ESO Documentation (draft)

No Author Given

No Institute Given

Citations For any citations on ESO, please use the following reference:
Roxane Segers, Piek Vossen, Marco Rospocher, Luciano Serafini, Egoitz Laparra and German Rigau (f.c.). ESO: A Frame based Ontology for Events and Implied Situations. In: Proceedings of MAPLEX2015, Yamagata, Japan.

ESO.owl The owl version of ESO can be found at:
`http://www.newsreader-project.eu/results/event-and-situation-ontology/`

Introduction This documentation describes the Event and Situation Ontology (ESO), a resource which formalizes the pre and post conditions of events and the roles of the entities affected by an event. The ontology reuses and maps across existing resources such as Wordnet, SUMO and Framenet and is designed for extracting information from text that otherwise would have been implicit.
For example, the expression 'Y fires X' implies that X must have been working for Y *before* the firing and that X is not working for Y *after* the firing. Likewise, the expression 'X works for Y', states that some situation holds *during* some period of time. For deriving these implications, ESO defines a) classes of events and the implications these events; b) what entities are affected by an event and c) how the implications of dynamic and static events can be linked.

Following best practices in Semantic Web technologies, ESO reuses parts of two existing vocabularies: there are mappings from our ontology to Framenet on class and role level and mappings to SUMO on class level. As such, we can define our classes without adhering to modeling choices in Framenet and SUMO. Through these mappings, ESO serves as a hub to other vocabularies as well, such as Princeton Wordnet (PWN) and the Wordnets in the Global Wordnet grid.

The remainder of this documentation is as follows: first we briefly introduce FrameNet, next we define 'events' and 'situations'. In paragraph 3,4,5 and 6 the metamodel of the ontology and the instantiating of situations is explained. Finally in parapgraph 7 we provide an overview of the current contents of the ontology. In the appendix we provide the mapping tables from ESO to FrameNet and SUMO.

1. FrameNet
2. Event and Situations
3. Representation of event instances and corresponding situations
4. Core classes and properties of the NewsReader Domain Ontology
5. Formalization of the rules for instantiating situations from events
6. Mappings from external resources to the NewsReader Domain Ontology
7. Ontology content

1) Intermezzo: Framenet For the entities that are involved in a change, we build upon Framenet and Semantic Role Labeling (SRL). SRL is concerned with the detection of the semantic arguments associated with the predicate of a sentence and the classification of these arguments into their specific roles. For instance, given sentences like:

1. Henry fired John
2. Hillary gave the car to Bill
3. Ellen left New York yesterday

the words 'fire', 'give' and 'leave' represent predicates. These predicates have arguments such as a subject (Ford) and an object (the car). SRL abstracts further over these arguments and assigns semantic roles:

1. Ford [employer] fired John [employee]
2a. Hillary [donor] gave the car [theme] to Bill[recipient]
2b. The car [theme] was given to Bill [recipient] by Hillary [donor]
3. Ellen [theme] left New York[source] yesterday

Due to this abstraction, sentences that have a different syntactic representation will still have the same semantic roles as is evident from sentence 2a and 2b. In the NewsReader project, in the background of which ESO was developed, the labeling of the roles is based on FrameNet Frames. In FrameNet, verbs that share similarities in how the arguments and roles are realized, are associated into a so called Frame. A frame provides a set of core and non core slots or Frame Entities that specify the different roles that a predicate can evoke in a sentence. Further, FrameNet provides set of predicates is provided for which these roles apply.

In NewsReader, the Predicate Matrix is used that integrates predicate and role information from several resources such as FrameNet, VerbNet, PropBank and Wordnet. As such, Framenet role and predicate annotations are assigned on document level. All definitions and assertions in ESO are fed back to the Predicate Matrix and as such to the documents. In this way, the ontology provides an additional layer of annotations that allows inferencing over events and implications. Note however, that ESO is developed on top of a subset of FrameNet frames. More information on which frames and roles ESO is mapped to, is given in paragraph 7.

