

# ESO Documentation (Draft)

Roxane Segers, Marco Rospocher

September 11, 2015

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

**Contact** For any questions, suggestions and remarks, please contact:  
Roxane Segers, Network Institute, VU Amsterdam (r.h.segers@vu.nl)  
Marco Rospocher, Fondazione Bruno Kessler (rospocher@fbk.eu)

**ESO.owl** The OWL version of ESO can be found at:  
<https://github.com/newsreader/eso>

## Table of Content

1. Introduction
2. A short introduction to Semantic Role Labeling and FrameNet
3. ESO: Ontological Metamodel and Instantiation
  - 3.1 Representing vent instances and corresponding situations
  - 3.2 Core Classes and Properties of ESO
  - 3.3 Formalization of the rules for instantiating situations from events
  - 3.4 Mappings from external resources to ESO
4. ESO: Content Description
  - 4.1 Building ESO

#### 4.2 The ESO event class hierarchies and mappings

#### 4.3 Properties and roles for defining the situation assertions

#### 4.4 Content overview of ESO

### Appendix

## 1 Introduction

This documentation describes the Event and Implied 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 this, we rely on Semantic Role Labeling techniques.

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. Such a chain of events and implied situations is presented in ?? 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.

## 2 A Short Introduction in Semantic Role Labeling and 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*

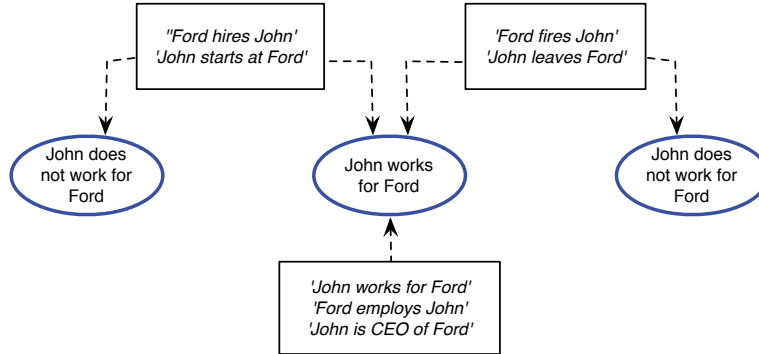


Figure 1: A chain of static and dynamic events and their implied situation

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.

### 3 ESO: Ontological Metamodel and Instantiation

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 or state 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 three situations: the situation before the event (pre-situation), the situation during the event (during-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 (a change in the world):

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

And if we consider an employment event (some state in the world):

In 2011, employeeA and employeeB are employed by companyA

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

employeeA works for companyA  
employeeB works 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”.

### 3.1 How to represent an event instance and its 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. Thanks to the Predicate Matrix which aligns PropBank information (as returned by MATE tools) to FrameNet labels, the SRL module of the pipeline will annotate the sentence “In 2012, employeeA and employeeB were fired by companyA” with the following information:

- fired  $\rightarrow$  frame fn:Firing;
- employeeA  $\rightarrow$  frame element fn:Employee of frame fn:Firing;
- employeeB  $\rightarrow$  frame element fn:Employee of frame fn:Firing;
- companyA  $\rightarrow$  frame element fn:Employer of frame fn:Firing;

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

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

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

These statements specify that the event is of a certain type (eso: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 “eso:LeavingAnOrganization” event will in turn trigger 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
    eso:hasPreSituation      :obj-graph-pre-situation-eventX ;
    eso:hasPostSituation     :obj-graph-post-situation-eventX .
}
```

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

```

:obj-graph-pre-situation-eventX {
  :companyA    eso:employs    :employeeA    ;
               eso:employs    :employeeB    .
}

:obj-graph-post-situation-eventX {
  :companyA    eso:notEmploys :employeeA    ;
               eso:notEmploys :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          eso:Situation          ;
    eso:hasTime :obj-graph-pre-situation-eventX-time ;
    nwr:producedBy eso:reasoner      .
  :obj-graph-post-situation-eventX
    a          eso:Situation          ;
    eso:hasTime :obj-graph-post-situation-eventX-time;
    nwr:producedBy eso: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 will be able to distinguish events 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.

In order to enable expressing events, situations, and to define the conditions and modalities on how to trigger such situations starting from events, ESO has to fulfil some requirements:

- define the core classes (e.g., Event, Situation) and the basic properties that enable relating them (e.g., to state that a Situation S is a pre-situation of an event E);
- define the type of events that are relevant, potentially abstracting from the specific way an event is mentioned in the text, so that different variants of the same event (e.g., firing, sacking) can be treated the same way;

- organize events into a taxonomy so to exploit the inferencing capabilities on the subclass relation between events (i.e., if an event triggers some situations, every event more specific than it should trigger the same situations);
- define how situations are triggered by events, specifying which assertions to instantiate in each situation.

### 3.2 Core classes and properties of ESO

The Event and Situation Ontology contains five core classes, which are further specialized in subclasses:

**Event** : this class is the root of the taxonomy of (proper) event types considered.. 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 considered and which capture more static circumstances (e.g., fn:Employing, fn:Possession). They typically directly trigger a situation holding at the time the event occurs (a “during situation”). 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 eso:hasRoleY :entityZ

where `eso: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 `eso:hasRole`: this way, given any event, we can retrieve the entities participating in it by looking at assertions having as predicate the property `eso:hasRole`.

Additional object properties are defined to enable:

- relating an event instance with the actual pre/post/during situations it triggers (resp., object property `eso:hasPreSituation`, `eso:hasPostSituation`, and `eso: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. `eso:triggersPreSituation`, `eso:triggersPostSituation`, and `eso:triggersDuringSituation`);
- relating a situation rule with the assertions that should be instantiated within the situation named graph associated with the rule (resp., `eso:hasSituationRuleAssertion`).

