Noname manuscript No.
(will be inserted by the editor)

Abstracting and Reasoning over Ship Trajectories and Web Data with the Simple Event Model (SEM)

Willem Robert van Hage · Véronique Malaisé · Gerben de Vries · Guus Schreiber · Maarten van Someren

Received: date / Accepted: date

Abstract Bridging the gap between low-level features and semantics is a problem commonly acknowledged in the Multimedia community. Event modeling can fill this gap by representing knowledge about the data at different level of abstraction. In this paper we present the Simple Event Model (SEM) and its application in a Maritime Safety and Security use case about Situational Awareness, where the data also come as low-level features (of ship trajectories). We show how we abstract over these low-level features, recognize simple behavior events using a Piecewise Linear Segmentation algorithm, and model the resulting events as instances of SEM. We aggregate web data from different sources, 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 comes from the Dutch Poseidon project.

W. R. van Hage
Web & Media Group
VU University Amsterdam
E-mail: wrvhage@few.vu.nl

V. Malaisé
Web & Media Group
VU University Amsterdam
E-mail: vmalaise@few.vu.nl

G. K. D. de Vries
TCS Group
University of Amsterdam
E-mail: G.K.D.deVries@uva.nl

A. Th. Schreiber
Web & Media Group
VU University Amsterdam
E-mail: schreiber@cs.vu.nl

M. W. van Someren
TCS Group
University of Amsterdam
E-mail: M.W.vanSomeren@uva.nl

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

1 Introduction

The notion of "bridging the gap" [20] 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, sensory data reporting ship movement) and semantics. Event modeling can fill the gap, *cf.* [8]. In this paper we show how the Simple Event Model (SEM) can be used as a semantic layer over abstractions derived from domain-level raw data. Data processing techniques can yield knowledge about the world at different levels of abstraction. For example, in the field of moving object analysis, there are machine learning techniques for recognizing flocking based on GPS data, and there are approaches for discovering the goal of a trip, like going back and forth to the office or the supermarket. The results of the latter technique give higher-level knowledge about the world than those of the former. With respect to multimedia applications Westermann and Jain state (in [24]) that *"Basing the representation of events in multimedia applications on a common model makes it easier to create homogeneous event views based on the same model that syndicate events from different applications. Thus, a common multimedia event model promotes the integration of applications. It also facilitates homogeneous access to and interlinking of events from different applications, thereby potentially giving users insights that they couldn't obtain from one application alone."* Although the application discussed in this paper comes from a different domain, the main goal of SEM is the same: to facilitate the integration of knowledge at different levels of abstraction. Problems that come with the integration of knowledge obtained from different methods are heterogeneity and incompleteness. SEM was designed to be robust against missing and duplicate information. We demonstrate the use of SEM to reason over ship behavior at various levels of abstraction integrating knowledge from the web. This use case is particularly interesting, because it shows how track data is not enough for a human system operator to get a good understanding of the maritime situation. This can only be achieved by combining the tracks with external knowledge.

We get ship movement tracks from 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 movement data, and to model them as instances of SEM. We write rules in SWI-Prolog [26] that determine the semantics of these movement events, and integrate them with GeoNames[2] concepts. This determination process follows a layered approach. First we recognize simple movement events like stopping and moving, then we build on these events to define more complex movement event patterns like trips (series of consecutive movements). These are then combined with knowledge about the surroundings and the ships to yield semantically richer events like anchoring (a stop at an anchorage), harbor approaches (movements that end in a stop at a harbor), and ferry trips (repetitive trips between the same two harbors).

The rest of this paper is organized as follows. We present SEM itself in section 2, and its relation to existing event models in section 3. We continue with the description of our

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

[2] `http://www.geonames.org/`

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 SEM: Simple Event Model

SEM was designed to represent events in the broad sense of the word, derived from various sources (from the web, sensory data, historical documents, etc.). These data can be incomplete (*e.g.* missing values) and partial (*e.g.* missing entire facets), and they follow different design decisions. SEM has to be very flexible to cope with these issues. As SEM is meant to represent data from uncontrollable sources, the notions of temporary validity (during what temporal interval an event or a statement holds), roles (the kind of participation in an event) and authority (according to whom an event or statement holds) become important. It is also important to allow all classes and properties in the model to be optional and duplicable, and to be flexible towards different ways of modeling time, place, role, and type. In the rest of this section we describe how these requirements are implemented in SEM. First we discuss the classes and properties that make up SEM; then how to model views, roles and temporary validity as constraints on properties, the notion of authority; how to model time and space with symbols (*c.q.* URIs) or values (*c.q.* coordinates). We illustrate these with a simple example of how a maritime event can be modeled in SEM, represented in figure 5.

Classes SEM's classes are divided in three groups: Core classes, Types, and Constraints. This is illustrated in Figure 1. There are four core classes: sem:Event (what happens), sem:Actor (who or what participated), sem:Place (where), sem:Time (when). Each core class has an associated sem:Type class, which contains resources that indicate the type of a core individual. Individuals and their types are usually borrowed from other vocabularies. For example, the sem:Place "Harwich" (geo:7116094) from our example (see figures 5 and 6) and its sem:PlaceType "Harbor" (geo:H.HBR) are borrowed from the geographic ontology GeoNames[3]. Alternatively, the types could also be borrowed from the LSCOM[4] ontology.

The sem:Type classes exist to aggregate the various implementations of type systems in any vocabulary. Some vocabularies do not have properties that exactly correspond to the sem:type property, even though a type can be derived from the value of other properties. This can be done by using Alan Rector's Value Sets and Value Partition patterns.[5] These design patterns are illustrated in figure 2. Having explicit sem:Type classes provides a placeholder to define these patterns. If you want to make the class of all harbors using GeoNames' geo:featureCode property you could do this in the following two ways. You could define geo:featureCode to be a subproperty of sem:placeType. This makes geo:H.HBR a class, containing all geo:Features that are a harbor. If you do not want to turn the individual geo:H.HBR into a class you can follow the value sets pattern and define the set of harbors to be a subclass of sem:Place and an owl:Restriction on the geo:featureCode property with owl:hasValue geo:H.HBR. This approach keeps geo:H.HBR an individual.

