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

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*

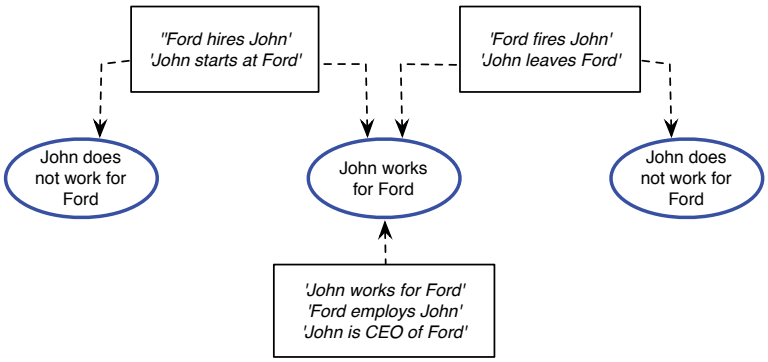


Figure 1: A chain of static and dynamic events and their implied situation

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 ;
        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 form 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 an 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[1] 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

[1] 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: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[2] 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

[2] `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, al 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 then 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 Entitites (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-

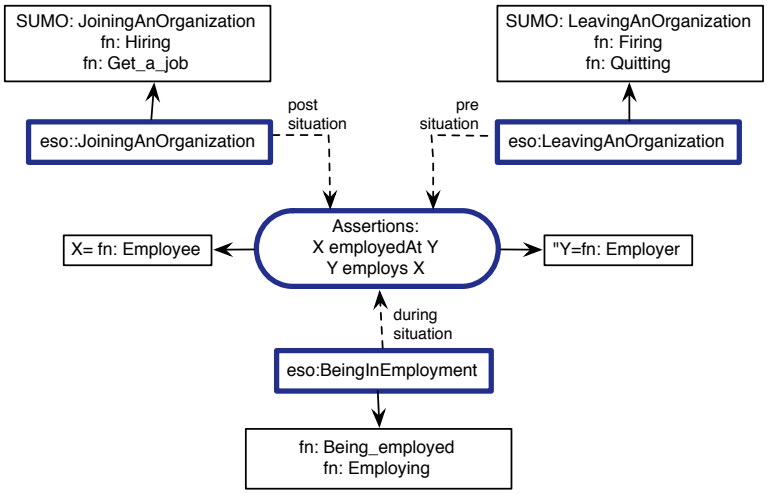


Figure 2: The shared assertion properties of a static and a dynamic event

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 were 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:

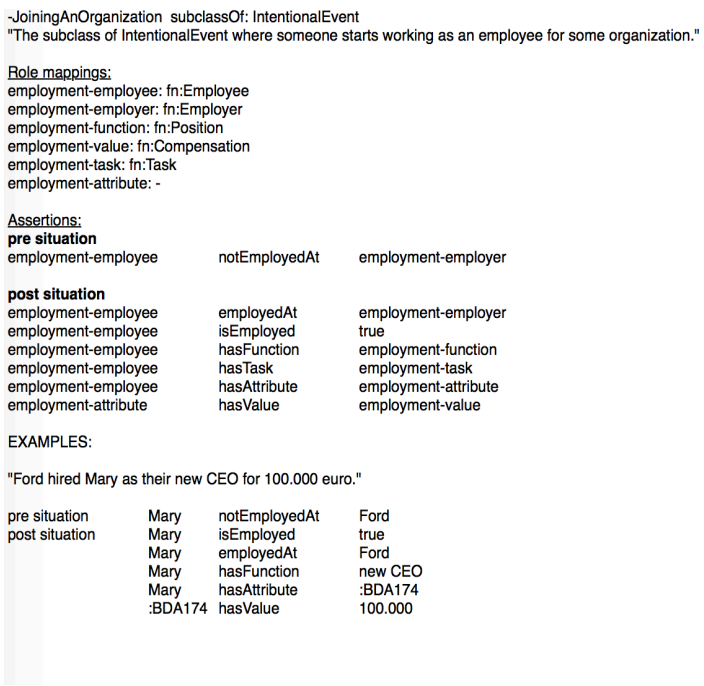
"Apple raised the price of the Iphone from 500 to 600 dollar."