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

**binary properties** : these properties are modelled as object properties and they enable to relate two entities (e.g., see property “`eso:employs`” and “`eso:notEmploys`” in the situations instantiated for the firing event example previous considered);

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

For binary properties, whenever appropriate, we defined additional property characteristics. In particular, three important characterizations 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.

**symmetric properties** : if two individuals  $a, b$  are related by a symmetric property, then the assertion  $a \ p \ b$  also implies the assertion  $b \ p \ a$ .



For instance, in ESO we defined “eso:employs” and “eso:notEmploys” as disjoint (only one of the two can hold at a certain time), as well as “eso:employs” and “eso:employedAt” as inverse properties (if :companyA eso:employs :employeeB, then :employeeB eso:employedAt :companyA holds, and viceversa). Further, we defined e.g. inRelationshipWith as a symmetric property; if A eso:inRelationshipWith B, then B eso:inRelationshipWith A.

### 3.3 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 define for each event type the kind of situations they have to trigger. Then, for each situation triggered by an event, we formalize 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 rely on owl:hasValue restrictions:

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

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

eso:pre_ChangeOfPossession    a eso:SituationRule .
eso:post_ChangeOfPossession    a eso:SituationRule .
```

Note that, by defining the “rule” for instantiating situations based on owl:hasValue restrictions, we can later 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 eso:Getting, and eso:Getting is a subclass of eso:ChangeOfPossession, the same rules for situations defined for eso:ChangeOfPossession automatically apply also for eso:Getting, without having to redefine them.

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

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

**eso: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;

**eso: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 `eso:pre_ChangeOfPossession` situation rule:

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

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

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

eso:pre_ChangeOfPossession_assertion2
  eso:hasSituationAssertionSubject    eso:possession-owner_2;
  eso:hasSituationAssertionProperty    eso:notHasInPossession;
  hasSituationAssertionObject          eso:possession-theme.
```

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

```
:eventX_pre {
  :instanceX    eso:hasInPossession    :instanceZ .
  :instanceY    eso:notHasInPossession  :instanceZ .
}
```

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

### 3.3.1 Adaptation of the instantiation of the assertions in ESO Version 2

In specific cases we also allow that assertions are instantiated even though no instance exists for the ESO role. We do this by adding an OWL existential restriction on the event class for the role considered. The reasoner will check if an instance of the role exists, if not it will create a blank node. The OWL existential restriction is applied in ESO to event classes that express a relative change in the value of an attribute (e.g. Damaging, Increasing, Attacking) where the attribute itself such as 'price' or 'damagedness' often remains implicit. As such, it is possible to assert statements based on 'incomplete' information if needed. It is possible to use this

restriction for more assertions as is done now in ESO. In this way, for each assertion it can be defined whether or not it should be instantiated if no instance for a role is found. This allows for extracting partial statements about instances.

We will explain the instantiation of assertions with this restriction by means of the class `eso:Increasing` where the value of some attribute (e.g. volume, speed, price) increases relatively to some previous value of the attribute: "Mary increased the production". This event corresponds to the triples:

```
:eventX_pre a eso:Increasing ;
            eso:quantity-item    :production ;
```

We defined the restriction on the role (quantity-attribute) at the class level:

```
eso:Increasing rdfs:subClassOf [
  a owl:Restriction ;
  owl:onProperty      eso:triggersPreSituationRule ;
  owl:hasValue        eso:pre_Increasing ] .
eso:Increasing rdfs:subClassOf [
  a owl:Restriction ;
  owl:onProperty      eso:triggersPostSituationRule ;
  owl:hasValue        eso:post_Increasing ] .
eso:Increasing rdfs:subClassOf [
  a owl:Restriction ;
  owl:onProperty      eso:quantity-attribute ;
  owl:someValuesFrom  owl:Thing] .

eso:pre_Increasing a eso:SituationRule .
eso:post_Increasing a eso:SituationRule .
```

These are the situation rule assertions defined for the pre and post situation of `eso:Increasing`:

```
eso:pre_Increasing_assertion1
  eso:hasSituationAssertionSubject    eso:quantity-item;
  eso:hasSituationAssertionProperty   eso:hasAttribute;
  eso:hasSituationAssertionObject     eso:quantity-attribute.

eso:pre_Increasing_assertion2
  eso:hasSituationAssertionSubject    eso:quantity-attribute;
  eso:hasSituationAssertionProperty   eso:hasRelativeValue;
  eso:hasSituationAssertionObjectValue '-'

eso:post_Increasing_assertion1
  eso:hasSituationAssertionSubject    eso:quantity-item;
  eso:hasSituationAssertionProperty   eso:hasAttribute;
  eso:hasSituationAssertionObject     eso:quantity-attribute.

eso:post_Increasing_assertion2
  eso:hasSituationAssertionSubject    eso:quantity-attribute;
  eso:hasSituationAssertionProperty   eso:hasRelativeValue;
  eso:hasSituationAssertionObjectValue '+'
```

The pre and post situation named graphs for our example sentence "Mary increased the production" can now be instantiated as follows:

```
:eventX_pre {
  :production      eso:hasAttribute      :xyz123
  :xyz123          eso:hasRelativeValue  '-'
:
:eventX_post {
  :production      eso:hasAttribute      :xyz123
  :xyz123          eso:hasRelativeValue  '+'
}
```

These instantiations can be paraphrased as follows: the production has some unknown attribute and the value of this attribute has become more (+) after the event then it was before the event (-), meaning that the production goes from less (-) to more (+).