2) Events and Situations To be able to represent events and situations, ESO defines two main classes of entities: events and situations. An *event* is an entity that describes some change in the world. It has participants and a time (interval) associated to it. An event exists independently from the fact that it actually happens (e.g., hypothetical events). Typically, an event is associated with two situations: the situation before the event (pre-situation) and the one after the event (post-situation). The effects of an event are described in terms of the statements that hold in the situations associated to the event.

If we consider for instance a firing event:

In 2012, employeeA and employeeB were fired by companyA

we can identify a pre-situation (i.e., before the event):

employeeA works for companyA
employeeB works for companyA

as well as a post-situation (i.e., after the event):

employeeA does not work for companyA
employeeB does not work for companyA

A *situation* is an entity which is associated with a period of time where a set of statements (aka *fluents* in situation calculus) are true. It is a partial and "perspectival" description of the state of the world during the period of time it is associated with. It is partial because it does not describe the totality of propositions that are true in the world during the period of time associated to the situation. It is perspectival because it describes the point of view of a particular "agent".

3) Representation of event instances and corresponding situations In the original situation calculus the predicate "holdsAt($r(a, b), s$)" is used to model the fact that "a and b are related with the relation r in situation s". In our proposal, we adopt recent advances in Semantic Web technologies, relying on the notion of "named graph": a named graph will be associated to each situation s, and it will contain all triples a, R, b holding in it.

Let's consider the aforementioned firing event example. The SRL module of the NewsReader annotates the sentence "In 2012, employeeA and employeeB were fired by companyA" with the following information:

- fired → frame fn:firing;
- employeeA → frame element fn:Employee of frame fn:Firing;
- employeeB → frame element fn:Employee of frame fn:Firing;
- companyA → frame element fn:Employer of frame fn:Firing;

In addition, a time expression will be associated to the term "in 2012".

From this linguistic annotations, we instantiate some individuals and assertions on them to formally represent the event according to standard Semantic Web formalisms. In details, we instantiate a named graph of the form

```
:obj-graph-eventX {
    :eventX
        a                                    nwr:LeavingAnOrganization ;
        nwr:LeavingAnOrganization_employee   :employeeA                ;
        nwr:LeavingAnOrganization_employee   :employeeB                ;
        nwr:LeavingAnOrganization_employer   :companyA                 ;
        sem:hasTime                          :time_eventX              .
}
```

These statements specify that the event is of a certain type (nwr:LeavingAnOrganization), that it involves a entity playing the role of an employer (:companyA) and two entities playing the role of employees (:employeeA,:employeeB), and that it occurred at a certain time (:time_eventX).

A "nwr:LeavingAnOrganization" event in turn, triggers the instantiation of two situations, one preceding the event (:obj-graph-pre-situation-eventX) and one following the event (:obj-graph-post-situation-eventX):

```
:obj-graph-eventX {
    :eventX
        nwr:hasPreSituation        :obj-graph-pre-situation-eventX  ;
        nwr:hasPostSituation       :obj-graph-post-situation-eventX .
}
```

As previously mentioned, each of these situations corresponds to a name graph containing assertions holding in them. In particular, for the example considered we instantiate the following two named graphs:

```
:obj-graph-pre-situation-eventX {
    :companyA   nwr:employ    :employeeA    ;
                nwr:employ    :employeeB    .
}

:obj-graph-post-situation-eventX {
    :companyA   nwr:notEmploy :employeeA    ;
                nwr:notEmploy :employeeB    .
}
```

stating that before the firing event, both employeeA and employeeB were employed at the company, while after the firing event none of them was working for the company.

Additional assertions may be attached to situation named graphs. These assertions may be used to characterize the time span of the situation, or the provenance of the statements defined in the situation. For instance, the assertions

```
:instances {
    :obj-graph-pre-situation-eventX
         a                nwr:Situation                         ;
         nwr:hasTime      :obj-graph-pre-situation-eventX-time ;
         nwr:producedBy   nwr:reasoner                          .
    :obj-graph-post-situation-eventX
         a                nwr:Situation                         ;
         nwr:hasTime      :obj-graph-post-situation-eventX-time;
         nwr:producedBy   nwr:reasoner                          .
    :obj-graph-pre-situation-eventX-time
         a                time:Interval    ;
         time:hasEnd      :time_eventX     .
    :obj-graph-post-situation-eventX-time
         a                time:Interval    ;
         time:hasBeg      :time_eventX     .
}
```