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-

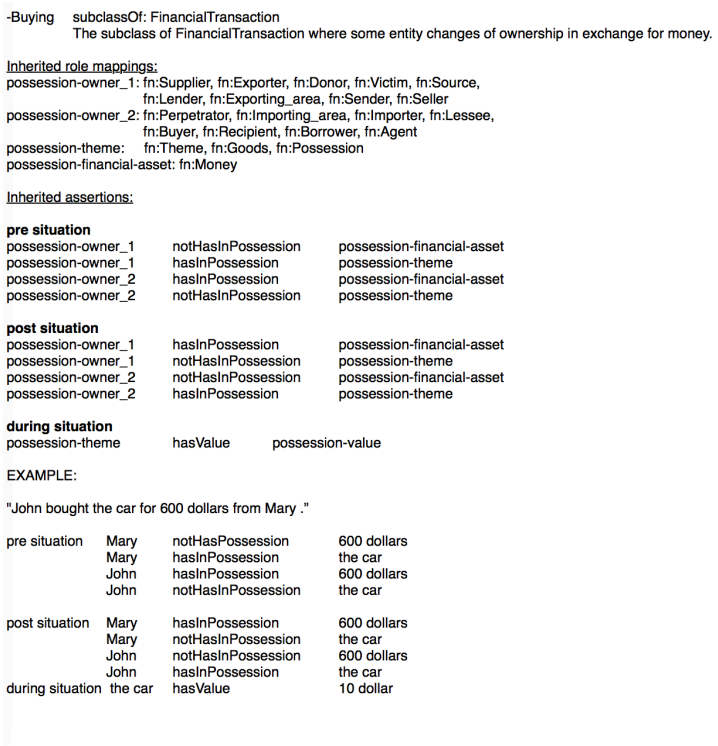


Figure 5: eso:Buying: assertions and assertion instantiation

fined 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	ShapeChange
	Reshaping	
Collaboration	Collaboration	Cooperation
Creating	Building	Constructing
	Intentionally_create	Making
	Creating	Creation
	Manufacturing	Manufacture
Damaging	Damaging	Damaging
	Render_nonfunctional	
Decreasing	bm:Change_of_quantity_of_possession	Decreasing
	bm:Cause_change_of_position_on_a_scale	
	bm:Change_position_on_a_scale	
	bm:Proliferating_in_number	
	bm:Expansion	
	bm:Cause_expansion	
Destroying	Cause_to_fragment	Destruction
	Destroying	
Distribution	Dispersal	-
DynamicEvent	-	-
EndingARelationship	bm: Forming_relationships	-
Escaping	Escaping	Escaping
	Fleeing	
Exporting	Exporting	Exporting
FinancialTransaction	Commercial_transaction	FinancialTransaction
Getting	Receiving	Getting
	Getting	
Giving	Sending	Giving
	Giving	

	Supply	
HavingAValue	Amounting_to	-
HavingInPossession	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
	Separating	
StartingAnActivity	Activity_start	-
StaticEvent	State	-
Stealing	Theft	Stealing
StoppingAnActivity	Activity_stop	-
Taking	Taking	UnilateralGetting
Translocation	Self_motion	Translocation
	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.

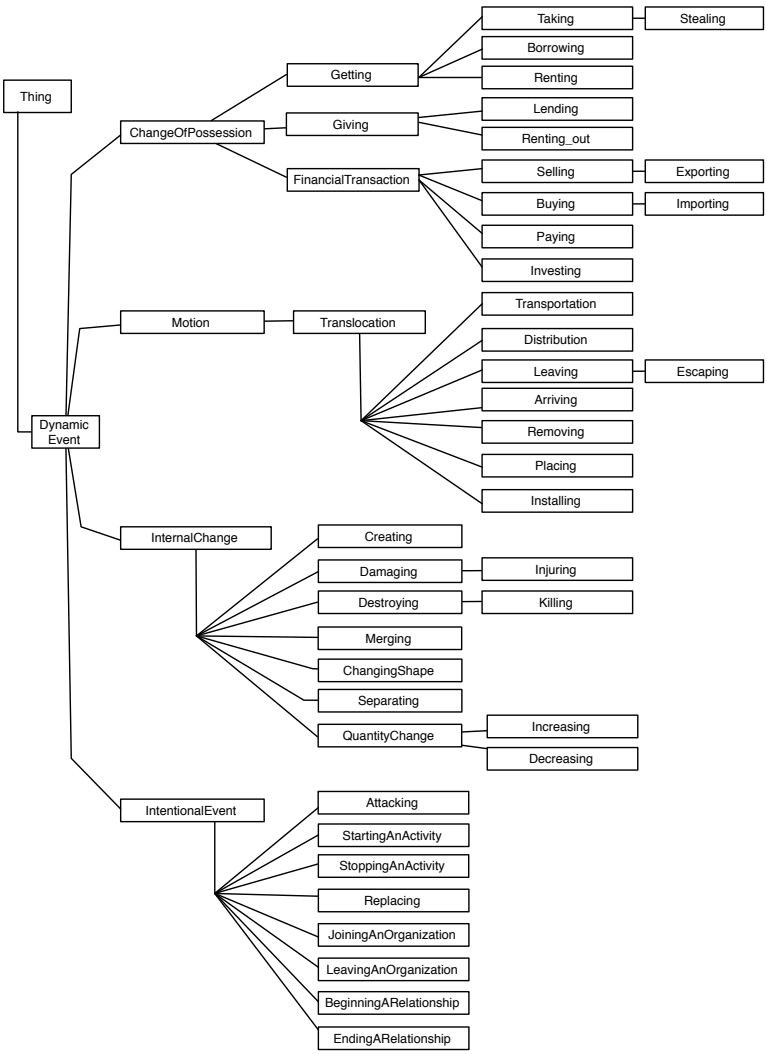


Figure 6: Overview of the dynamic event class hierarchy

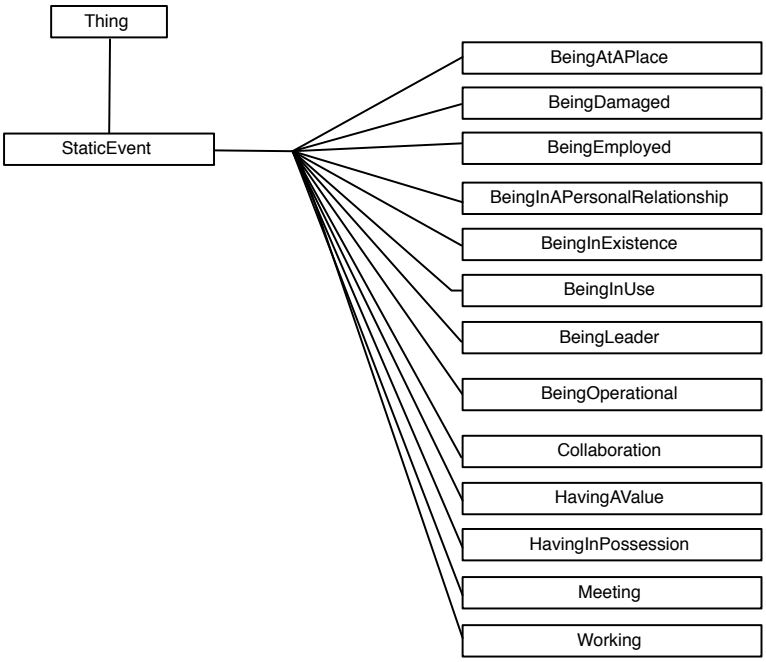


Figure 7: Overview of the dynamic event class hierarchy

APPENDIX

ESO Classes: Definitions, Class mappings, Role Mappings, Assertions and Examples of the Instantiation of the Assertions.