Alternatively, if the attribute is known, the assertions will instantiate the role that models the actual attribute. For a sentence like "Mary increased the price of the components", the event will look as follow:

```
:eventX_pre a eso:Increasing ;
eso:quantity-item      :component ;
eso:quantity-attribute :price ;
```

and the assertions will be instantiated as follows:

```
:eventX_pre {
  :component      eso:hasAttribute      :price
  :price          eso:hasRelativeValue  '- '
:}
:eventX_post {
  :component      eso:hasAttribute      :price
  :price          eso:hasRelativeValue  '+ '
:}
```

### 3.4 Mappings from external resources to ESO

A key ingredient of ESO is the mapping from ESO roles to FrameNet Frame Entities and from ESO classes to FrameNet frames and SUMO<sup>1</sup> classes. The mappings to FrameNet are 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 class and each role in ESO, we defined a set of annotations representing the corresponding frames and frame elements:

- `correspondToFrameNetFrame_relatedMatch`: this property is defined to refer to FrameNet frames that express a related concept
- `correspondToFrameNetFrame_closeMatch`: this property is defined to refer to FrameNet frames that more or less express the same concept
- `correspondToFrameNetFrame_broadMatch`: this property is defined to refer to FrameNet frames that express a more general concept
- `correspondToFrameNetElement`: this property is defined to refer to FrameNet Frame Entities
- `correspondToSUMOClass_relatedMatch`: this property is defined to refer to FrameNet frames that express a related concept

---

<sup>1</sup><http://www.ontologyportal.org>

- `correspondToSUMOClass_closeMatch`: this property is defined to refer to FrameNet frames that more or less express the same concept
- `correspondToSUMOClass_broadMatch`: this property is defined to refer to FrameNet frames that express a more general concept

For instance, the following assertions via property `eso:correspondsToFrameNetFrame_closeMatch` are defined for the event type `eso:Giving`:

```
eso:Giving    eso:correspondsToFrameNetFrame_closeMatch
              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 `eso:Giving`, and therefore pre/post/during situation rules defined for `eso:Giving` should be triggered.

The `correspondToFrameNetFrame_broadMatch` property is used to specify that some ESO class is related to a FrameNet frame that expresses a more general concept. The following mappings are defined for the event class `eso:Increasing`:

```
eso:Increasing    eso:correspondsToFrameNetFrame_broadMatch
                  fn:Change_of_quantity_of_possession
                  fn:Cause_change_of_position_on_a_scale
                  fn:Change_position_on_a_scale
                  fn:Proliferating_in_number
                  fn:Expansion
                  fn:Cause_expansion
```

meaning that, if a frame of type `fn:Change_of_quantity_of_possession` is identified in the text, it a) has to be considered as an event of type `eso:Increasing`, and b) the pre and post situation assertions defined for `eso:Increasing` should be triggered only for a subset of the predicates associated to this frame. As such, these mappings allow to define assertions on a more specific level than the FrameNet frame. The `correspondToFrameNetFrame_relatedMatch` property is defined in ESO but currently not in use. (See also Section 4.2 on the mappings from ESO to FrameNet and SUMO.)

For ESO roles we use the `eso:correspondsToFrameNetElement` property. For instance the ESO role `eso:possession-owner-1` is mapped to the following frame elements, meaning that if a `fn:Seller` or `fn:Victim` is identified in text, it is considered to be of the type `eso:possession-owner-1`:

```
eso:possession-owner-1    eso:correspondsToFrameNetElement
                          fn:Seller,fn:Supplier,fn:Lender,
                          fn:Sender,fn:Donor,fn:Source,
                          fn:Exporter,fn:Victim, Exporting_area.
```

## 4 The Event and Situation Ontology: Content Description

In this section we first explain how ESO was build, next we describe the various content of ESO in more details with respect to the class hierarchy, the roles and the assertions.

## 4.1 Building the Event and Situation Ontology

As a first step in building a domain specific ontology, we carried out a statistical analysis of the events in a subset of the car data set. We chose to include only events related to FrameNet for this analysis as the frames associated to predicates provide a set of roles (Frame entities); both are needed to formulate the pre and post situations of the events. We extracted all predicates with an external reference to FrameNet from a set of 65,540 NAF files. This yielded a total of 3,612,511 predicates, 2,147 unique combinations of a lexical unit and a FrameNet frame and 428 unique frames. Note that a frame can be linked to multiple lexical units. In order to select the domain events and related frames, we annotated all predicates as being either contextual, grammatical, cognitive, perceptive or related to communication:

- Communication: all predicates related to communication, communicative gestures, motions and actions: (*remark, write, hush, forbid, howl, smile, censure, translate, nod, sing, wave*)
- Cognitive predicates: all predicates expressing states of mind and mental processes that may or may not induce actions: (*prefer, expect, worry, hope, deduce, classify, interpret, know, adopt, choose*);
- Perception: all predicates that denote physical experiences and sensations: (*feel, sense, hurt, observe, find, spy, taste*);
- Grammatical: all predicates that express aspect of another verb and light verbs: (*prevent, stop, take, remain, precede, engage, contain, imply*);
- Contextual predicates. All predicates that do not belong to one of the previous classes are contextual and potentially important for the domain: (*fluctuate, meet, break, melt, buy, accompany, refresh, sleep*).

All predicates belonging to Communication, Cognition and Perception were set aside as input for the attribution model, whereas the grammatical predicates are left out because they do not introduce events in a timeline but rather express properties of events. The contextual predicates then form the group of potential important events for the car domain. Table 1 shows the statistics on the extracted predicates related to a FrameNet frame in this data set. About 63% of all predicates found is not domain-specific; grammatical and communication related predicates make up the majority of the not domain-specific predicates with 27.73% and 22.65% respectively. The contextual predicates dominate the statistics, both in the number of unique frames (234), unique predicates (1306) and total predicate frequency (1,357,524).

For building the ontology, we defined the following structures:

Predicate type	Number of frames	Unique predicates	Total predicate frequency
Communication	88	396	818,291 (22.65%)
Cognitive	36	222	337,766 (9.34%)
Perception	9	50	96,821(2.68%)
Grammatical	78	173	1,002,109 (27.73%)
Contextual	234	1306	1,357,524 (37.57%)
Totals	445	2147	3,612,511

Table 1: Statistics on the predicates related to a FrameNet frame per predicate type

1. A hierarchy of events that are important for the domain and allow for inferencing;
2. A set of properties that allows for defining the most salient pre, post and during situation of the event;
3. A set of statements that defines the roles of the entities affected by the change.

## 4.2 The ESO Event Class Hierarchies and Mappings

To derive the first component for the ontology—a hierarchy of important domain events—we used the list of extracted contextual predicates with FrameNet mappings. As such, we started with 234 frames and 1,306 unique predicates with potentiality to be domain important. To scope this set, we put a threshold on the frames: all frames that were found only once, and in combination with a predicate with a frequency under 100, were not taken into account. As a result, 183 frames remained. Next, we experimented with three approaches to select a set of frames for modeling the event ontology.

In the first approach, we tried to select the most important frames by sorting on: a) the number of unique predicates that were found for this frame; b) the frequency of these predicates in our data; c) a combination of both. However, it turned out that these frequency statistics were not reliable enough. The number of predicates found for a frame depends solely on how many predicates have been defined in FrameNet. As such, it is not a strong pointer to dominant concepts. Additionally, some predicates are known to be high frequent for a frame, and this biases the frequency statistics we derived: a predicate such as *make* sometimes makes up half of the total predicate frequency of a frame.

In the second approach, we experimented with manually relating the frames from the car data back to the FrameNet to see if we could conceptually group and select concepts for the ontology. This turned out to be problematic as well, since

there is no full subclass hierarchy in FrameNet. Also, the frames themselves are organized by frame-semantic principles, meaning that some frames group lexical units that represent different concepts from a more ontological point of view. For instance `fn:Forming_Relationships` groups both *marry* and *divorce* and `fn:Change_position_on_a_scale` encompasses *increase* and *decrease*. As such, we decided to use FrameNet in a later stage of modeling the ontology.

In the third and final approach, we turned to another background model to organize the frames. For this, we have used the SUMO ontology<sup>2</sup> as it is freely available, well-documented, it has a good coverage and is mapped to English Wordnet. First, we made a selection of the 183 frames based on their expected importance for the domain: frames such as `Cooking_creation`, `Ingest_substance` and `Location_of_light` were left out. This resulted in 92 frames with the potential to be domain specific. The workflow for defining the hierarchy of dynamic event classes is as follows:

1. The initial and unstructured set of 92 frames was mapped manually to SUMO classes in order to organize the frames. All frames that expressed static events were set aside.
2. From this mapping, we selected four top nodes in SUMO that represented the main conceptual clusters for the frames expressing dynamic events: `Motion`, `InternalChange`, `ChangeOfPossession` and `IntentionalProcess`. In this step, we also started to group similar frames into one class. For instance, the main difference between the frames `Departing` and `Quitting_a_place` is a specification of the entity that moves. For our purposes, this level of granularity is not necessary. As such, both frames have been defined as corresponding to the ESO class `Departing`.
3. Next, we checked the SUMO class hierarchy of `Motion`, `InternalChange`, `ChangeOfPossession` and `IntentionalProcess` to select additional classes that may be of importance for the car domain, such as `Investing` and `Importing`.
4. We defined four hierarchies consisting of ESO classes with a mapping to SUMO and FrameNet and potential ESO classes with only a SUMO mapping.
5. To increase the coverage, we mapped back from these ESO classes to FrameNet frames. For this, we used the existing frame-to-frame relations in FrameNet. These additional were either a) found in the car data, but previously ruled out by the thresholds or b) not found in the car data but a frame for the ESO

---

<sup>2</sup>[www.ontologyportal.org](http://www.ontologyportal.org)



class does exist in FrameNet. In some cases, frames were found for which we had no SUMO-based ESO class. In those cases, a new ESO subclass was defined. Also, for some SUMO-based ESO classes no corresponding frame could be found. These classes were kept in the ontology nevertheless as placeholder for future extensions. As such, we have ESO classes with mappings to both FrameNet and SUMO, ESO classes with only a mapping to FrameNet, and ESO classes with only a mapping to SUMO. Furthermore, to keep the hierarchy clean, we opted to use single inheritance only for all event classes in the ontology.

For the static event classes that were set aside earlier, we performed the same workflow. However, the static event classes are represented as a flat hierarchy in the ontology since inferencing over these classes is not deemed to be useful here.

#### **4.2.1 Updates ESO Version 2**

In version 2 of the ESO ontology, the class hierarchy has been updated and extended. The following changes were made:

- New classes: Increasing, Decreasing, BeginningARelationship, EndingARelationship, BeingInUse, BeingDamaged, StartingAnActivity, StoppingAnActivity and HavingAValue.
- Deleted classes: ChangeOfLeadership (merged with Replacing), OrganizationalEvent (non-functional class), ChangeOfRelationship (split into BeginningARelationship and EndingARelationship), SocialInteraction (non-functional class), Constructing (merged with Creating) and Manufacturing (also merged with Creating).
- Hierarchy change: Meeting was replaced from subclass of DynamicEvent to subclass of StaticEvent.
- Label change: InEmployment has changed into BeingInEmployment to prevent confusion with the labels of some properties related to this class.