permit to assert that the two situations were instantiated by the agent nwr:reasoner, that obj-graph-pre-situation-eventX was in place before eventX, and that obj-graph-post-situation-eventX is in place after eventX. Likewise, we are be able to distinguish situations that are explicitly described in the text and claimed by the sources from situations that are indirectly derived through the nwr:reasoner. In the former case, the named graph has an nwr:attributedTo property with the source, and in the latter case the nwr:producedBy property to the reasoner.

4) Core classes and properties of the NewsReader Domain Ontology
The NewsReader Domain Ontology contains five core classes, which are further specialized in subclasses:

Event : this class is the root of the taxonomy of (proper) event types. Any event detected in a text will be an instance of some class of this taxonomy;

DynamicEvent : this is a subclass of Event (for which dynamic changes are defined) that apply to FrameNet frames that can be considered as proper events (e.g., fn:firing);

StaticEvent : this is another subclass of Event for "static" event types and which capture more static circumstances (e.g., fn:possession, fn:organization); they typically directly trigger a situation holding at the time the event occurs (a "during situation", differently from pre/post-situations in proper events); a "static" event detected in a text will be an instance of some class of this taxonomy;

Situation : the individuals of this class are actual pre/post/during situations that will be instantiated starting from the event instances detected in the text;

SituationRule : the individuals of this class enable to encode the rules for instantiating pre/post/during situations when a certain type of event is detected;

SituationRuleAssertions : the individuals of this class enable to encode the assertion that has to be instantiated within each pre/post/during situation associated to some event.

Analogously to FrameNet frame elements for frames, ESO enables to represent the role of an entity in an event. Roles are formalized as object properties: this way, an event instance :eventX can be related to an entity :entityZ participating in it with assertions of the form:

:eventX nwr:hasRoleY :entityZ

where nwr:hasRoleY specify the role of :entityZ in :eventX. Each object property defining a role in ESO is defined as subproperty of the top object property nwr:hasRole: this way, given any event, we can retrieve the entities participating in it by looking at assertions having as predicate the property nwr:hasRole.

Additional object properties are defined to enable:

– relating an event instance with the actual pre/post/during situations it triggers (resp., object property nwr:hasPreSituation, nwr:hasPostSituation, and nwr:hasDuringSituation);
– relating an event type with the pre/post/during situation rules that should be triggered when an instance of that event type is detected (resp. nwr:triggersPreSituation, nwr:triggersPostSituation, and nwr:triggersDuringSituation);
– relating a situation rule with the assertions that should be instantiated within the situation named graph associated with the rule (resp., nwr:hasSituationRuleAssertion).

Finally, ESO specifies the properties that can be used as predicate in assertions within a situation named graph.

binary properties : these properties are modelled as object properties and they enable to relate two entities;

unary properties : these properties are modelled as datatype properties and they enable to express facts such as that an entity exists. Typically, the range of such properties is a boolean value type.

For binary properties, whenever appropriate, we defined additional properties characteristics. In particular, two important characterization are in-place:

disjoint properties : two binary properties p, q are defined as disjoint if no individual a can be connected to an individual b by both triples $a\ p\ b$ and $a\ q\ b$.

inverse properties : if two binary properties p, q are defined as one the inverse of the other, an assertion $a\ p\ b$ implies also the assertion $b\ q\ a$, and viceversa.

For instance, in ESO we defined “nwr:employ” and “nwr:notEmploy” as disjoint (only one of the two can hold at a certain time), as well as “nwr:employ” and “nwr:employedAt” as inverse properties (if :companyA nwr:employ :employeeB, then :employeeB nwr:employedAt :companyA holds, and viceversa).