This file provides a human readable version of the Event and Situation Ontology Version 2, developed for the NewsReader project (www.newsreader-project.eu).

All classes are in alphabetical order. For each class we provide:
-the subclass relation
-the class definition
-the mappings from ESO classes to FrameNet and SUMO (as available online at June 20, 2015)
-the mappings from ESO roles to FrameNet Frame Elements
-the assertions for each class defining the situation that holds before, after and/or during the event (in a non-formal transcription).
-examples that show what the ESO class assertions can infer from a sentence annotated with FrameNet-based SRL.

Date: June 24th 2015

For questions and remarks, please contact:
r.h.segers@vu.nl

ESO CLASSES IN ALPHABETICAL ORDER:

-**Arriving** subclassOf:Translocation
"The subclass of Translocation where someone or something arrives at a location."

Class mappings:
closeMatch: fn:Arriving
closeMatch: fn:Vehicle_landing
closeMatch: sumo:Arriving

For the roles and assertions and, see: Translocation.

EXAMPLES:

"Mary approached the White House with a grim face."

pre situation	Mary	notAtPlace	the White House
post situation	Mary	atPlace	the White House

"Mary arrived in Washington from Dulles National Airport."

pre situation	Mary	atPlace	Dulles National Airport
	Mary	notAtPlace	Washington
post situation	Mary	atPlace	Washington
	Mary	notAtPlace	Dulles National Airport

-**Attacking** subclassOf: IntentionalEvent
"The subclass of IntentionalEvent where someone or something is assaulted with the intention to cause some harm."

Class mappings:
closeMatch: fn:Attack
closeMatch: sumo:ViolentContest

Role mappings:
damaging-undergoer: fn: Object, fn:Victim, fn: Experiencer, fn:Body_part,

fn: Patient, fn: Artifact
damaging-state-1: - (blank node)
damaging-state-2: - (blank node)
damaging-damage: -
activity: -

Assertions:

pre situation:	damaging-undergoer	inState	damaging-state_1
	damaging-state-1	hasRelativeValue	"+"
post situation:	damaging-undergoer	inState	damaging-state_2
	damaging-state-2	hasRelativeValue	"-"
	damaging-undergoer	isDamaged	true
	damaging-undergoer	hasDamage	damaging-damage
	damaging-damage	hasNegativeEffectOn	activity

Note that the last two assertions will not be instantiated as no FrameNet roles exist for the ESO roles damaging-damage and activity.
Note that damaging-state-1 and damaging-state-2 are modeled with an existential restriction that allows to create a blank node in the named graph.

EXAMPLES:

"Marie attacked John with a knife."

pre situation	John	inState	:xyz123
	:xyz123	hasRelativeValue	+
post situation	John	inState	:xyz124
	:xyz124	hasRelativeValue	-
	John	isDamaged	true

"The army bombed the power plant."

pre situation	the power plant	inState	:xyz125
	xyz125	hasRelativeValue	+
post situation	the power plant	inState	:xyz126
	:xyz126	hasRelativeValue	-
	the power plant	isDamaged	true

"The hurricane struck West-Virginia."

pre situation	West-Virginina	inState	:abc123
	:abc123	hasRelativeValue	+
post situation	West-Virginia	inState	:abc124
	:abc124	hasRelativeValue	-
	West-Virginia	isDamaged	true

-BeginningARelationship subclassOf: IntentionalEvent
"The subclass of IntentionalEvent were people start or form a personal relationship with each other".

Class mappings:
broadMatch: fn:Forming_relationships

Role mappings:
relationship-partner-1: fn:Partner_1
relationship-partner-2: fn:Partner_2
relationship-partners: fn:Partner_1, fn:Partner_2, fn:Partners

Assertions:

pre situation	relationship-partner-1 relationship-partners	notInRelationshipWith inRelationship	relationship-partner_2 false
post situation	relationship-partner-1 relationship-partners	inRelationshipWith inRelationship	relationship-partner_2 true

EXAMPLES:

"John married Mary in 2011."

pre situation	John John, Mary	notInRelationshipWith inRelationship	Mary false
post situation	John John, Mary	inRelationshipWith inRelationship	Mary true

"The secret wedding of John and Mary!"

pre situation	John and Mary	inRelationship	false
post situation	John and Mary	inRelationship	true

"John married again in 2014."

pre situation	John	inRelationship	false
post situation	John	inRelationship	true

-BeingAtAPlace subclassOf: StaticEvent
"Static event where some entity is at a location."

Class mappings:
closeMatch: fn:Residence
closeMatch: fn:Presence
closeMatch: fn:Temporary_stay
closeMatch: fn:Being_located

Role mappings:
atPlace-theme: fn:Theme, fn:Resident, fn:Entity, fn:Guest.
atPlace-location: fn:Location

Assertions:

during situation:	atPlace-theme	atPlace	atPlace-location

EXAMPLES:

"Marie stayed at the Hilton Hotel."

during situation	Marie	atPlace	Hilton Hotel

"Oil reservoirs are present in Rotterdam."

during situation	oil reservoirs	atPlace	Rotterdam

"John lives in Amsterdam."

during situation	John	atPlace	Amsterdam

"John is the first resident at King's Landing."

during situation John atPlace King's Landing

-BeingDamaged subclassOf: StaticEvent
"Static event where some entity is in a damaged state."

Class mappings:
broadMatch: fn:Being_operational

Role mappings:
damaging_undergoer: fn:Object, fn:Victim, fn: Experiencer, fn:Body_part,
fn: Patient, fn: Artifact.
damaging-damage: -
activity: -

Assertions:

during-situation:	damaging-undergoer	isDamaged	true
	damaging-undergoer	hasDamage	damaging-damage
	damaging-damage	hasNegativeEffectOn	activity

Note that the last two assertions will not be instantiated as no FrameNet roles exist for the ESO roles damaging-damage and activity.