[3] http://www.geonames.org/

[4] http://www.lscom.org/ontology/

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

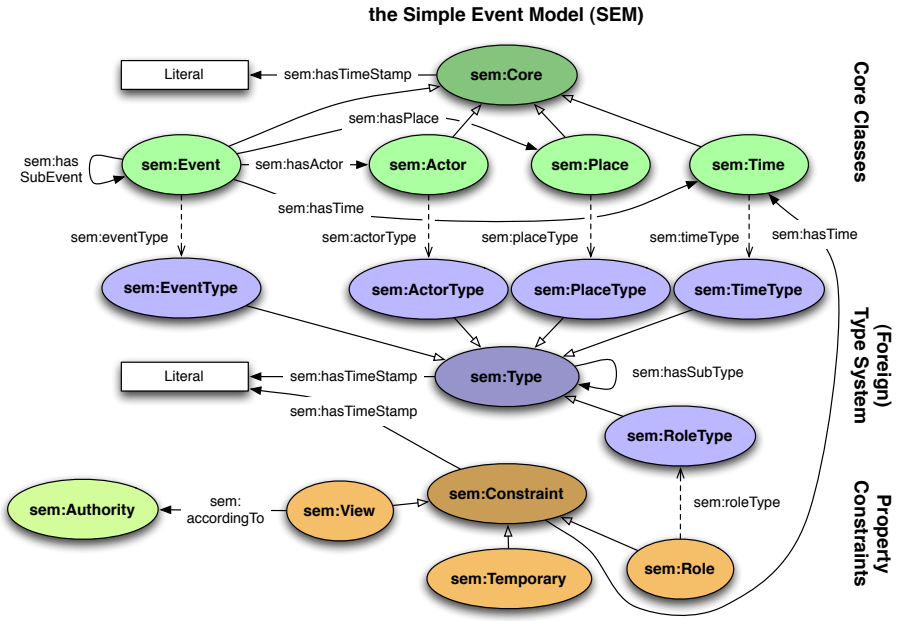

Fig. 1 The classes of the Simple Event Model. Arrows with open arrow heads symbolize rdfs:subClassOf properties. Dashed arrows symbolize subproperties of rdf:type; regular arrows represent other properties.

Besides the sem:Actor class, a class sem:Object has been defined as a rdfs:subClassOf sem:Actor, for the cases where it is necessary to specify a distinction between these two concepts. For example, a container loaded on a container ship is a simple object that does not participate in a trip, but might be interesting to mention in the context of the event. If there would be an event in which the container falls overboard then it would be a sem:Actor even though, like the ship, it is an inanimate object.

The class sem:Authority is used to indicate according to whom a statement is valid. Individuals of sem:Authority can be, but are not necessarily sem:Actors. They can also symbolize data sources, such as the URN of a web services. The sem:Authority class is meant as a hook for provenance and trust reasoning, even though SEM itself does not explicitly provides these. Additional trust reasoning, like evidential reasoning [4], can be superimposed on SEM.

The class sem:Place defines a symbolic place, which does not need to have a location indicated by coordinates per se, but which can be given a geolocation. This way SEM can represent both concrete and symbolic places (*e.g.* "sandy desert"). In our use case, the location of events is attached to the segment using properties from the W3C WGS84 vocabulary[6]. This is illustrated in figure 7 on line 11.

The class sem:Time defines a symbolic time, analogous to the symbolic places described above, which values can be taken from the W3C's Time ontology[7] amongst other time ontologies. It is also possible to define time as a simple (set of) data value(s)

[6] http://www.w3.org/2003/01/geo/
[7] http://www.w3.org/TR/owl-time/

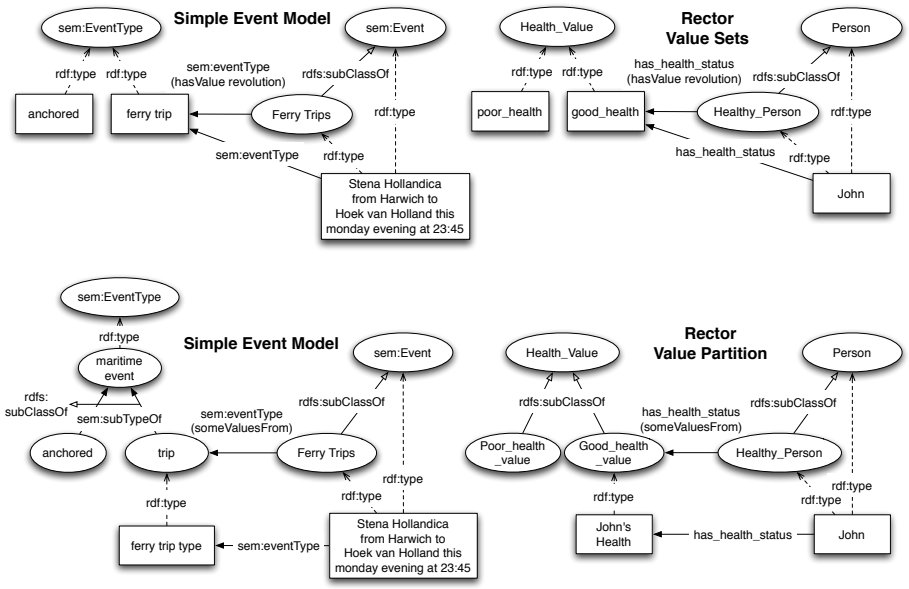

Fig. 2 Alan Rector's value sets (top) and value partition (bottom) patterns applied to SEM (left) compared to the original examples from the W3C working group note (right).

in SEM, see the presentation of the sem:hasTimeStamp properties below. In our use case, time is represented as data values in ISO 8601 as a RDF Literal or TIMEX format[8] as a RDF Literal of type rdf:XMLLiteral attached to the segment with sem:hasBeginTimeStamp and sem:hasEndTimeStamp, both subproperties of sem:hasTimeStamp. This is illustrated in figure 7 on line 13 to 16.

Properties SEM's properties are divided in three kinds: sem:eventProperties, sem:type properties and a few miscellaneous properties like sem:accordingTo and sem:hasTimeStamp's subproperties, see figure 3. The sem:eventProperties relate sem:Events to other individuals. A sem:type relates individuals of the sem:Core class to individuals of sem:-Type. There are specific subproperties of sem:type for each of the core classes, for example sem:eventType, to facilitate querying. They reduce the strain on reasoners, because sem:eventType subproperty already tells you that it points to an individual of sem:EventType, hence this does not have to be derived by subsumption reasoning. sem:accordingTo relates a sem:View to a sem:Authority and is used to represent opinions. There are seven sem:hasTimeStamp properties. One for single time values, sem:hasTimeStamp; two for time intervals, sem:hasBeginTimeStamp and sem:hasEndTimeStamp; and four for uncertain time intervals, sem:hasEarliestBeginTimeStamp, sem:hasLatestBeginTimeStamp, sem:hasEarliestEndTimeStamp, and sem:hasLatestEndTimeStamp. The latter kind of intervals is used to describe any kind of uncertainty about the begin or end of a period. It does not imply, for example, a fuzzy interpretation of time. Open-ended intervals can be expressed by omitting begin or end timestamps.