5) Formalization of the rules for instantiating situations from events
The formalization of the rules for instantiating situations from events consists in defining the assertions to be instantiated in pre/post/during situations of an event, based on the roles of the entities involved in it. We rely on a two level schema: first, we defined for each event type the kind of situations they have to trigger (i.e., whether pre/post/during situations); then, for each situation triggered by an event, we formalized the type of assertions that have to be instantiated, specifying how the roles of the event triggering the situation map to the assertions' subject and object. We illustrate this with a concrete example, based on the event type “ChangeOfPossession”, which refers to the event when something (role “possession-theme”) passes from an entity (role “possession-owner_1”) to another entity (role “possession-owner_2”). An event of type “ChangeOfPossession” has to trigger a pre-situation and a post-situation, each of them asserting some possession statements. To model the relation between an event type and the type of situations it triggers we relied on owl:hasValue restrictions:

```
nwr:ChangeOfPossession   rdfs:subClassOf [
a owl:Restriction ;
    owl:hasValue   nwr:pre_ChangeOfPossession   ;
    owl:onProperty nwr:triggersPreSituationRule ] .

nwr:ChangeOfPossession   rdfs:subClassOf [
    a owl:Restriction ;
    owl:hasValue   nwr:post_ChangeOfPossession   ;
    owl:onProperty nwr:triggersPostSituationRule ] .

nwr:pre_ChangeOfPossession    a nwr:SituationRule .
nwr:post_ChangeOfPossession   a nwr:SituationRule .
```

Note that, by defining the "rule" for instantiating situations based on owl:hasValue restrictions, we can exploit reasoning to infer that the same pre/post/during situations have to be triggered for any event type more specific than the considered one: e.g., if we are considering an event of type nwr:Getting, and nwr:Getting is a subclass of nwr:ChangeOfPossession, the same rules for situations defined for nwr:ChangeOfPossession automatically apply also for nwr:Getting, without having to redefine them.

Each nwr:SituationRule individual is specialized to define exactly how the triples inside the Situation named graph has to be defined. This is done by defining an individual (of type SituationRuleAssertion) for each assertion to be created, having three annotation properties assertions:

nwr:hasSituationAssertionSubject : the object of this triple is the role of the event to be used as subject in the assertion;

nwr:hasSituationAssertionProperty : the object of this triple is the predicate to be used in the assertion. It is either a binary property or an unary property;

nwr:hasSituationAssertionObject : the object of this triple is the role of the event or the data value (in case of unary properties) to be used as object in the assertion.

Consider for instance the nwr:pre_ChangeOfPossession situation rule:

```
nwr:pre_ChangeOfPossession
    nwr:hasSituationRuleAssertion    pre_ChangeOfPossession_assertion1;
    nwr:hasSituationRuleAssertion    pre_ChangeOfPossession_assertion2.
```

This rule triggers the instantiation of two assertions, nwr:pre_ChangeOfPossession_assertion1 and nwr:pre_ChangeOfPossession_assertion2, defined as follow:

```
nwr:pre_ChangeOfPossession_assertion1
    nwr:hasSituationAssertionSubject     nwr:possession-owner_1;
    nwr:hasSituationAssertionProperty    nwr:hasInPossession;
    hasSituationAssertionObject          nwr:possession-theme.

nwr:pre_ChangeOfPossession_assertion2
    nwr:hasSituationAssertionSubject     nwr:possession-owner_2;
```

```
    nwr:hasSituationAssertionProperty    nwr:notHasInPossession;
    hasSituationAssertionObject          nwr:possession-theme.
```

Therefore, from an event instance :eventX of type nwr:ChangeOfPossession, having roles :instanceX (nwr:possession-owner_1 role), :instanceY (nwr:possession-owner_2 role), and :instanceZ (nwr:possession-theme role), by interpreting the aforementioned rule schema we can instantiate a pre-situation named graph, :eventX_pre, defined as follow:

```
:eventX_pre {
    :instanceX    nwr:hasInPossession      :instanceZ .
    :instanceY    nwr:nothasInPossession    :instanceZ .
}
```

where the first assertion is created due to nwr:pre_ChangeOfPossession_assertion1, while the second assertion is due to nwr:pre_ChangeOfPossession_assertion2.