The motivation for the hierarchy changes is based on the observation that some important concepts were missing in ESO. Also, for some classes such as `eso:Constructing` and `eso:Manufacturing` the distinction between the two classes and their superclass was not clear, as such, all three classes have been merged into one (`eso:Creating`). Additionally, some classes have been split to enable proper modeling of the pre and post situation assertions such as `eso:ChangeOfRelationship` for which we now have two new classes (`eso:BeginningARelationship`, `eso:EndingARelationship`)

and `eso:QuantityChange` that has two new subclasses (`eso:Increasing` and `eso:Decreasing`). Classes such as `eso:OrganizationalEvent` have been removed as its only function was to serve as an intermediate class in the hierarchy. Further, we added a few new static events that could capture the explicit mentions that will also be of inferred by the reasoner as a pre or post situations pertaining to a dynamic event. All new classes have mappings to at least one FrameNet frame; SUMO mappings were added where possible.

In ESO version 2, we also changed the mapping properties to the external resources FrameNet and SUMO as was explained in section 3.4. This is motivated by the observation that some of the FrameNet frames should be considered as e.g. more general than our ESO classes; we specified this by SKOS-like mappings. For instance, in the new ESO the class `eso:QuantityChange` has been split into `eso:Increasing` and `eso:Decreasing`. For each class we specified a `correspondToFrameNetFrame_broadMatch` to e.g. `fn:Change_position_on_a_scale`. This frame associates predicates expressing both increases and decreases into one frame. With the new mapping property we specify that a subset of predicates associated to this frame will pertain to `eso:Increasing` and the other set to `eso:Decreasing`. The SKOS-like mappings will be replaced by proper SKOS mappings as soon as ESO is published as Linked Open Data. In addition to `ESO.owl`, manual mappings to Princeton Wordnet 3.0 have been created for all lexical units in a FrameNet frame associated to ESO. Also, for each lexical unit the relation to the ESO class is specified. For instance, the lexical unit `'increase.v'` belongs to the frame `fn:Cause_change_position_on_a_scale`; we mapped this lexical unit manually to two synsets and we specified that this lexical unit belongs conceptually to `eso:Increasing`. As such, we specified which lexical units from `fn:Change_position_on_a_scale` belong to `eso:Increasing` and which ones to `eso:Decreasing`. This information is vital for a) updating the PredicateMatrix with the new ESO classes and roles, b) ensuring that the correct assertions in ESO are triggered. In total, 1614 lexical units from FrameNet have been mapped, covering 1918 Wordnet synsets. The mappings have been kept outside ESO in order not to overburden the ontology; the file itself can be downloaded from the NewsReader website.

### 4.3 Properties and Roles for Defining the Situation Assertions

The second and third component of the ontology consists of properties and roles which are used for defining the assertions of the pre, post and during situations. All properties are hand-build, based on the shared semantics of the predicates related to a FrameNet frame and ESO class. The ESO roles define what entities are affected by a change and serve as the domain and range of the properties. The majority of the ESO roles is mapped to a selection of FrameNet Frame Entities (FEs);

these were selected manually from the FrameNet frames that correspond to an ESO class. This implies that not all FrameEntities of a frame are mapped to ESO but only those that play a role in the assertions. For instance, the FEs `fn:Self_mover` and `fn:Theme` are mapped to `eso:translocation-theme`, while the FEs 'Speed' and 'Manner' from the same frame are not used in ESO as they are of no importance for our assertions. An important modeling decision is that we define all assertions at the highest possible level in the ontology. This way, all subclasses will inherit the same assertions and roles which reduces redundancy. As such, many ESO roles have mappings to FEs that are aggregated from all mappings from ESO classes to FrameNet frame in a given subhierarchy. This is especially the case for the subhierarchies `eso:Translocation` and `eso:ChangeOfPossession`. These aggregated mappings have been checked carefully on conflicting roles. Another notable modeling choice here, is that the assertion properties for static event classes are partially shared with the assertion properties of the dynamic event classes. This is illustrated in Figure 2. Here, the same properties (`employedAt` and `employs`) are used in the pre situation assertion for the dynamic event class `LeavingAnOrganization`, in the post situation assertion for the dynamic event class `JoiningAnOrganization` and in the during situation assertion of the static event class `BeingInEmployment`. As a result, the relation between the inferred situation of a dynamic event and the explicit mention of some state by a static event becomes explicit. Modeling the properties this way facilitates querying for chains of related changes and states in the KnowledgeStore.

#### 4.3.1 Updates Version 2

An important update in ESO version 2 with respect to the previous version is the addition of many new assertions and thus properties and roles. This was motivated by the observation that the assertions were not expressive enough to capture important information for the financial-economic domain. Also, the modeling of scalar values turned out to be too basic. The meta model of the ontology has been adapted to allow for asserting the scalar and relative values as explained in section 3.3. Another modification in the new version of ESO is the addition of during situation assertions for a selection of dynamic events. In the previous version, during situation assertions were only applied for static event classes. This addition enables us to capture information that only holds during the time span of the dynamic event, which is especially important for `FinancialTransaction` and all its subclasses such as `eso:Buying` where the value of some exchanged entity only holds during the transaction. Here, we illustrate the expressivity of the new situation assertions in a non-formal way by means of three classes in ESO: `eso:Increasing` (Figure 3), `eso:JoiningAnOrganization` (Figure 4) and `eso:Buying` (Figure 5). For more exam-

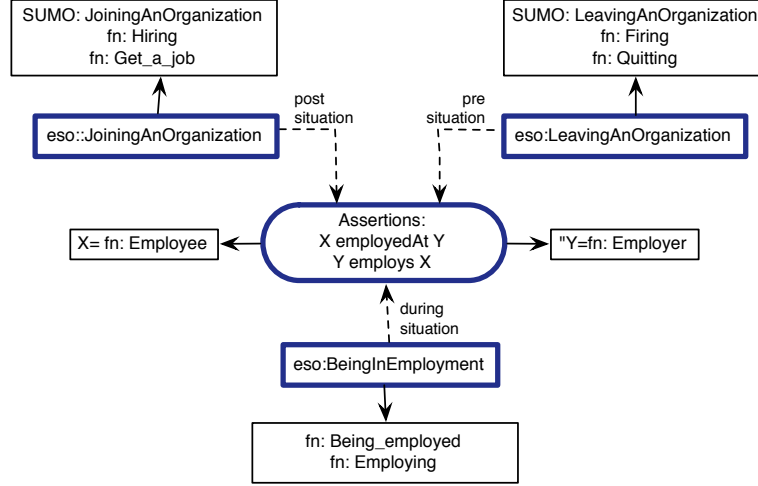


Figure 2: The shared assertion properties of a static and a dynamic event

ples of all the properties, roles, assertions and instantiations in ESO, we refer to the Appendix.

The assertions for the class `eso:Increasing` are exemplary for how scalar values are modeled in ESO. Scalar values are also defined for the classes `eso:Decreasing`, `eso:ChangingShape` and `eso:Damaging`, amongst others. Here, the pre and post situations of `eso:Increasing` allow to define that something has increased in relation to the state before the event, even if the actual attribute is not known. If we find an instance of the ESO role that models the attribute (here: `eso:quantity-attribute`), the role is instantiated as is shown in the first example sentence (*price*). If the attribute remains implicit, we create a blank node that allows us to still capture the relative values as is shown for the second example sentence. For the attribute, we define that it has a relative value ‘-’, or ‘minus’ before the event and ‘+’ or ‘more’ after the event. Additionally, we defined properties that define the actual value (`hasValue`) and the ratio of the increase (`hasRelativeIncrease`). Note that the property ‘`hasValue`’ is also used for the static event class `eso:HavingAValue` where static values of attributes are captured. In the case of e.g. `eso:Injuring`, `eso:Damaging` and `eso:Attacking`, we also use ‘+’ and ‘-’ as the range of the property `hasRelativeValue`. Since we do not formally define the semantics of the encodings, these symbols can be understood here as ‘better state’ and ‘worse state’ respectively.

Next, we illustrate the expressivity of the properties and assertions for the class `eso:JoiningAnOrganization` (Figure 4). The assertions of this class show the additional information that can be captured with respect to the previous version of

-Increasing      subclassOf: QuantityChange  
 "The subclass of InternalChange where some physical quantity or value is increased."

Role mappings:

quantity-item: fn: Item, fn: Possession, fn: Set  
 quantity-attribute: fn: Attribute, fn: Dimension  
 quantity-ratio: fn: Size\_change, fn: Difference  
 quantity-value\_1: fn: Initial\_value, fn: Initial\_number, fn: Initial\_size, fn: Value\_1  
 quantity-value\_2: fn: Final\_value, fn: Final\_number, fn: Value\_2, fn: Result\_size

Assertions:

**pre situation**

quantity-item	hasAttribute	quantity-attribute
quantity-attribute	hasRelativeValue	-
quantity-attribute	hasValue	quantity-value_1

**post situation**

quantity-item	hasAttribute	quantity-attribute (optional blank node)
quantity-attribute	hasRelativeValue	+
quantity-attribute	hasValue	quantity-value_2
quantity-item	hasRelativeIncrease	quantity-ratio

**EXAMPLES:**

"Apple raised the price of the Iphone from 500 to 600 dollar."

pre situation	Iphone	hasAttribute	price
	price	hasRelativeValue	-
	price	hasValue	500
post situation	Iphone	hasAttribute	price
	price	hasRelativeValue	+
	price	hasValue	600

"Ford increased the production with 2%."

pre situation	production	hasAttribute	:XYQ899
	:XYQ899	hasRelativeValue	-
post situation	production	hasAttribute	:XYQ899
	:XYQ899	hasRelativeValue	+
	production	hasRelativeIncrease	2%

Figure 3: eso:Increasing: assertions and assertion instantiation