[8] `http://fofoca.mitre.org/`

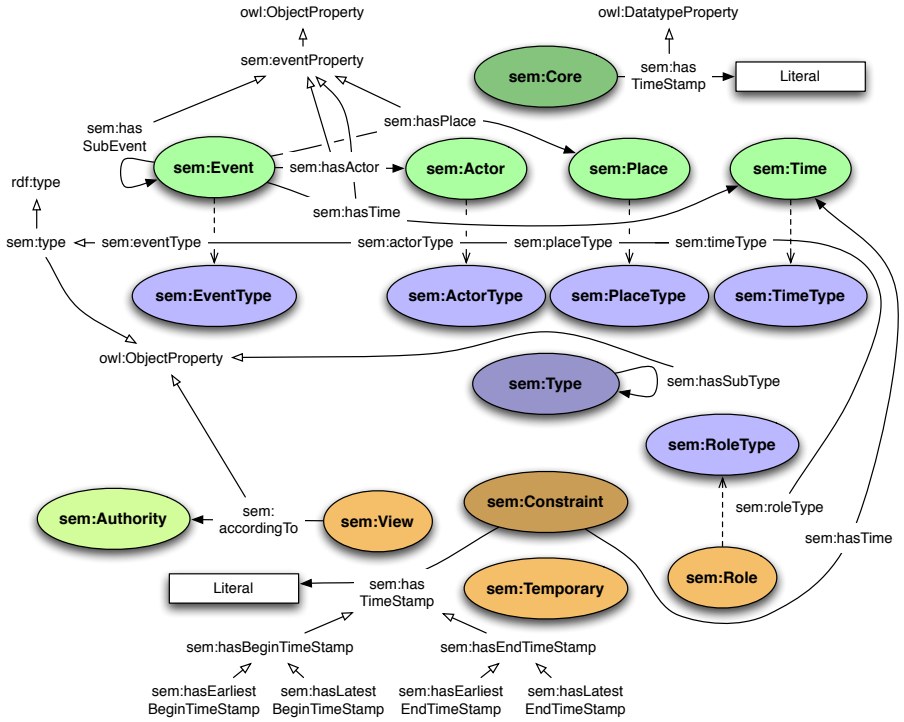

Fig. 3 The properties of the Simple Event Model. Arrows with open arrow heads symbolize rdfs:subPropertyOf properties. Dashed arrows symbolize subproperties of rdf:type; regular arrows represent other properties.

There are two aggregation relations amongst the sem:eventProperty and sem:type properties: sem:hasSubEvent (see the example in figure 6) and sem:hasSubType. These can be used to indicate that respectively a sem:Event or sem:Type is related to another more generic sem:Event or sem:Type, without any further commitments. For example, poseidon:anchored has sem:subTypeOf poseidon:stopped; and the sem:Event instance ex:wimbledon_2010_mahut_isner_game_183 sem:subEventOf ex:wimbledon_2010_-first-round_match_mahut_isner. More specific relations between events and between types are not part of SEM and should be taken from other ontologies, like GEM [27].

Constraints Property constraints can be applied to any property. They constrain the validity of the property and are expressed as either a reification of the property or by adding attributes to the property and turning it into an n-ary relation. There are three permissible ways to represent sem:Constraints: as a named graph, as a reification (with rdf:Statement, see `http://www.w3.org/TR/rdf-schema/#ch_statement`) and with an rdf:value pattern (see `http://www.w3.org/TR/2004/REC-rdf-primer-20040210/#example16`). The default representation is the rdf:value pattern, which is often used when representing the unit of measure of a value.[9]

[9] *cf.* the MUO ontology `https://forge.morfeo-project.org/wiki_en/index.php/How_to_use_MUO`

There are three kinds of sem:Constraints: sem:Role, sem:Temporary and sem:View. sem:Role defines the role that an individual of a class is playing in the context of a specific event (*i.e.* to which it is linked with a sem:eventProperty). Roles can be specified for all sem:Core individuals, for example, Actors ("pusher" in the case of a Tugboat, or "anchorman" in the case of a news item) as well as places ("destination"). sem:Roles are not meant to model roles in the sense of temporary or dependent types, like "mother". Instead, sem:Role explicitly models the event-bounded role. For example, a maritime pilot is guiding ships through dangerous or congested waters, such as harbors or river mouths. In the case of an event "Ship arriving in a harbor", the maritime pilot has the role "guide", which is bounded to the sem:Event. sem:Temporary defines the temporal boundary within which a property holds. For example, the flag or name of a ship can change during its existence, independently from any event. sem:View defines points of view, opinions: in the case of a collision, the description of the event might well depend on the person who reports it. This can be modeled as a sem:View constraint on the property sem:roleType, for example, if the responsibility (the actor's role) is contested. A view holds sem:accordingTo a sem:Authority. Another example of the use of sem:View is to distinguish the sources of two conflicting ship positions for a ship at a given time. Multiple kinds of sem:Constraints can be used in combination to create conjunctive statements.

Availability and Extension SEM is available online at the URL: `http://semanticweb.cs.vu.nl/2009/11/sem/`. It is mapped to a set of event models: Event Ontology [15]; CultureSampo [17]; Dublin Core[10]; CIDOC-CRM [6], and of commonly used upper level ontologies: DOLCE [5]; SUMO[11]; and CYC[12]. This set of mappings has been modeled in SKOS[13].

3 Related Work

With respect to the semantic analysis of moving objects, comparable work has been done by [14]. Their work mainly focuses on describing collective behavior in OWL, we focus more on developing a framework for integrating external knowledge sources. Also, we choose to use all of Prolog as our reasoning tool as opposed to an OWL reasoner.

With respect to event models, different other models have been proposed to bridge the gap between domain-level features and the *semantic* level. For example, the MPEG-7 [12] Multimedia Description Scheme contains the two aspects. The model is complex, though, 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 [10,22]. COMM [2] allows combination of descriptions from MPEG-7 with a semantic description. In [2], they take as example DOLCE [13] and its extension, the Description and Situation pattern[5], to describe the semantics related to the low-level data described. 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 (like a tag) or a complex description, typically event models (as suggested in [10]). We adopted the

[10] `http://dublincore.org/`
[11] `http://www.ontologyportal.org/`
[12] `http://www.cyc.com/cyc/opencyc/`
[13] `http://www.w3.org/2004/02/skos/`

same idea and designed SEM in the purpose of associating different levels of semantics to abstractions over low-level data. The event models that had the greatest influence on the development of SEM are: EO [15], CIDOC-CRM [6], LODE [19], and E [23].

Event models can be described through different characteristics: concept-based ([10,6]) vs property based modeling ([15,19,17]); size (minimal number of classes and properties like EO versus large ontology of CIDOC-CRM); level of axiomatization (lightweight like EO versus more constrained model like LODE). SEM defines a set of classes *and* properties to represent and reason about events, standing between concept-based and property-based models. SEM is also average with respect to size, and it does not import any restrictive semantics from other models. In particular, the links to other models and ontologies are done with SKOS mappings in order to avoid inheriting constraints from these external resources.