EXAMPLE:

"The suspension of this car is broken."

during-situation

the suspension of this car	isDamaged	true
(this car	hasDamage	broken suspension)
(broken suspension	hasNegativeEffectOn	operating)

-BeingEmployed subclassOf: StaticEvent
"Static event where someone is working in a position and is compensated for her work by some form of payment."

Class mappings:
closeMatch: fn:Being_employed
closeMatch: fn:Employing

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:

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

Note that employment-attribute is modeled with an existential restriction that allows to create a blank node in the named graph.

EXAMPLES:

"Ford employed Marie as CFO."

during situation	Marie	employedAt	Ford
	Marie	hasFunction	CFO
	Marie	isEmployed	true

"Marie works as CFO for 2000 dollar a month."

during situation	Marie	hasFunction	CFO
	Marie	hasAttribute	:xyz667
	:xyz667	hasValue	2000 dollar
	Marie	isEmployed	true

"Marie is employed at Ford to handle the severe financial issues."

during situation	Marie	employedAt	Ford
	Marie	hasTask	to handle the severe financial issues
	Marie	isEmployed	true

-BeingInAPersonalRelationship subclassOf:StaticEvent

"The subclass of StaticEvent where persons are in some personal relationship."

Class mappings:
closeMatch: fn:Personal_relationship

Role mappings:
relationship-partner-1: fn:partner_1
relationship-partner-2: fn:partner_2
relationship-partners: fn:partners, fn: partner_1, fn: partner_2

Assertions:

during situation	relationship-partner-1	inRelationshipWith	relationship-partner_2
during situation	relationship-partners	inRelationship	true

EXAMPLES:

"John dates Marie."

during-situation	John	inRelationshipWith	Marie
	John, Marie	inRelationship	true

"John is married to Marie."

during situation	John	inRelationshipWith	Marie
	John, Marie	inRelationship	true

-BeingInExistence subclassOf: StaticEvent

“Static event where some entity exists.”

Class mappings:
closeMatch: fn:Existence

Role mappings:
exist-theme: fn:Entity

Assertions:

during situation	exist-theme	exist	true

EXAMPLES:

"Cars with a Wankel engine still exist."

during situation	cars with a Wankel engine	exist	true

"There were human settlements near the volcano."

during situation	human settlements near the volcano	exist	true

-BeingInUse subclassOf StaticEvent

"The static event class where something is in use by an agent (in some particular role or for some purpose)."

Class mappings:
closeMatch: fn:Using
closeMatch: fn:UsingResource
broadMatch: fn:BeingOperational

Role mappings:
inuse-entity-1: fn:Agent
inuse-entity-2 fn:Instrument, fn:Resource, fn:Object
inuse-function: fn:Role
inuse-purpose: fn:Purpose

Assertions:

during situation	inuse-entity_1	uses	inuse-entity_2
	inuse-entity_2	hasFunction	inuse-function
	inuse-entity_2	hasPurpose	inuse-purpose
	inuse-entity_2	inFunction	true

"Ford uses codename X for operations in India."

during situation	Ford	uses	codename X
	codename X	hasPurpose	operations in India
	codename X	inFunction	true

"Ford used codename X name as cover."

during situation	Ford	uses	operational name
	codename X	hasFunction	cover
	codename X	inFunction	true

"Mary used her Peugeot 205 to drive to work."

during situation	Mary	uses	her Peugeot 205
	her Peugeot 205	hasPurpose	drive to work
	her Peugeot 205	inFunction	true

"The system works."

during situation	the system	inFunction	true

-BeingLeader subclassOf: StaticEvent

“StaticEvent where someone is leader of some group of persons or organization.”

Class mappings:
closeMatch: fn:Leadership

Role mappings:
leader-entity: fn:Leader

leader-governed-entity: fn:Governed
leader-function: fn:Role

Assertions:

during situation:	leader-entity	isLeader	true
	leader-entity	isLeaderOf	leader-governed_entity
	leader-entity	hasFunction	leader-function

EXAMPLES:

"John chairs the committee"

during situation	John	isLeader	true
	John	isLeaderOf	the committee

"John ruled over Apple as a king"

during situation	John	isLeader	true
	John	isLeaderOf	Apple
	John	hasFunction	king

"Ford is setting up an operation which is headed by Mary as general manager"

during situation	Mary	isLeader	true
	Mary	hasFunction	general manager

"John is chairman of the committee."

during situation	John	isLeader	true
	John	isLeaderOf	the committee

-BeingOperational subclassOf: StaticEvent
Static event where some device is in function.

Class mappings:
closeMatch: fn:Being-operational

Role mappings:
operational-theme: fn:Object

Assertions:

during situation	operational-theme	inFunction	true

EXAMPLES:

"The new welding power supply works."

during situation	the new welding power supply	inFunction	true

"The new welding power supply is functional."

during situation	the new welding power supply	inFunction	true

-Borrowing subclassOf: Getting
"The subclass of Getting where a person gets something in possession for some period of time after which the item should be given back."

Class mappings:
closeMatch: fn:Borrowing
closeMatch: fn:Borrowing

For the roles and assertions, see: ChangeOfPossession.

EXAMPLE:

"Mary borrowed the car from John"

pre situation	John	hasInPossession	the car
	Marie	notHasInPossession	the car
post situation	John	notHasInPossession	the car
	Marie	hasInPossession	the car

-Buying subclassOf: FinancialTransaction
The subclass of FinancialTransaction where some entity changes of ownership in exchange for money. Note that the buyer is not necessarily the new owner of the entity.

Class mappings:
closeMatch: fn:Commerce_buy
closeMatch: sumo:Buying

For the roles and assertions, see: ChangeOfPossession.