```

-JoiningAnOrganization subclassOf: IntentionalEvent
"The subclass of IntentionalEvent where someone starts working as an employee for some organization."

Role mappings:
employment-employee: fn:Employee
employment-employer: fn:Employer
employment-function: fn:Position
employment-value: fn:Compensation
employment-task: fn:Task
employment-attribute: -

Assertions:
pre situation
employment-employee      notEmployedAt      employment-employer

post situation
employment-employee      employedAt      employment-employer
employment-employee      isEmployed      true
employment-employee      hasFunction      employment-function
employment-employee      hasTask      employment-task
employment-employee      hasAttribute      employment-attribute
employment-attribute      hasValue      employment-value

EXAMPLES:

"Ford hired Mary as their new CEO for 100.000 euro."

pre situation      Mary      notEmployedAt      Ford
post situation      Mary      isEmployed      true
                    Mary      employedAt      Ford
                    Mary      hasFunction      new CEO
                    Mary      hasAttribute      :BDA174
                    :BDA174      hasValue      100.000

```

Figure 4: eso:JoiningAnOrganization: assertions and assertion instantiation

ESO. Initially, we defined a basic assertions for each situation: before the event, the employee is not employed for the employer and after the event the employer is employed at the employer. In the new ESO, nothing has changed in the pre situation of this class as no other statements than can be defined that will always hold (someone may have had the same function or task at another employer). However, for the post situation, we defined additional assertions that capture important information for the domain, e.g. that someone has some function and a task after the event. Also, we modeled the value that is associated with the employment such as the value of e.g. the salary or the hiring fee. Since this value is not a direct property of the employee, we modeled this with an attribute and an optional blank node as was also shown for eso:Increasing. Note that not all assertions will always fire; if no instance can be found for the role employment-task, the assertion rule will be skipped by the reasoner.

Next, we show the new during situation assertions by means of the dynamic event class eso:Buying (Figure 5). This example shows how the during situation assertion is applied to capture the value of the item. As such we also have de-

```

-Buying    subclassOf: FinancialTransaction
           The subclass of FinancialTransaction where some entity changes of ownership in exchange for money.