We present here a more detailed overview of the relationships between SEM and three RDF based event models. These models were selected as representatives of the aforementioned overlapping categories: EO as a concept-based lightweight small ontology, LODE as a lightweight property based ontology with some restrictions and CIDOC-CRM as a large concept-based ontology with no formal restriction. We discuss these models on basis of how they model (or not) the notions of Role, Type, View and Temporary. These notions go beyond the most common components (event, participant, time and place) and are part of our requirements.

3.1 EO

In the context of musical performances Queen Mary University of London developed the Event Ontology[14] (EO) [15]. EO has a very simple design. It consists of four classes (eo:Event and three implicit classes which are the ranges of EO properties: Agent, Factor and Product) and seventeen properties. EO defines a minimal event, and relies on external vocabularies to refine the knowledge expressed. For example, no Agent class is defined per se, but their eo:agent property has foaf:Agent as a range: EO benefits therefore from the richness of the FOAF vocabulary.[15] Roles, Types, Views and Temporary are not defined in EO. Place, Time and Agent are defined via range restrictions on EO's properties. The explicit linking to vocabularies brings EO its richness, but also constrains the possible values for these properties. SEM is compatible with more Place, Time and Actor representations, as we decided not to have such restrictions. The main common point between SEM and EO is the modularity in the design: most classes are optional in EO; In SEM, even the sem:Event class is optional. This allows the representation of actors without events in which they participate. This is useful when the different parts of the event are gathered from different sources, as in our use case.

3.2 LODE

LODE [19] also aims at a minimal modeling of events. It contains one class (Event) and six properties: lode:atTime, lode:circa, lode:inSpace, lode:atPlace, lode:involved and

[14] http://motools.sourceforge.net/event/event.html
[15] http://www.foaf-project.org/

lode:involvedAgent. Both the class and the properties are formally mapped to other event models like the CIDOC-CRM, EO and DOLCE's DUL version, by the use of owl:sameAs and rdfs:subPropertyOf. In this way, interoperability is enabled and a user can benefit from existing more complex vocabularies, while LODE itself keeps its own classes and properties at the lowest possible number. Role, Type and View can be expressed via their mapping to DUL, by using the Description and Situation patterns, or via the interpretation and mereology patterns of F [18].[16] In SEM, we also adopt the principle of using external vocabularies for modeling properties that are beyond the model's scope, like the causality. But to the difference with LODE, we do not make formal mappings, functional property restrictions and do not conform to one single vocabulary for our properties. We do not benefit from the other models or vocabularies directly, but stay open to more diversity. The other vocabularies can be connected to SEM via our placeholders for Role and Type. Time is expressed using the OWL Time ontology[17], in which temporal entities are represented instances, as opposed to data values. This complicates the representation of time unnecessarily for our use case. Another reason why we did not use LODE for this work is that, like EO, LODE does not have explicit Actor and Place classes.

3.3 CIDOC-CRM

CIDOC-CRM [6] was created for describing museum artifacts, in the goal of enhancing their exchange across musea. The whole model is quite large: it contains 140 classes and 144 properties. A subset of these can be used to represent events. Roles are represented in the same fashion as in SEM: as constraints on a property. But unlike SEM, the Role can only be assigned to the Actor. Types can apply to all entities of CIDOC-CRM, but time-stamps (modeled with a two-position pattern) can only apply to TemporalEntities. Roles, Types and other event constituants cannot be time-stamped. We generalize the CIDOC-CRM's model with SEM, and add the representation of View. The reason why we did not use CIDOC-CRM for our use case is that it only allows one type per object. This means that a ship can only have one type, but also that its behavior can only have one type, which is too restrictive for this work.

3.4 Comparison

SEM gathers the elements that give a light-weight description of events, but without importing strong semantic definitions that easily lead to inconsistency, *e.g.* owl:FunctionalProperty, owl:disjointWith. In addition to this SEM specifies the necessary additions for dealing with heterogeneous and messy data from the web, *i.e.* foreign types, constraints, and authority.

4 Use Case: Maritime Situational Awareness

We describe a Semantic Web application in which we automatically recognize events from domain-level data representing ship trajectories. From these atomic events, mod-

[16] F specializes D&S patterns from DUL.

[17] `http://www.w3.org/2006/time`

eled as SEM instances, we derive ship behavior types (slowing down, speeding up, anchored) to reason about patterns, *e.g.* ship maneuvering when approaching an anchorage or a ferry trip. The reasoning involves various type of knowledge, which we fetch from various sources. We describe these sources in section 4.1. We transform the ship trajectories into segments of consistent movement using a piecewise linear segmentation compression (PLS) algorithm. This gives us our low-level SEM event instances. The PLS algorithm is described in section 4.2. We describe the conversion to SEM in section 4.3 and the matching of ships and places to external resources that describe them in section 4.4. The architecture of the system is shown in figure 4. In section 4.5 we show how we make abstractions over and reason about the data sources that we gather. We define rules building on these basic blocks. These rules yield more complex SEM events, like trips. Then we add relevant maritime geographical features from GeoNames and define further rules over these two sources of knowledge, that determine higher-level events, like ferry trips and anchoring.

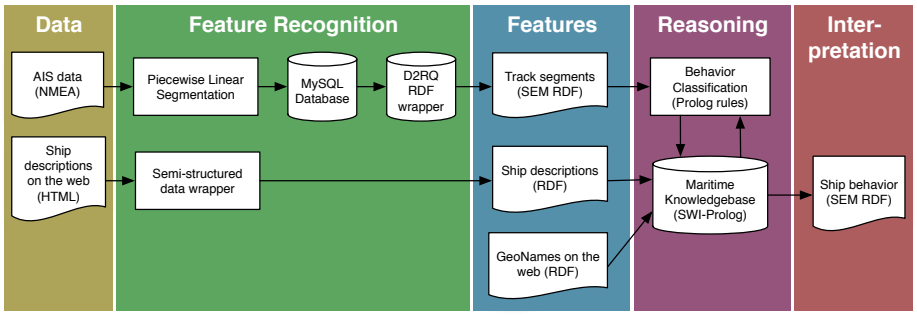

Fig. 4 Data flow diagram of the entire ship behavior recognition system.

4.1 Data Sources

The main data source for ship trajectory data in our application comes from the Automatic Identification System (AIS)[18]. Each commercial vessel over 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 use GeoNames as ontology of geographic data. We extended GeoNames with 64 harbors and anchorages and corrected the position of 36 existing harbors. GeoNames is created collaboratively with a wiki where anybody can add and change features. The RDF version of GeoNames is periodically generated automatically from the wiki data.