EXAMPLES:

"John bought the flowers for 10 dollar."

pre situation	John	hasInPossession	10 dollar
	John	notHasPossession	the flowers
post situation	John	hasInPossession	the flowers
	John	notHasInPossession	10 dollar
during situation	the flowers	hasValue	10 dollar

"John bought the flowers from Mary."

pre situation	John	notHasInPossession	the flowers
	Mary	hasInPossession	the flowers
post situation	John	hasInPossession	the flowers
	Mary	notHasInPossession	the flowers

"John bought the flowers for Mary."

pre situation	John	notHasInPossession	flowers
	Mary	notHasInPossession	flowers
post situation	John	hasInPossession	flowers
	Mary	hasInPossession	flowers*

**Note that Mary is the 'Recipient' in FrameNet. While this FrameNet role is important for some subclasses of eso: ChangeOfPossession, for eso:Buying, this role is less prominent. However, the roles and assertions for this sub hierarchy are modeled at the highest possible level in the ontology (ChangeOfPossession) and are inherited by e.g. Buying.*
As a result, in some cases the assertions of the post situation of Buying can generate a questionable statement.

-ChangeOfPossession subclassOf: DynamicEvent
"The subclass of DynamicEvent where some entity changes possession. Note that this often but not necessarily implies a change of location of the entity."

Class mappings:
relatedMatch: fn:Transfer
closeMatch: sumo: ChangeOfPossession

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

Assertions:

pre situation	possession-owner_1	hasInPossession	possession-theme
	possession-owner_2	notHasInPossession	possession-theme
post situation	possession-owner_1	notHasInPossession	possession-theme
	possession-owner_2	hasInPossession	possession-theme

EXAMPLES:

"Marie stole the car keys from John"

pre situation	John	hasInPossession	car keys
	Marie	notHasInPossession	car keys
post situation	John	notHasInPossession	car keys
	Marie	hasInPossession	car keys

"Ford exported 3000 cars to India last month"

pre situation	Ford	hasInPossession	3000 cars
	India	notHasInPossession	3000 cars
post situation	Ford	notHasInPossession	3000 cars
	India	hasInPossession	3000 cars

-ChangingShape subclassOf:InternalChange
"The subclass of InternalChange where the shape of an entity is changed."

Class mappings:
closeMatch: fn:Manipulate_into_shape
closeMatch: fn:Reshaping
closeMatch: sumo:ShapeChange

Role mappings:
changingshape-entity: fn:Undergoer, fn:Theme
changingshape-initialshape: -
changingshape-finalshape: fn:Configuration, fn:Resultant_configuration, fn:Result

Assertions:

pre situation	changingshape-entity	inState	changingshape-initialshape
	changingshape-entity	notInState	changingshape-finalshape
post situation	changingshape-entity	inState	changingshape-finalshape
	changingshape-entity	notInState	changingshape-initialshape

Note that changingshape-initialshape and changingshape-finalshape are modeled with an existential restriction that allows to create a blank node in the named graph.

EXAMPLES:

"John moulded the paste into a ball."

pre situation	the paste	inState	:xyz130
	the paste	notInState	ball
post situation	the paste	inState	ball
	the paste	notInState	:xyz130

"John folded the paper."

pre situation	the paper	inState	:xyz134
	the paper	notInState	:abc123
post situation	the paper	inState	:abx123
	the paper	notInState	:xyz134

-Collaboration subclassOf: StaticEvent

"Static event where people work together for some period of time."

Class mappings:
closeMatch: fn:Collaboration
closeMatch: sumo:Cooperation

Role mappings:
collaboration-partner-1: fn:Partner_1
collaboration-partner-2: fn:Partner_2
collaboration-partners: fn:Partner_1, fn:Partner_2, fn:Partners
collaboration-project: fn:Undertaking

Assertions:

during situation	collaboration-partner_1	collaboratesWith	collaboration-partner_2
	collaboration-partners	inCollaboration	true
	collaboration-partners	hasProject	collaboration-project

EXAMPLES:

"John collaborates with Mary on a book."

during situation	John	collaboratesWith	Mary
	John, Mary	hasProject	a book
	John, Mary	inCollaboration	true

"The left wing parties are conspiring to impeach the president."

during situation	the left wing parties	hasProject	to impeach the president
	the left wing parties	inCollaboration	true

-Creating subclassOf: InternalChange

"The subclass of InternalChange where something is made, created, build, constructed, etc."

Class mappings:
closeMatch: fn:Building
closeMatch: fn:Intentionally_create
closeMatch: fn:Creating
closeMatch: fn:Manufacturing
closeMatch: sumo:Constructing
closeMatch: sumo:Creation
closeMatch: sumo:Manufacture
closeMatch: sumo:Making

Role mappings:
creating-theme: fn: Product, fn:Created_entity

Assertions:

pre situation	creating-theme	exist	false
post situation	creating-theme	exist	true

EXAMPLES:

"The company was founded in 1981."

pre situation	the company	exist	false
post situation	the company	exist	true

"Rover assembled 22.000 Morris Minis from 1986 onwards."

pre situation	22.000 Morris Minis	exist	false
post situation	22.000 Morris Minis	exist	true

"Mary builds a new house on the hill."

pre situation	a new house on the hill	exist	false
post situation	a new house on the hill	exist	true

-Damaging subclassOf: InternalChange

"The subclass of InternalChange where something is damaged."

Class mappings:
closeMatch: fn:Render_nonfunctional, fn:Damaging
closeMatch: sumo:Damaging

Role mappings:
damaging-undergoer: fn: Object, fn:Victim, fn: Experiencer, fn:Body_part,
fn: Patient, fn: Artifact
damaging-state-1: -
damaging-state-2: -
damaging-damage: -
activity: -

Assertions:

pre situation:	damaging-undergoer	inState	damaging-state_1
	damaging-state_1	hasRelativeValue	"+"
post situation:	damaging-undergoer	inState	damaging-state_2
	damaging-state_2	hasRelativeValue	"-"
	damaging-undergoer	isDamaged	true
	damaging-undergoer	hasDamage	damaging-damage
	damaging-damage	hasNegativeEffectOn	activity

Note that the last two assertions will not be instantiated as no FrameNet roles exist for the ESO roles 'damaging-damage' and 'activity'.
Note that damaging-state1 and damaging-state-2 have an existential restriction that allows to create a blank node in the named graph.