6) Mappings from external resources to the NewsReader Domain Ontology A key ingredient of the NewsReader Domain Ontology is the mapping of the FrameNet frames and frame elements to the event types and roles that we defined. This mapping is necessary to translate the annotations provided by the SRL module to our ontology vocabulary, exploited by the reasoning module to instantiate situations from events.

For each event type (modelled as class in the NewsReader Domain Ontology) and each role (modelled as object property in the NewsReader Domain Ontology) we defined some annotations (nwr:correspondsToFrameNetFrame and nwr:correspondsToFrameNetElement) representing the corresponding frames and frame elements. For instance, the following assertions via property nwr:correspondsToFrameNetFrame are defined for the event type nwr:Giving:

```
nwr:Giving    nwr:correspondsToFrameNetFrame    fn:Giving,fn:Sending,fn:Supply.
```

meaning that, if a frame of type fn:Giving, fn:Sending, or fn:Supply is identified in the text, it has to be considered as an event of type nwr:Giving, and therefore pre/post/during situation rules defined for nwr:Giving should be triggered.

Similarly, given the role nwr:possession-owner_1, it is mapped to the following frame elements with the nwr:correspondsToFrameNetElement assertions:

```
nwr:Giving   nwr:correspondsToFrameNetElement   fn:Seller,fn:Supplier,fn:Lender,
                                                fn:Sender,fn:Donor,fn:Source,
                                                fn:Agent,fn:Exporter,fn:Victim.
```

We also defined the mapping from the NewsReader Domain Ontology event types to SUMO[1] classes (as explained in section **??**), via nwr:correspondsToSUMOClass annotation assertions. E.g., the following mapping of nwr:Giving to a SUMO class was defined:

```
nwr:Giving    nwr:correspondsToFrameNetFrame    fn:Giving,fn:Sending,fn:Supply.
```

An overview of the mappings from ESO classes to FrameNet and SUMO is given in Figure 1.

[1] `http://www.ontologyportal.org`

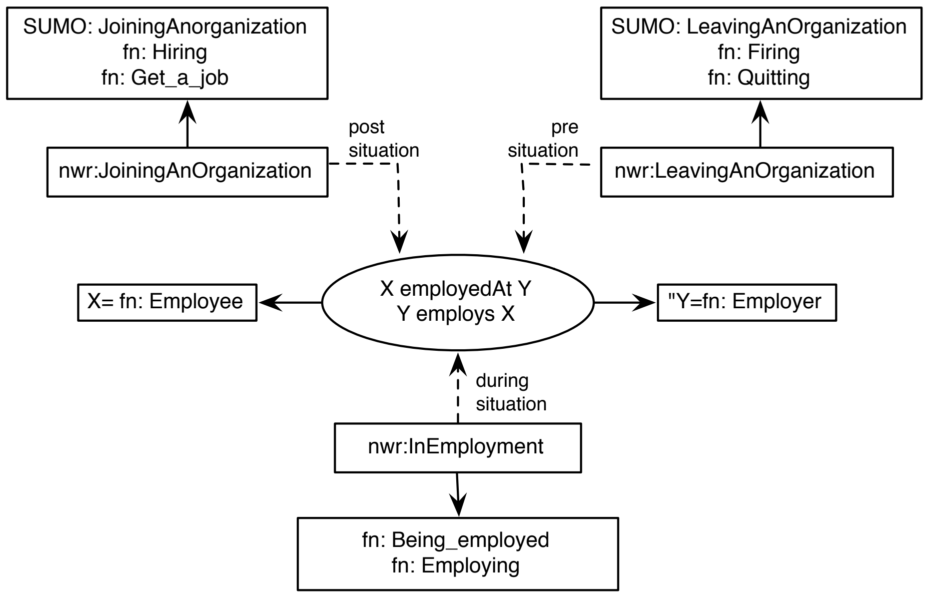

Fig. 1. Mappings from ESO classes to Framenet and SUMO.