Ship information, like the callsign, flag, and owner, are fetched from various websites: `http://www.marinetraffic.com/`, `http://www.vesseltracker.com/`, `http://www.havenais.com/`, and `http://www.xvas.it/`. We use Marinetraffic.com as a baseline and extend it with information from the other websites. During the course of the project Xvas.it restricted its access policy. The information about ship types derived from these sources

[18] `http://www.uais.org/`

is converted to our own small internal actor type vocabulary, which is aligned to WordNet[19] 2.0 with SKOS properties.

4.2 From AIS Data to Segments of Consistent Movement

In this section we briefly describe a method to automatically convert "raw" movement data in the form of trajectories into SEM events that we call *segments*. This method is based on a piecewise linear segmentation compression technique for trajectories. The compression of single AIS messages into segments decreases the total number of atomic events we have to deal with roughly by a factor 25, which makes further processing significantly faster. We detail this technique and describe how we use it to create segment SEM events. These segments contain the parameters that Andrienko and Andrienko [1] identify as the basic data for describing movements: the entity (via an identifier), the (geo)-coordinates where the event starts and stops and the time when the start and stop occurs. Furthermore, these segments can easily be classified as *stop* or *move*. These concepts where recently identified [21] as the first step in giving semantics to moving object trajectories.

Trajectories We mentioned that the ship trajectory data in our application comes from the Automatic Identification System (AIS). Now, let us define a trajectory more formally as: $T = \{(x_1, y_1, v_1, c_1, t_1), \ldots, (x_n, y_n, v_n, c_n, t_n)\}$, where x and y are the coordinates[20], v the speed, and c the course at time t. As useful shorthands we also define: $T(i) = (x_i, y_i, v_i, c_i, t_i)$ and $T(i, j) = \{(x_i, y_i, v_i, c_i, t_i), \ldots, (x_j, y_j, v_j, c_j, t_j)\}$, furthermore, $T'((x_i, y_i, v_i, c_i, t_i)) = i$.

As the trajectories 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 trajectories suggests that they can be compressed quite well using piecewise linear representation techniques. The idea behind using a piecewise linear representation method is that this technique segments a trajectory into pieces which have more or less constant movement. These segments of constant behavior are the lowest level SEM events and the building blocks for more complex events.

We use a two-step variant of piecewise linear segmentation, described in algorithms 1 and 2. This two-step version, which first looks at the speed of a moving object (algorithm 1) and then at the location (algorithm 2), is better at preserving the concepts of *stop* and *move* that we mentioned above[21].

First, we consider the standard piecewise linear segmentation algorithm given in 1 which is used twice in our two-step variant. This algorithm goes by many names [11]. It is best known as the Douglas-Peucker algorithm [7] in carthography and Ramer's algorithm [16] in image processing. The algorithm recursively compresses a line, or in

[19] `http://www.w3.org/TR/wordnet-rdf/`

[20] Usually these are latitude and longitude, which, because of the shape of the earth, do not allow for easy geometrical computations. However we assume here that we can do this, *e.g.* because they are adequately projected.

[21] We will explore this issue more in a future paper.

our case a trajectory T, defined as a list of points, into linear segments. The start and end point of the line or trajectory are selected and for each point in between, the error with respect to the linear interpolation between the start and end point is computed. The point with the maximum error, higher than a fixed threshold ϵ is kept and the recursion continues with that point as a new start and end point. Recursion stops when there is no point with an error higher than ϵ.

Algorithm 1 pls(T, ϵ)

```
1  We use end to indicate the index of the last element of a trajectory.
2  d_max = 0
3  i_max = 0
4  for i = 2 to end − 1 do
5     d = E(T(i), {T(1), T(end)})
6     if d > d_max then
7        i_max = i
8        d_max = d
9     end
10 end
11 if d_max ≥ ε then
12    A = pls(T(1, i_max), ε)
13    B = pls(T(i_max, end), ε)
14    T^C = {A, B(2, end)}
15 else
16    T^C = {T(1), T(end)}
17 end
18 return T^C
```

There are a number of options for the error function (algorithm 1, line 5) that piecewise linear segmentation can use, especially when considering trajectories (*cf.* [3, 9]). We only use two. The first one is simple two dimensional euclidean distance, defined for our trajectories as:

$$\mathbf{E}_2((x_i, y_i, v_i, c_i, t_i), \{(x_1, y_1, v_1, c_1, t_1), (x_n, y_n, v_n, c_n, t_n)\}) = \sqrt{(x_i - x_i')^2 + (y_i - y_i')^2},$$

where (x_i', y_i') is the closest point on the line-segment $\{(x_1, y_1), (x_n, y_n)\}$. (1)

The second one is defined on the speed attribute. Here we compare the speed at a certain time t_i to the speed that we would get if we linearly interpolate between t_1 and t_n at the same t_i.

$$\mathbf{E}_v((x_i, y_i, v_i, c_i, t_i), \{(x_1, y_1, v_1, c_1, t_1), (x_n, y_n, v_n, c_n, t_n)\} = \|v_i - v_i'\|$$

where v_i' is the point on the line-segment $\{(v_1, t_1), (v_n, t_n)\}$ with time t_i. (2)

In our two-step variant of piecewise linear segmentation, given in algorithm 2, we apply algorithm 1 to a trajectory in two steps. First we only compress based on the speed (v) of the trajectory (line 1 of algorithm 2). In this case we use the error function $\mathbf{E}_v$. Then we apply compression to each segment created in the first compression step (line 6 of algorithm 2), but we look at location[22], which only takes into account x and y. Here we use the error function $\mathbf{E}_2$.

[22] This is the traditional Douglas-Peucker algorithm.

Algorithm 2 2step-pls(T,ϵ_v,ϵ_p)

```
1  A = pls(T, ε_v)
2  T^C = ∅
3  for i = 1 to ||V^C|| − 1 do
4     m = T'(A(i))
5     n = T'(A(i + 1))
6     B = pls(T(m, n), ε_p)
7     T^C = T^C ∪ B
8  end
9  return T^C
```

Storing the Segments The result of the two-step piecewise linear segmentation, described above, is stored in an SQL-database in the table *segments*, see figure 4. A segment describes a constant piece of movement. Let T be a trajectory as defined earlier, then T^C is its compressed variant: $T^C = \mathbf{2step\text{-}pls}(T, \epsilon_v, \epsilon_p)$.

Now, we insert the following tuples into the segments table:

$$\langle \mathtt{uri}, x_i, y_i, x_{i+1}, y_{i+1}, v_i, v_{i+1}, c_i, c_{i+1}, t_i, t_{i+1} \rangle$$
$$\text{for all } i \text{ such that } T^C(i) = (x_i, y_i, v_i, c_i, t_i)$$
$$\text{and } T^C(i+1) = (x_{i+1}, y_{i+1}, v_{i+1}, c_{i+1}, t_{i+1}). \quad (3)$$

For each segment we generate a URI based on the ship's unique identifier, the Maritime Mobile Service Identity (MMSI) number, and the start time (t_i). This URI uniquely identifies the segment. Furthermore, the segment contains a start (x_i, y_i) and end (x_{i+1}, y_{i+1}) position, a start (v_i) and end (v_{i+1}) speed, a start (c_i) and end (c_{i+1}) course, and a start (t_i) and end (t_{i+1}) time.