EXAMPLES:

"Marie dented the car"

pre situation	car	inState	:abc123
	:abc123	hasRelativeValue	+
post situation	car	inState	:xyz556
	:xyz556	hasRelativeValue	-
	car	isDamaged	true

"John incapacitated the aircraft."

pre situation	the aircraft	inState	:efg123
	:efg123	hasRelativeValue	+
post situation	the aircraft	inState	:efg345
	:efg345	hasRelativeValue	-
	the aircraft	isDamaged	true

-Decreasing subclassOf: QuantityChange

"The subclass of QuantityChange where some physical quantity or value is decreased."

Class mappings:
broadMatch: fn:Change_of_quantity_of_possession
broadMatch: fn:Cause_change_of_position_on_a_scale
broadMatch: fn:Change_position_on_a_scale
broadMatch: fn:Proliferating_in_number
broadMatch: fn: Expansion
broadMatch: fn: Cause_expansion
closeMatch: sumo:Decreasing

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
	quantity-attribute	hasRelativeValue	-
	quantity-attribute	hasValue	quantity-value_2
	quantity-item	hasRelativeDecrease	quantity-ratio

Note that quantity-attribute is modeled with an existential restriction that allows to create a blank node in the named graph.

EXAMPLES:

"Ford decreased the production with 2%."

pre situation	production	hasAttribute	:qwe123
	:qwe123	hasRelativeValue	+
post situation	production	hasAttribute	:qwe123
	:qwe123	hasRelativeValue	-
	production	hasRelativeDecrease	2%

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

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

"The profit shrunk dramatically."

pre situation	profit	hasAttribute	:bnm234
	:bnm234	hasRelativeValue	+

post situation	profit	hasAttribute	:bnm234
	:bnm234	hasRelativeValue	-

-Destroying subclassOf: InternalChange

"The subclass of InternalChange where something gets destroyed."

Class mappings:
closeMatch: fn:Cause_to_fragment
closeMatch: fn:Destroying
closeMatch: sumo:Destruction

Role mappings:
destroying-theme: fn:Whole_patient, fn:Executed, fn:Undergoer, fn:Victim

Assertions:

pre situation:	destroying-theme	exist	true
post situation:	destroying-theme	exist	false

EXAMPLES:

"They demolished the Vauxhall factory."

pre situation	the Vauxhall factory	exist	true
post situation	the Vauxhall factory	exist	false

"Mary tore up the license agreement."

pre situation	the license agreement	exist	true
post situation	the license agreement	exist	false

-Distribution subclassOf: Translocation

"The subclass of Translocation where someone or something translocates a physical object from one location to a bigger area."

Class mappings:
closeMatch: fn:Dispersal

For the assertions and role mappings, see: Translocation.

EXAMPLES

"Bats spread the disease across Sudan."

pre situation	the disease	notAtPlace	Sudan
post situation	the disease	atPlace	Sudan

"The engines were mainly distributed in Korea."

pre situation	the engines	notAtPlace	Korea
post situation	the engines	atPlace	Korea

-DynamicEvent This class is the root of the dynamic event class hierarchy.
(no mappings, no assertions)

-EndingARelationship subclassOf: IntentionalEvent

"The subclass of IntentionalEvent were people end a relationship with each other."

Class mappings:

broadMatch: fn:Forming_relationships

Role mappings:
relationship-partner-1: fn:Partner_1
relationship-partner-2: fn:Partner_2
relationship-partners: fn:Partner_1, fn:Partner_2, fn:Partners

pre situation	relationship-partner_1	inRelationshipWith	relationship-partner_2
	relationship-partners	inRelationship	true
post situation	relationship-partner_1	notInRelationshipWith	relationship-partner_2
	relationship-partners	inRelationship	false

EXAMPLES

"Mary split up with John."

pre situation	John	inRelationshipWith	Mary
	John, Mary	inRelationship	true
post situation	John	notInRelationshipWith	Mary
	John, Mary	inRelationship	false

"John divorced in 2013."

pre situation	John	inRelationship	true
post situation	John	inRelationship	false

"The divorce of John and Mary is on the front page of all tabloids!"

pre situation	John and Mary	inRelationship	false
post situation	John and Mary	inRelationship	true

-Escaping subclassOf: Leaving
"The subclass of Leaving where a person leaves an unwanted location."

Class mappings
closeMatch: fn:Escaping
closeMatch: fn:Fleeing
closeMatch: sumo:Escaping

For the assertions and role mappings, see: Translocation.

EXAMPLES:

"John escaped from Alcatraz."

pre situation	John	atPlace	Alcatraz
post situation	John	notAtPlace	Alcatraz

"John fled to the United States."

pre situation	John	notAtPlace	the United States
post situation	John	atPlace	the United States

-Exporting subclassOf: Selling
"The subclass of Selling where goods are exported to another nation
in exchange for money."

Class mappings:

closeMatch: fn:Exporting
closeMatch: sumo:Exporting

For the assertions and role mappings, see: FinancialTransaction

EXAMPLES:

"Ford exported 10.000 cars to India."

pre situation	Ford	hasInPossession	10.000 cars
	India	notHasInPossession	10.000 cars
post situation	Ford	notHasInPossession	10.000 cars
	India	hasInPossession	10.000 cars

"Car exportation to India."

pre situation	India	notHasInPossession	car
post situation	India	hasInPossession	car

-FinancialTransaction: subclassOf: ChangeOfPossession
"The subclass ofChangeOfPossession where some item changes of ownership in exchange for money."