Inherited role mappings:
possession-owner_1: fn:Supplier, fn:Exporter, fn:Donor, fn:Victim, fn:Source,
                    fn:Lender, fn:Exporting_area, fn:Sender, fn:Seller
possession-owner_2: fn:Perpetrator, fn:Importing_area, fn:Importer, fn:Lessee,
                    fn:Buyer, fn:Recipient, fn:Borrower, fn:Agent
possession-theme:   fn:Theme, fn:Goods, fn:Possession
possession-financial-asset: fn:Money

Inherited assertions:

pre situation
possession-owner_1    notHasInPossession    possession-financial-asset
possession-owner_1    hasInPossession        possession-theme
possession-owner_2    hasInPossession        possession-financial-asset
possession-owner_2    notHasInPossession     possession-theme

post situation
possession-owner_1    hasInPossession        possession-financial-asset
possession-owner_1    notHasInPossession     possession-theme
possession-owner_2    notHasInPossession     possession-financial-asset
possession-owner_2    hasInPossession        possession-theme

during situation
possession-theme      hasValue                possession-value

EXAMPLE:

"John bought the car for 600 dollars from Mary ."

pre situation  Mary    notHasPossession    600 dollars
               Mary    hasInPossession     the car
               John    hasInPossession     600 dollars
               John    notHasInPossession   the car

post situation  Mary    hasInPossession     600 dollars
               Mary    notHasInPossession   the car
               John    notHasInPossession   600 dollars
               John    hasInPossession     the car