4.3 From Segments to Semantics

Segments as Events in SEM Every segment in the database is assigned either the two basic movement types *stop* or *move*. These are stored in an additional column in the MySQL database. Stops are determined by means of a threshold[23] on the average speed of the segment. All additional semantics are described outside of the database, in RDF. To make the transition from the database to RDF we use the D2RQ server[24] by the Free University Berlin. This is a database wrapper that provides an RDF graph view over the flat database table we use to store the segments, see figure 4. Each segment (*c.q.* row in the database) corresponds to a single instance of a sem:Event, with an additional sem:eventType poseidon:etype_stopped or poseidon:etype_moving depending on the basic movement type of the segment. Also, each segment describes the state of a single ship, identified by its MMSI number, which corresponds to a single instance of a sem:Actor, which is connected to the segment event by the sem:hasActor property. The ship gets a ais:mmsi property to the value of its MMSI number. Additional properties of the ship that are fetched from the web are added later as properties of the instance representing the ship. The begin place and end place are represented as two instances of sem:Place, which are connected to the event by the seg:hasBeginPlace and

[23] In the order of 0.1 knots.

[24] `http://www4.wiwiss.fu-berlin.de/bizer/d2rq/`

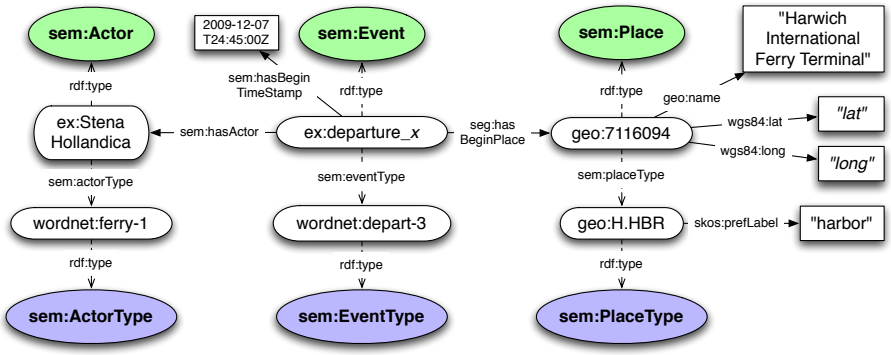

Fig. 5 An example of an event describing the departure of a ferry from Harwich. Segment events initially do not have a meaningful place, only an anonymous place (*c.q.* blank node) with coordinates. The meaningful place shown in this example is attached to the event by conflating it to the anonymous place of the segment by means of spatial proximity reasoning.

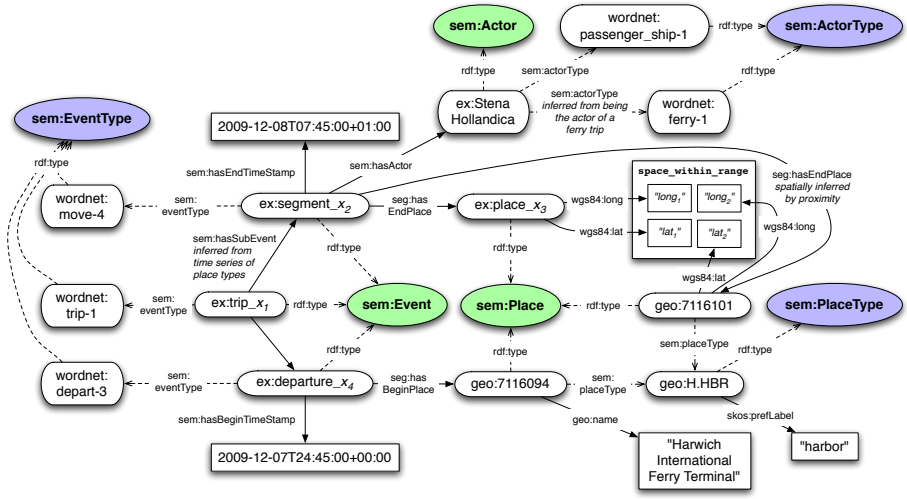

Fig. 6 An example of two subevents of a ferry trip, an arrival (ex:segment_x_2) and a departure (ex:departure_x_4). This example shows the seg:hasEndPlace property instance that is inferred by conflation of places. For the sake of readability not all properties and inferences are shown.

seg:hasEndPlace properties, subproperties of sem:hasPlace. We attach the additional properties like begin and end speed to the event instance by segment-specific properties like seg:hasBeginSpeedOverGround. An example of the RDF generated in this way is shown in figure 7. A simple illustration of the structure of the resulting RDF graph is shown in figure 5 and an elaborate example in figure 6.

4.4 Conflation of Places and Actors

Matching Places To classify the places at which events happen we use GeoNames Features. We relate the location of anonymous places (see line number 7 to 12 in figure 7) indicated with wgs84:lat and wgs84:long in the RDF representation of the segments to the typed places in GeoNames by geographical proximity reasoning using the Haversine distance function:

$$d = R \cdot 2\arctan^2(\sqrt{a}, \sqrt{1-a})$$
$$a = \sin^2(\delta\text{lat}/2) + \cos(\text{lat}_1) \cdot \cos(\text{lat}_2) \cdot \sin^2(\delta\text{long}/2)$$

where R = the earth's radius, δlat is the difference in latitude and δlong is the difference in longitude. Using the SWI-Prolog Space Package, based on an R*-tree implementation from the spatialindex package[25], 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. GeoNames associates instances of places with geo-coordinates to GeoNames feature codes like geo:H.HBR (harbor), and geo:H.ANCH (anchorage). The Space Package derives that the coordinates of a given segments are close to coordinates defined in GeoNames, and further reasoning can then use the associated semantic type to refine the classification of a ship's behavior: a segment typed as poseidon:etype_stopped and for which the place of stop has the type geo:H.HBR, gets the additional sem:eventType poseidon:etype_stopped_in_harbor. The spatial conflation is illustrated in figure 6 and in the code example in figure 8 on line 4–8.

Matching Actors We automatically convert the information about ships described in the various websites mentioned in section 4.1 to RDF properties of the ships (Actors in SEM). Amongst these properties are datatype properties like ais:length and ais:callsign, but also types, like passenger vessel, which we map to our local vocabulary that is aligned to WordNet. In this case, passenger vessel would be translated to poseidon:atype_passenger_vessel, which is aligned to wordnet:synset-passenger_ship-noun-1. This is illustrated in figure 7 on line 20.