Class mappings:
closeMatch: fn:CommercialTransaction
closeMatch: sumo:FinancialTransaction

Role mappings:
possession-financial-asset: fn: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

Assertions:

pre situation	possession-owner_1	notHasInPossession	poss.-financial-asset
	possession-owner_2	hasInPossession	poss.-financial-asset
post situation	possession-owner_1	hasInPossession	poss.-financial-asset
	possession-owner_2	notHasInPossession	poss.-financial-asset
during situation	possession-theme	hasValue	possession-value

Inherited assertions from ChangeOfPossession:

pre situation	possession-owner_1	hasInPossession	possession-theme
	possession-owner_2	notHasInPossession	possession-theme
post situation	possession-owner_1	notHasInPossession	possession-theme
	possession-owner_2	hasInPossession	possession-theme

EXAMPLES:

"Marie bought the car from John for 600 dollars"

pre situation	Marie	hasInPossession	600 dollar
	Marie	notHasInPossession	the car
	John	hasInPossession	the car
	John	notHasInPossession	600 dollar

post situation	Marie	hasInPossession	the car
	Marie	notHasInPossession	600 dollar
	John	hasInPossession	600 dollar
	John	notHasInPossession	the car
during situation	the car	hasValue	600 dollar

"Mary paid 600 dollar for the car."

pre situation	Mary	notHasInPossession	the car
	Mary	hasInPossession	600 dollar
post situation			
	Mary	hasInPossession	the car
	Mary	notHasInPossession	600 dollar
during situation	the car	hasValue	600 dollar

-Getting subclassOf: ChangeOfPossession

"The subclass of ChangeOfPossession where a person gets or receives some item."

Class mappings:
closeMatch: fn:Receiving
closeMatch: fn:Getting
closeMatch: sumo:Getting

For the assertions and role mappings, see: ChangeOfPossession.

EXAMPLES:

"Mary received the strategic report from John."

pre situation	John	hasInPossession	the strategic report
	Mary	notHasInPossession	the strategic report
post situation	John	notHasInPossession	the strategic report
	Mary	hasInPossession	the strategic report

"Mary gained the respect of her staff."

pre situation	Mary	notHasInPossession	the respect of her staff
post situation	Mary	hasInPossession	the respect of her staff

"Ford secured the European market."

pre situation	Ford	notHasInPossession	the European market
post situation	Ford	hasInPossession	the European market

-Giving subclassOf: ChangeOfPossession

The subclass of ChangeOfPossession where a person gives something to someone else.

Class mappings:
closeMatch: fn:Sending
closeMatch: fn:Giving
closeMatch: fn:Supply
closeMatch: sumo:Giving

For the assertions and role mappings, see: ChangeOfPossession.

EXAMPLES:

"Mary gave John a nice bouquet."

pre situation	Mary	hasInPossession	a nice bouquet
	John	notHasInPossession	a nice bouquet
post situation	Mary	notHasInPossession	a nice bouquet
	John	hasInPossession	a nice bouquet

"The US shipped tents and food to Indonesia after the tsunami."

pre situation	the US	hasInPossession	tents and food
	Indonesia	notHasInPossession	tents and food
post situation	the US	notHasInPossession	tents and food
	Indonesia	hasInPossession	tents and food

-HavingAValue subclassOf: StaticEvent

"The subclass of StaticEvent where something is having some value."

Class mappings:
closeMatch: fn:Amounting_to.

Role mappings:
value-attribute: fn:Attribute
value: fn:Value

Assertions:

during situation	value-attribute	hasValue	value

EXAMPLE:

"Maries income amounted to 100.000 euro a year."

during situation	Maries income	hasValue	100.000 euro

-HavingInPossession subclassOf: StaticEvent

"Static event where someone has something in possession."

Class mappings:
closeMatch: fn:Possession
closeMatch: fn:Retaining

Role mappings:
possession-owner: fn:Agent, fn:Owner
possession-theme: fn:Theme, fn:Goods, fn:Possession

Assertions:

during situation	possession-owner	hasInPossession	possession-theme

EXAMPLES:

"Tata Steel has 10.000 employees."

during situation	Tata Steel	hasInPossession	10.000 employees

"Mary owns a house in Spain."

during situation	Mary	hasInPossession	a house in Spain

"The US retains political support from Europe."

during situation	The US	hasInPossession	political support from Europe

"Mary kept her old wedding gown."

during situation	Mary	hasInPossession	her old wedding gown

-Importing: subclassOf: Buying
"The subclass of Buying where goods are imported from some country in exchange for money."

Class mappings:
closeMatch: fn:Importing
relatedMatch: sumo:Exporting

For assertions and role mappings, see: FinancialTransaction.

EXAMPLES:

"Canada imported 45.000 cars from Europe last year."

pre situation	Europe	hasInPossession	45.000 cars
	Canada	notHasInPossession	45.000 cars
post situation	Europe	notHasInPossession	45.000 cars
	Canada	hasInPossession	45.000 cars

"Iran's import of nuclear material was monitored."

pre situation	Iran	notHasInPossession	nuclear material
post situation	Iran	hasInPossession	nuclear material

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

Class mappings:
broadMatch: fn:Change_of_quantity_of_possession
broadMatch: fn:Cause_change_of_position_on_a_scale
broadMatch: fn:Change_position_on_a_scale
broadMatch: fn:Proliferating_in_number
broadMatch: fn: Expansion
broadMatch: fn: Cause_expansion
closeMatch: fn:Cause_proliferation_in_number
closeMatch: sumo:Increasing

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

quantity-attribute	hasRelativeValue	+
quantity-attribute	hasValue	quantity-value_2
quantity-item	hasRelativeIncrease	quantity-ratio

Note that quantity-attribute is modeled with an existential restriction that allows to create a blank node in the named graph.

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	:asd123
	:asd123	hasRelativeValue	-
post situation	production	hasAttribute	:asd123
	:asd123	hasRelativeValue	+
	production	hasRelativeIncrease	2%

"Their debt tripled in nine years."

pre situation	their debt	hasRelativeValue	-
post situation	their debt	hasRelativeValue	+

"He widened his eyes."

pre situation	his eyes	hasAttribute	:zxc234
	:zxc234	hasRelativeValue	-
post situation	his eyes	hasAttribute	:zxc234
	:zxc234	hasRelativeValue	+

"The balloon expanded with 2 centimetres".

pre situation	the balloon	hasAttribute	:abc123
	:abc123	hasRelativeValue	-
post situation	the balloon	hasAttribute	:abc123
	:abc123	hasRelativeValue	+
	the balloon	hasRelativeIncrease	2 centimetres

-Injuring subclassOf: Damaging

"The subclass of Damaging where someone gets injured (mentally and/or physically)."

