illustrated in figure 4 on line 20.

4.3 Defining SEM Events

Complex events are derived from the simple events that correspond to the Piecewise Linear Segmentation results by executing Prolog deduction rules. In figure 5 we show the rules involved in classifying segments belonging to ferry trips. The low-level segments are accessed in line 38, 49, and 50. The temporal order of the segments is established by defining a successor relation in line 49. Simple behavior rules for ships sitting still and moving are defined on line 33, 34, and 35. Complex behavior rules for ship trips and the more specific ferry trip, which goes back and forth between two harbors are defined respectively on line 23 and 19. The complex trip behavior rule uses the phenomenon "ongoing behavior" to define that a ship does not stop during a trip (otherwise it would be two trips). This ongoing behavior is defined with a conditional successor relation, which is defined by the clauses on line 54, 62, and 69. The actual classification of the ferry trip happens in the clause defined on line 2. This clause adds the RDF triples that create the new complex event for the ferry trip.

5. CONCLUSION AND FUTURE WORK

We learn event instances from raw data: AIS messages transmitting information about ships' navigation parameters. To recognize simple behavior events from these sensor data, we use a compression, Piecewise Linear Segmentation. This decreases the number of atomic events we have to deal with roughly by a factor 25. We represent the different facets of behavior events, *when* (TimeStampedEntity) did *who* (Actor) do *what* (Event), *where* (Place), to *whom* (Roles of Actors), in the Simple Event Model. We combine spatial reasoning, semantic web reasoning and rules in SWI-Prolog to create new, higher-level, events on top of the recognized events.

In the future SEM will be used as a basic schema for the Semantic Search Engine ClioPatria[25] [15]. We would like to extend the spatial indexing Prolog package to deal with common complex shapes, like polygons and linestrings, and time-parametrized shapes. This would allow us to write efficient rules about the relative position of moving ships. At the moment this is not possible, as we can only index static objects, like harbors. We would like to extend the web information extraction toolkit we use to find ship information like callsign, flag, and draught to extend the range of queries we can formulate about ships. A future challenge is to move from only using existing place features like harbors to also using automatically discovered features, like unofficial ship lanes or queues for tankers in front of a harbor.

6. ACKNOWLEDGMENTS

This work has been carried out as a part of the Poseidon project in collaboration with Thales Nederland, under the responsibilities of the Embedded Systems Institute (ESI). This project is partially supported by the Dutch Ministry of Economic Affairs under the BSIK program.

[21]`http://trac.gispython.org/spatialindex/`
[22]*e.g.* `http://www.vesseltracker.com/`
[23]see Appendix 43 of the International Telecommunications Union Radio Regulations
[24]`http://www.uais.org/`
[25]`http://e-culture.multimedian.nl/software/ClioPatria.shtml`

7. REFERENCES

[1] R. Arndt, R. Troncy, S. Staab, L. Hardman, and M. Vacura. COMM: designing a well-founded multimedia ontology for the web. In *The Semantic Web: ISWC 2007 + ASWC 2007*, volume 4825 of *Lecture Notes in Computer Science*, pages 30–43, Berlin, Heidelberg, 2008. Springer.

[2] N. Crofts, M. Doerr, T. Gill, S. Stead, and M. S. (editors). Definition of the cidoc conceptual reference model. Technical report, 2009.

[3] D. Douglas and T. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*, 10(2):112–122, 1973.

[4] J. Hunter. Combining the cidoc crm and mpeg-7 to describe multimedia in museums. In *Museums and the Web international conference*, Boston, April 2002.

[5] E. J. Keogh, S. Chu, D. Hart, and M. J. Pazzani. An online algorithm for segmenting time series. In N. Cercone, T. Y. Lin, and X. Wu, editors, *ICDM*, pages 289–296. IEEE Computer Society, 2001.

[6] J. M. Martínez, R. Koenen, and F. Pereira. Mpeg-7: the generic multimedia content description standard. *IEEE Computer Society*, 9(Issue 2):78 – 87, April-June 2002.

[7] C. Masolo, S. Borgo, A. Gangemi, N. Guarino, A. Oltramari, and L. Schneider. The wonderweb library of foundational ontologies and the dolce ontology. Technical report, WonderWeb, 2002.

[8] C. Matuszek, J. Cabral, M. Witbrock, and J. DeOliveira. An introduction to the syntax and content of cyc. In *Proceedings of the 2006 AAAI Spring Symposium on Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering*, 2006.

[9] A. Pease, I. Niles, and J. Li. The suggested upper merged ontology: A large ontology for the semantic web and its applications. In *In Working Notes of the AAAI-2002 Workshop on Ontologies and the Semantic Web*, 2002.

[10] Y. Raimond and S. Abdallah. The event ontology, owl-dl ontology, 2006. . Online, 2006. Available: http://purl.org/NET/c4dm/event.owl.

[11] U. Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*, 1(2):244–256, 1972.

[12] T. Ruotsalo and E. Hyvönen. An event-based approach for semantic metadata interoperability. In *6th International and 2nd Asian Semantic Web Conference (ISWC2007, ASWC2007)*, pages 407–420, November 2007.

[13] A. W. M. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain. Content-based image retrieval at the end of the early years. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(12):1349–1380, December 2000.

[14] C. Tsinaraki, P. Polydoros, F. Kazasis, and S. Christodoulakis. Ontology-based semantic indexing for mpeg-7 and tv-anytime audiovisual content. *Multimedia Tools Appl.*, 26(3):299–325, 2005.

[15] J. Wielemaker, M. Hildebrand, J. van Ossenbruggen, and G. Schreiber. Thesaurus-based search in large heterogeneous collections. In A. P. Sheth, S. Staab, M. Dean, M. Paolucci, D. Maynard, T. W. Finin, and K. Thirunarayan, editors, *International Semantic Web Conference*, volume 5318 of *Lecture Notes in Computer Science*, pages 695–708. Springer, 2008.

[16] J. Wielemaker, Z. Huang, and L. van der Meij. Swi-prolog and the web. *Theory and Practice of Logic Programming*, 8(3):363–392, 2008.