4.5 Deriving Complex SEM Events

To derive more complex behavior than the simple poseidon:etype_(stopped|moving) events we defined a set of rules that build on the typed segment event. For example, to derive the complex behavior "trip" we use a rule that is based on the assumption that if we do not know about an explicit stop between consecutive moving events that it does not exist, *i.e.* we temporarily make a closed world assumption. This allows us to deal with missing ship observations (which happens frequently). We conclude that if we do not know about any stop at a harbor between two stops at harbors a and b, that there was a trip between harbor a and b. This is shown in line 14–21 of figure 8. We encode this trip as a new event, which sem:hasSubEvent the segments that compose the trip. This is shown in line 32–50 of figure 9. The harbors of departure and arrival, a and b, become seg:has(Begin|End)Place properties of the new trip event.

When the RDF describing trip events has been added to the knowledgebase we can use it as a new layer on which we can build new rules. For example, we can define

[25] `http://trac.gispython.org/spatialindex/`

a ferry trip as a trip back and forth between two different harbors, see line 23–30 of figure 8. The ferry trips recognized in this way can subsequently be inserted into the knowledgebase as new events, like the trips, but not referring to segments anymore. The trip and the return trip composing the ferry trip become subevents of the event representing the ferry trip. This is described on line 54–66 of figure 9.

An important advantage of storing the intermediate results of all the rules at various layers of abstraction is that it does not matter in which way the RDF representing an event was generated. For example, as long as its subevents exist we can derive ferry trips. This means that some trips could be derived from AIS segments, like discussed before, while others could be derived from another source, like radar, text mining on a ferry schedule on the web, or even manual extension or correction of the knowledgebase.

```
1  poseidon:segment_mmsi_timestamp a sem:Event ;
2      sem:eventType seg:AISsegment ;
3      % low-level behavior semantics
4      sem:eventType poseidon:etype_departing ;
5      % high-level behavior semantics
6      sem:subEventOf poseidon:ferry_trip_mmsi_n ;
7      seg:hasBeginPlace [
8          a sem:Place ;
9          % classified as a harbor due to proximity to
10         % geoi:7116101, see line number 26
11         wgs84:lat "51.9762" ;  wgs84:long "4.1245" ;
12     ] ;
13     sem:hasBeginTimeStamp "<timex2object>
14         <timex2 VAL="2008-08-04T03:00">
15             2008-08-04T03:00+01:00
16         </timex2object>"^^rdf:XMLLiteral ;
17     sem:hasActor poseidon:actor_ship_mmsi .
18
19 poseidon:actor_ship_mmsi a sem:Actor ;
20     sem:actorType poseidon:atype_passenger_vessel ;
21     ais:name "USS Enterprise" ;
22     ... ;
23     ais:mmsi "mmsi" .
24
25 # matched to the segment location by proximity
26 geoi:7116101 a geo:Feature ;
27     geo:name "Berghaven" ;
28     geo:parentFeature geoi:7116101 ;
29     wgs84:lat "51.97697" ;  wgs84:long "4.12401" ;
30     ... ;
31     geo:featureCode geo:H.HBR .
```

Fig. 7 A ship behavior segment modeled in SEM. Line 1–23 illustrates the SEM RDF format of segment events that is provided by the D2RQ database wrapper. Line 26–31 shows a GeoNames Feature that was conflated with the sem:Place of the event. In this case, the ship is at a harbor.

5 Conclusion and Future Work

We learn event instances from raw data: AIS transceivers transmitting information about ship navigation parameters. To recognize simple behavior events from these

```
stopped_at(Seg, Near, FeatureCode) :-
   rdfs_individual_of(Seg, poseidon:etype_stopped), % from the database
   rdf(Seg, sem:hasPlace, Loc),
   space_nearest_bounded(Loc, Near, 0.15),
   rdf(Near, geo:featureCode, FeatureCode),
   (   rdf_equal(geo:'H.HBR', FeatureCode)
   ;   rdf_equal(geo:'H.ANCH', FeatureCode)
   ).

stopped_at_harbor(Seg, Hbr) :-
   rdf_equal(geo:'H.HBR', HarborCode),
   stopped_at(Seg, Hbr, HarborCode).

trip(FromSeg, FromHarbor, ToSeg, ToHarbor) :-
   stopped_at_harbor(FromSeg, FromHarbor),
   % fetches all movement segments of the ship between FromSeg and ToSeg
   % and checks that these are not known to be stops at some harbor
   segments_between(FromSeg, ToSeg, Between),
   forall(member(Seg, Between),
          \+stopped_at_harbor(Seg,_)),
   stopped_at_harbor(ToSeg, ToHarbor).

ferry_trip(Trip, ReturnTrip) :-
   rdfs_individual_of(Trip, poseidon:etype_trip),
   rdf(Trip, seg:hasBeginPlace, HarborA),
   rdf(Trip, seg:hasEndPlace, HarborB),
   rdfs_individual_of(ReturnTrip, poseidon:etype_trip),
   rdf(ReturnTrip, seg:hasBeginPlace, HarborB),
   rdf(ReturnTrip, seg:hasEndPlace, HarborA),
   HarborA \= HarborB.
```

Fig. 8 First part of a code example illustrating how we use SWI-Prolog rules to derive simple (stopped) and complex (ferry trip) event types from low-level segment events in SEM RDF format. The example is continued in figure 9. The rules shown in this figure show how you can define the behavior of "stopping", "stopping at a harbor", "trip", and making a "ferry trip". The actual assertion of the RDF statements that classify the behavior exhibited in segments is shown in figure 9.

sensor data, we use a compression algorithm, Piecewise Linear Segmentation. This decreases the number of atomic events we have to deal with roughly by a factor 25, which greatly improves the processing speed of the rest of our system. We represent the different facets of behavior events, *when* (sem:hasTimeStamp) did *who* (sem:Actor) do *what* (sem:Event), *where* (sem:Place), 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 movement patterns. This allows representation of events at different levels of abstraction. We keep the link between the different layers of semantics, information and data that come from very different applications (machine learning and text mining). We syndicate the output of the applications in a single event representation. Our event model also enables the combination of events with other background knowledge. The integration happens at the knowledge level. Abstraction, syndication and the integration with background knowledge are part of the requirements for a relevant Event Model for Multimedia defined by [24]. The author emphasises one drawback of current models: *"Although event detection on various abstraction levels and for different domains is a central topic in content analysis, the focus has mostly been on the use of content features for detecting events within media*

```
% semantic classification of trip behavior
% (complex event consisting of simple events)
classify_trip_behavior :-
    % find instances of trips with trip/2
    findall(trip(From, FromHarbor, To, ToHarbor),
            trip(From, FromHarbor, To, ToHarbor),
            Trips),
    Trips \= [],
    % define ferry trip event
    forall(member(trip(From, FromHarbor, To, ToHarbor), Trips),
           rdf_bnode(Trip),
           rdf_assert(Trip, rdf:type, poseidon:etype_trip),
           rdf_assert(Trip, seg:hasBeginPlace, FromHarbor),
           rdf_assert(Trip, seg:hasEndPlace, ToHarbor),
           % make intermediate segments part of the trip
           (   segments_between(From , To, Between),
               forall(member(Seg, Between),
                      rdf_assert(Trip, sem:hasSubEvent, Seg))
           )).

% semantic classification of ferry trip behavior
% (complex event consisting of complex events)
classify_ferry_trip_behavior :-
    findall(ferry_trip(Trip, ReturnTrip, HarborA, HarborB),
            (   ferry_trip(Trip, ReturnTrip),
                rdf(Trip, seg:hasBeginPlace, HarborA),
                rdf(Trip, seg:hasEndPlace, HarborB)
            ),
            FerryTrips),
    FerryTrips \= [],
    forall(member(trip(From, FromHarbor, To, ToHarbor), Trips),
           rdf_bnode(FerryTrip),
           rdf_assert(FerryTrip, rdf:type, poseidon:etype_ferry_trip),
           rdf_assert(FerryTrip, sem:hasSubEvent, Trip),
           rdf_assert(FerryTrip, sem:hasSubEvent, ReturnTrip)).
```