7) Ontology contents The first version of the ESO now consists of 59 event classes divided over dynamic events (50) and static events (9). The dynamic event class hierarchy consists of four major nodes: ChangeOfPossession (16 subclasses), Translocation (10 subclasses), InternalChange (11 subclasses) and IntentionalEvents (11 subclasses). An overview of the dynamic hierarchy is given in Figure 2, the static events are represented in Figure 3.

For 53 classes we have one or multiple mappings to FrameNet frames. In total, 94 mappings to FrameNet were made, covering 532 unique combinations of a predicate and a frame. Additionally, 49 out of 59 event classes have a mapping to SUMO. Furthermore, we defined 24 properties (20 binary and 4 unary) such as 'atPlace', 'employedAt' and 'hasInPossession' which define the situations statements for 35 out of 50 dynamic event classes and all 9 static event classes. Finally, we defined 33 different roles for the entities affected by an event or situation. Each role is mapped to one or more Frame Entities in FrameNet (60 mappings in total).

All mappings from ESO classes to SUMO and Framenet Frames and Frame Elements are represented in the tables in the appendix of this document. The classes in these tables are organized conceptually into dynamic event classes: 1) Translocations 2) Change of possession, 3) Intentional events and 4) Internal change. Furthermore, the static events and their mappings are representes in 5) Static Events.

For all classes pertaining to Translocation and ChangeOfPossession holds that the mappings from ESO to FrameNet Frame Elements apply to all classes. For the other dynamic and static event classes, the mappings from roles to Frame Entities are specified per class.