during situation  the car  hasValue         10 dollar

```

Figure 5: eso:Buying: assertions and assertion instantiation

financed that the economic value of the item is defined by the transaction and is not intrinsic to the item itself. Additionally, this example shows that this class inherits its roles and assertions from its superclasses `eso:FinancialTransaction` and `eso:ChangeOfPossession`. A number of mappings from ESO roles to FrameNet FEs may seem odd for this class (e.g. `fn:Victim` and `fn:Borrower`), however these FEs are not associated in FrameNet to the frame `fn:Commerce_buy`. Therefore, they will not be found by the NWR Semantic Role Labeling module in relation to predicates that express a buying event in text.

#### 4.4 Content Overview of the Event and Situation Ontology

In this section, we first give statistics of the content of the current version and previous version of ESO; next we show some more detailed overviews of the content. As was reported in the previous sections, ESO has had some major extensions to

Component	ESO Version 1	ESO Version 2
Number of event classes	59	63
DynamicEvent classes	50	50
StaticEvent classes	9	13
SUMO class mappings	46	46
FrameNet Frame mappings	94	103
Situation rules	30	50
Situation rule assertions	35	123
Pre situation rule assertions	15	41
Post situation rule assertions	11	52
During situation rule assertions	9	30
Properties	24	58
Unary properties	4	11
Binary properties	20	47
ESO roles	33	65
Mappings to FrameNet FEs	58	131

Table 2: Overview of the content in ESO Version 1 and in ESO Version 2

comply with the needs of the end users in the financial-economic domain. The volume of this extension is shown in table 2.

Most notable in this table is the increase in the number of assertions (from 35 to 123), properties (from 24 to 58) and ESO roles (from 33 to 65) with respect to the previous version. Additionally, many more mappings to FrameNet FEs were created (from 58 to 131) to capture the entities affected by a change.

ESO Version 2 comprises 63 event classes divided over dynamic event classes (50) and static event classes (13). The dynamic event class hierarchy consists of four major nodes: `eso:ChangeOfPossession`, `eso:Motion`, `eso:InternalChange` and `eso:IntentionalEvent`. An overview of the dynamic event hierarchy is presented in Figure 6. The static events are modeled into a flat hierarchy; an overview of the static events classes is presented in Figure 7. Finally, an overview of the mappings from ESO classes to FrameNet Frames and SUMO classes is given in Table 3. The default mapping here is `closeMatch`; the `relatedMatch` and `broadMatch` mappings have been shortened here in `'rm:'` and `'bm:'` respectively.

ESO Class	FrameNet frame	SUMO class
Arriving	Arriving Vehicle_landing	Arriving
Attacking	Attack	ViolentContest
BeginningARelationship	bm:Forming_relationships	-



BeingAtAPlace	Residence Presence Temporary_stay Being_located	-
BeingDamaged	bm:Being_operational	-
BeingEmployed	Being_employed Employing	-
BeingInAPersonalRelationship	Personal_relationship	-
BeingInExistence	Existence	-
BeingInUse	bm:Being_operational Using_resource Using	-
BeingLeader	Leadership	-
BeingOperational	Being_operational	-
Borrowing	Borrowing	Borrowing
Buying	Commerce_buy	Buying
ChangeOfPossession	rm:Transfer	ChangeOfPossession
ChangingShape	Manipulate_into_shape Reshaping	ShapeChange
Collaboration	Collaboration	Cooperation
Creating	Building Intentionally_create Creating Manufacturing	Constructing Making Creation Manufacture
Damaging	Damaging Render_nonfunctional	Damaging
Decreasing	bm:Change_of_quantity_of_possession bm:Cause_change_of_position_on_a_scale bm:Change_position_on_a_scale bm:Proliferating_in_number bm:Expansion bm:Cause_expansion	Decreasing
Destroying	Cause_to_fragment Destroying	Destruction
Distribution	Dispersal	-
DynamicEvent	-	-
EndingARelationship	bm: Forming_relationships	-
Escaping	Escaping Fleeing	Escaping
Exporting	Exporting	Exporting
FinancialTransaction	Commercial_transaction	FinancialTransaction
Getting	Receiving Getting	Getting
Giving	Sending Giving	Giving

HavingAValue	Supply	
HavingInPossession	Amounting_to	-
	Possession	-
	Retaining	
Importing	Importing	rm:Exporting
Increasing	bm:Change_position_on_a_scale	Increasing
	Cause_proliferation_in_number	
	bm:Change_of_quantity_of_possession	
	bm:Expansion	
	bm:Proliferating_in_number	
	bm:Cause_expansion	
	bm:Cause_change_of_position_on_a_scale	
Injuring	Experience_bodily_harm	Injuring
	Cause_harm	
Installing	Installing	Installing
IntentionalEvent	Intentionally_act	IntentionalProcess
InternalChange		InternalChange
Investing	-	Investing
JoiningAnOrganization	Hiring	bm:JoiningAnOrganization
	Get_a_job	
Killing	Execution	Killing
	Killing	
Leaving	Vehicle_departure_initial_state	Leaving
	Departing	
	Setting_out	
	Quitting_a_place	
LeavingAnOrganization	Firing	TerminatingEmployment
	Quitting	
Lending	Lending	Lending
Meeting	Assemble	Meeting
	Come_together	
	Social_event	
Merging	Amalgamation	Combining
	Cause_to_amalgamate	
Motion	Motion	Motion
Paying	Commerce_pay	Payment
Placing	Placing	Putting
QuantityChange		QuantityChange
Removing	Removing	Removing
Renting	Renting	Renting
RentingOut	Renting_out	-
Replacing	Replacing	Substituting
	Take_place_of	
	Change_of_leadership	
Selling	Commerce_sell	Selling

Separating	Becoming_separated	Separating
StartingAnActivity	Separating	-
StaticEvent	Activity_start	-
Stealing	State	Stealing
StoppingAnActivity	Theft	-
Taking	Activity_stop	UnilateralGetting
Translocation	Taking	Translocation
	Self_motion	
	Cotheme	
	Traversing	
	Use_vehicle	
	Intentional_traversing	
	Ride_vehicle	
	Travel	
	Operate_vehicle	
	Cause_motion	
Transportation	Bringing	Transportation
	Delivery	
Working	Working_a_post	-
	Work	

---

Table 3: Mappings from ESO classes to FrameNet frames and SUMO including a shortened specification of the mapping.

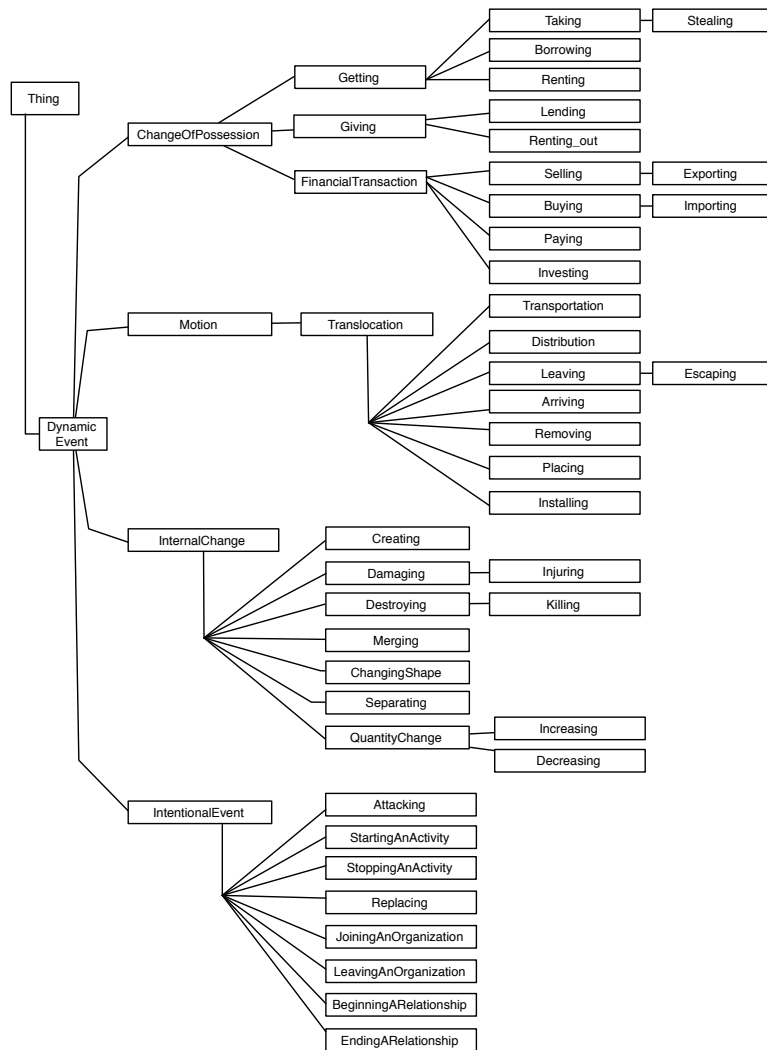


Figure 6: Overview of the dynamic event class hierarchy

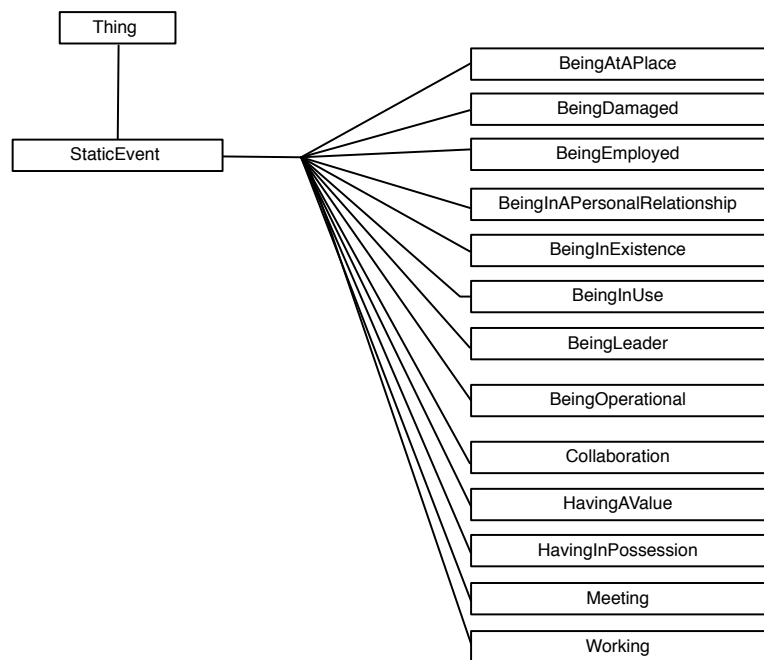


Figure 7: Overview of the dynamic event class hierarchy