Fig. 9 Second part of a code example illustrating how we use SWI-Prolog rules to derive complex event types (regular trips and ferry trips) from low-level segment events in SEM RDF format. The first part of this example is shown in figure 8. The rules shown in this figure show how the RDF assertions are made that classify the behavior exhibited in movement segments.

and less on the modeling of the detected events or their use for detection of higher-level events. Thus, event models applied in multimedia content analysis, if made at all explicit, typically lack media independence..." SEM addresses this by modeling events independently from the data.

In the future SEM will be used as a basic schema supported by the Semantic Search Engine ClioPatria[26] [25]. SEM will also be used in completely different domains than maritime safety, *e.g.* in Cultural Heritage and historical applications. In these domains SEM can also be used to bridge the gap between data (low-level object and fact descriptions) and semantics at the level of human queries by offering a new conceptual event-based semantic description. Although SEM, as an event model, does not provide all of the steps necessary for bridging the semantic gap (part of the bridging is done by signal processing and rules linking the different levels of abstraction together), it is

[26] `http://e-culture.multimedian.nl/software/ClioPatria.shtml`

at the core of the process: a unique interface for the representation of heterogeneous data, that allows for a unified reasoning.

As future work, we would like to extend the SWI-Prolog Space Package to deal with moving object indexing. This would allow us to write efficient rules about the relative position of moving ships with respect to each other. Currently, this is not possible, as we use an R*-tree which can not natively deal with time-parametrized geometries. We would like to extend the web information extraction toolkit we use to find ship information to find more properties of ships so that we can extend the range of queries we can formulate about ships (banned ships, historical records, etc.). A future challenge is to move from only using existing place features like harbors to also using automatically discovered points of interest, 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.

References

1. N. Andrienko and G. Andrienko. Designing visual analytics methods for massive collections of movement data. *Cartographica*, 42(2):117–138, 2007.
2. 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.
3. H. Cao, O. Wolfson, and G. Trajcevski. Spatio-temporal data reduction with deterministic error bounds. *VLDB J.*, 15(3):211–228, 2006.
4. D. Ceolin, W. R. van Hage, and W. Fokkink. A trust model to estimate quality of annotations using the web. In *WebSci10: Extending the Frontiers of Society On-Line*, 2010.
5. S. B. Claudio Masolo, A. Gangemi, N. Guarino, A. Oltramari, and L. Schneider. Wonderweb deliverable d18. ontology library library. Technical report, ISTC-CNR WonderWeb project, 2003.
6. N. Crofts, M. Doerr, T. Gill, S. Stead, and M. S. (editors). Definition of the cidoc conceptual reference model. online, November 2009.
7. 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.
8. A. Gangemi, C. Catenacci, and M. Battaglia. Inflammation ontology design pattern: an exercise in building a core biomedical ontology with descriptions and situations. *Ontolgies in Medicine*, pages 64–80, 2004.
9. J. Gudmundsson, J. Katajainen, D. Merrick, C. Ong, and T. Wolle. Compressing spatio-temporal trajectories. *Computational Geometry*, 42(9):825 – 841, 2009.
10. J. Hunter. Combining the cidoc crm and mpeg-7 to describe multimedia in museums. In *Museums and the Web international conference*, Boston, April 2002.
11. 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.
12. 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.

13. 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.
14. D. Orellana and C. Renso. Developing an interactions ontology for characterising pedestrian movement behavior. In M. Wachowicz, editor, *Movement-Aware Applications for Sustainable Mobility: Technologies and Approaches*. IGI Global Publishing, 2009.
15. Y. Raimond and S. Abdallah. The event ontology. online, 2007. http://purl.org/NET/c4dm/event.owl.
16. U. Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*, 1(2):244–256, 1972.
17. 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.
18. A. Scherp, T. Franz, C. Saathoff, and S. Staab. F—a model of events based on the foundational ontology dolce+dns ultralight. In *International Conference on Knowledge Capturing (K-CAP)*, Redondo Beach, CA, USA,, September 2009.
19. R. Shaw, R. Troncy, and L. Hardman. Lode: Linking open descriptions of events. In *4th Annual Asian Semantic Web Conference (ASWC'09)*, pages 153–167, Shanghai, China, December 6-9 2009.
20. 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.
21. S. Spaccapietra, C. Parent, M. L. Damiani, J. A. F. de Macêdo, F. Porto, and C. Vangenot. A conceptual view on trajectories. *Data Knowl. Eng.*, 65(1):126–146, 2008.
22. 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.
23. U. Westermann and R. Jain. E - a generic event model for event-centric multimedia data management in echronicle applications. In *ICDEW '06: Proceedings of the 22nd International Conference on Data Engineering Workshops*, page 106, Washington, DC, USA, 2006. IEEE Computer Society.
24. U. Westermann and R. Jain. Toward a common event model for multimedia applications. *IEEE Multimedia*, 14:19–29, 2007.
25. J. Wielemaker, M. Hildebrand, J. van Ossenbruggen, and G. Schreiber. Thesaurus-based search in large heterogeneous collections. pages 695–708, 2008.
26. J. Wielemaker, Z. Huang, and L. van der Meij. Swi-prolog and the web. In A. Bossi, editor, *Theory and Practice of Logic Programming*, volume 8, pages 363–392. Cambridge University Press, 2008.
27. M. F. Worboys and K. Hornsby. From objects to events: Gem, the geospatial event model. In M. J. Egenhofer, C. Freksa, and H. J. Miller, editors, *Third International Conference on Geographic Information Science, GIScience 2004*, volume 3234 of *Lecture Notes in Computer Science*, pages 327–344. Springer, 2004.