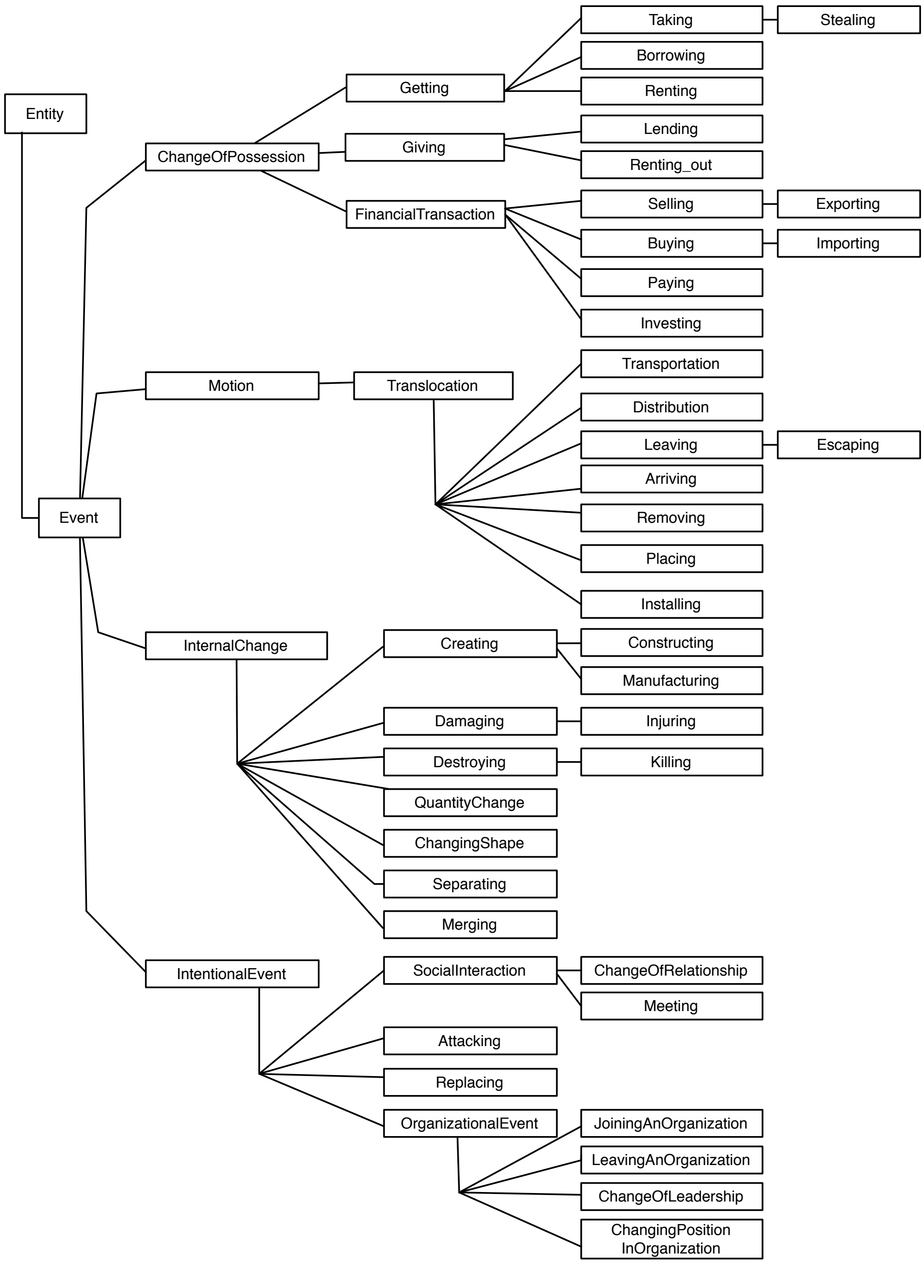

Fig. 2. Overview of the dynamic event class hierarchy

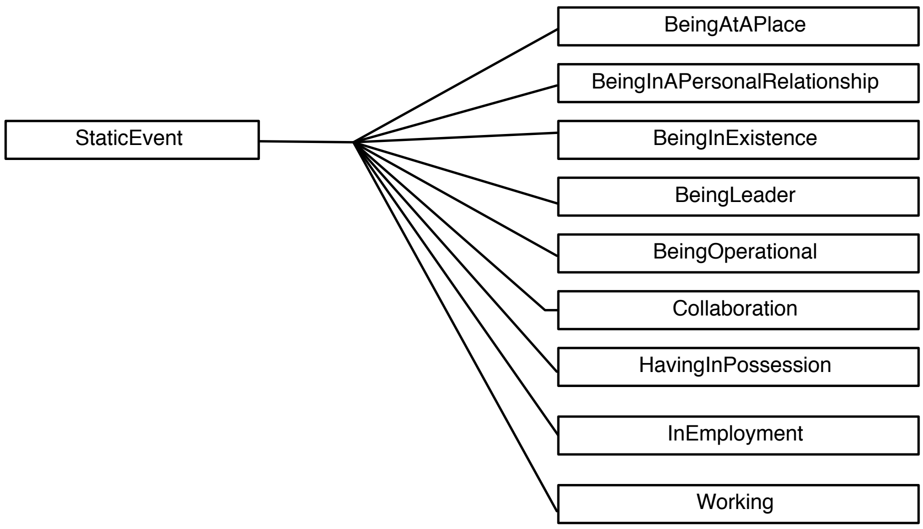

Fig. 3. Overview of the static event classes

APPENDIX

Mapping tables from ESO classes to SUMO and FrameNet Frames
and from ESO roles to FrameNet Frame Elements

1)

ESO	SUMO	FrameNet	ESO roles	Framenet Frame Entity
Translocation				
Arriving	Translocation	Arriving	translocation_source	Source
		Vehicle_landing		
			translocation_goal	Goal
Distribution	_	Dispersal		
			translocation_theme	Deliverer
Escaping	Escaping	Escaping		Self_mover
		Fleeing		Agent
				Driver
Installing	Installing	Installing		Carrier
				Traveler
Leaving	Leaving	Departing		Vehicle
		Quitting_a_place		Escapee
		Setting_out		Cotheme
		Vehicle_departure		Individuals
				Component
Placing	Putting	Placing		
Removing	Removing	Removing		
Transportation	Transportation	Delivery		
		Bringing		
Translocation	Translocation	Cause_motion		
		Cotheme		
		Intentional_traversing		
		Operate_vehicle		
		Ride_vehicle		
		Self_motion		
		Travel		
		Traversing		
		Use_vehicle		

2)

ESO	SUMO	FrameNet
ChangeOfPossession		
Borrowing	Borrowing	Borrowing
Buying	Buying	Commerce_buy
ChangeOfPossession	ChangeOfPossession	_
Exporting	Exporting	Exporting
FinancialTransaction	FinancialTransaction	Commercial_transaction
Getting	Getting	Getting Receiving
Giving	Giving	Giving Sending Supply
Importing	Exporting	Importing
Investing	Investing	_
Lending	Lending	Lending
Paying	Payment	Commerce_pay
Renting	Renting	Renting
RentingOut	_	Renting_out
Selling	Selling	Commerce_sell
Stealing	Stealing	Theft
Taking	UnilateralGetting	Taking

ESO roles	Framenet Frame Entity
possession-owner_1	Agent Donor Exporter Lender Seller Sender Source Supplier Victim
possession-owner_2	Borrower Buyer Goal Lessee Perpetrator Recipient
possession-theme	Possession Theme

3)

ESO	SUMO	Framenet	ESO roles	FrameNet Frame Entity
IntentionalEvent				
Attacking	Attack	Attack	_	
ChangeOfLeadership	_	Change_of_leadership	_	
ChangeOfRelationship	_	Forming_relationships	_	
ChangingPositionInOrganizatic	TransferringPosition	_	_	
IntentionalEvent	IntentionalProcess	Intentionally_act	_	
JoiningAnOrganization	JoiningAnOrganization	Get_a_job Hiring	employment-employer employment-employee	Employer Employee
LeavingAnOrganization	LeavingAnOrganization	Firing Quitting	employment-employer employment-employee	Employer Employee
Meeting	Meeting	Assemble Come_together Social_event	_	
OrganizationalEvent	OrganizationalEvent	_	_	
Replacing	Substituting	Replacing Take_place_of	_	
SocialInteraction	SocialInteraction	_	_	

4)

ESO	SUMO	Framenet	ESO roles	FrameNet Frame Entity
InternalChange				
ChangingShape	ShapeChange	Manipulate_into_shape Reshaping	_	
Constructing	Constructing Making	Building	creating-theme	Created_entity Product
Creating	Creation	Creating Intentionally_create	creating-theme	Created_entity Product
Destroying	Destruction	Cause_to_fragment Destroying	destroying-theme	Executed Undergoer Victim Whole_patient
Injuring	Injuring	Cause_harm Experience_bodily_harm	_	
InternalChange	InternalChange		_	
Killing	Killing	Execution Killing	destroying-theme	Executed Undergoer Victim Whole_patient
Manufacturing	Manufacture	Manufacturing	creating-theme	Created_entity Product
Merging	Combining	Amalgamation Cause_to_amalgamate	merging-theme_1 merging-theme_2 merging-theme_3	Part_1 Part_2 Whole
QuantityChange	QuantityChange	Cause_change_of_position_on_a_scale Cause_expansion Cause_proliferation_in_number	quantity-theme	Item Set
		Change_of_quantity_of_possession Change_position_on_a_scale Expansion Proliferating_in_number	quantity-value_1	Initial_number Initial_size Initial_value Value_1
			quantity-value_2	Final_number Final_value Result_size Value_2
Separating	Separating	Becoming_separated Separating	separating-theme_1 separating-theme_2 separating-theme_3	Part_1 Part_2 Whole

ESO	SUMO	Framenet	ESO roles	FrameNet Frame Entity
STATIC EVENTS				
BeingAtAPlace	_	Being_located Presence Residence Temporary_stay	atPlace-theme atPlace-location	Entity Guest Resident Theme
BeingInAPersonalRelationship	_	Personal_relationship	relationship-partner_1 relationship-partner_2	Partner_1 Partner_2
BeingInExistence	_	Existence	exist-theme	Entity
BeingLeader	Leadership	BeingLeader	leader-entity leader-governed_entity	Leader Governed
BeingOperational	_	Being_operational	operational-theme	Device
Collaboration	_	Collaboration	collaboration-partner_1 collaboration-partner_2	Partner_1 Partner_2
HavingInPossession	_	Possession	possession-theme	Possession Theme
			possession-owner	Owner
InEmployment	_	Being_employed Employing	employment-employer employment-employee	Employer Employee
Working	_	Working	working-entity	Agent