Class mappings:
closeMatch: fn:Cause_harm
closeMatch: fn:Experience_bodily_harm
closeMatch: sumo:Injuring

For the assertions and role mappings, see: Damaging.

EXAMPLES:

"Marie wounded John."

pre situation	John	inState	:qwe556
	qwe556	hasRelativeValue	+
post situation	John	inState	:zxc678
	:zxc678	hasRelativeValue	-
post situation:	John	isDamaged	true

"John broke his leg after falling off the stage"

pre situation	John, his leg	inState	:abc123
	:abc123	hasRelativeValue	+
post situation	John, his leg	inState	:abc124
	:abc124	hasRelativeValue	-
post situation:	John, his leg	isDamaged	true

"Mary broke his leg with her bare hands!"

pre situation	his leg	inState	:jkl234
	:jkl234	hasRelativeValue	+
post situation	his leg	inState	:asd345
	:asd345	hasRelativeValue	-
post situation:	his leg	isDamaged	true

-Installing subclassOf: Placing

"The subclass of Placing where some entity is put in a new and fixed location, e.g. the installation of fixtures."

Class mappings:
closeMatch: fn:Installing
closeMatch: sumo:Installing

For the assertions and role mappings, see: Translocation.

EXAMPLES:

"Mary installed a new engine in her Land Rover Defender."

pre situation	a new engine	notAtPlace	Land Rover Defender
post situation	a new engine	atPlace	Land Rover Defender

"John confirmed the installation of cameras in the offices."

pre situation	cameras	notAtPlace	in the offices
post situation	cameras	atPlace	in the offices

-IntentionalEvent subclassOf:DynamicEvent

"The subclass of DynamicEvent where some event is carried out by some cognitive agent(s) and with some specific purpose."

Class mappings:
closeMatch: fn:Intentionally_act
sumo: IntentionalProcess

No assertions are defined for this class.

-InternalChange subclassOf: DynamicEvent

"The subclass of DynamicEvent where some internal quality of an item changes."

Class mappings:
closeMatch: sumo:InternalChange

No assertions are defined for this class.

-Investing subclassOf: FinancialTransaction
The subclass ofFinancialTransaction where a person or company invests some asset in either another or its own company with the prospect of some future profit.

Class mappings:
closeMatch: sumo:Investing

For the assertions, see: FinancialTransaction.

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

Class mappings:
closeMatch: fn:Hiring,
closeMatch: fn:Get_a_job
broadMatch: sumo:JoiningAnOrganization

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

Note that employment-attribute is modeled with an existential restriction that allows to create a blank node in the named graph.

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	:abc124
	:abc124	hasValue	100.000 euro

"John was hired to clean the house."

pre situation	-		
post situation	John	isEmployed	true

	John	hasTask	to clean the house

"John signed on with Marie to clean her house."

pre situation	John	notEmployedAt	Marie
post situation	John	isEmployed	true
	John	employedAt	Marie
	John	hasTask	to clean her house

-Killing subclassOf: Destroying

"The subclass of Destroying where animate beings are killed."

Class mappings:
closeMatch: fn:Execution
closeMatch: fn:Killing
closeMatch: sumo:Killing

For assertions and role mappings, see: Destroying.

EXAMPLES:

"Mary was executed by three men in black ties."

pre situation	Mary	exist	true
post situation	Mary	exist	false

"Low levels of oxygen asphyxiated the fish in John's pond."

pre situation	the fish in John's pond	exist	true
post situation	the fish in John's pond	exist	false

-Leaving subclassOf:Translocation

"The subclass of Translocation where someone or something leaves a location."

Class mappings:
closeMatch: fn:Vehicle_departure_initial_state
closeMatch: fn:Departing
closeMatch: fn:Setting_out
closeMatch: fn:Quitting_a_place
closeMatch: sumo:Leaving.

For the assertions and role mappings, see: Translocation.

EXAMPLES:

"John set out from Lake Louise in a canoe."

pre situation	John	atPlace	Lake Louise
post situation	John	notAtPlace	Lake Louise

"John left for Lake Michigan."

pre situation	John	notAtPlace	Lake Michigan
post situation	John	atPlace	Lake Michigan*

**Note that Johns arrival at Lake Michigan is not certain.*

-LeavingAnOrganization subclassOf: IntentionalEvent

"The subclass of IntentionalEvent where a person stops working as an employee for an organization."

Class mappings:
closeMatch: fn:Quitting,
closeMatch: fn:Firing
closeMatch: sumo:TerminatingEmployment

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

Assertions:

pre situation	employment-employee	employedAt	employment-employer
	employment-employee	isEmployed	true
	employment-employee	hasFunction	employment-function
	employment-employee	hasTask	employment-task
post situation	employment-employee	notEmployedAt	employment-employer

EXAMPLES:

"Ford fired Mary as their CEO."

pre situation	Mary	employedAt	Ford
	Mary	isEmployed	true
	Mary	hasFunction	CEO
post situation	Mary	notEmployedAt	Ford

"John was fired from cleaning the house."

pre situation	John	isEmployed	true
	John	hasTask	cleaning the house
post situation	-		

"John left Ford."

pre situation	John	employedAt	Ford
post situation	John	notEmployedAt	Ford

-Lending subclassOf:Giving

"The subclass of Giving where a person gives something in possession for some period of time after which the item should be given back."

Class mappings:
closeMatch: fn:Lending
closeMatch: sumo:Lending

For the assertions and role mappings, see: ChangeOfPossession.

EXAMPLE:

"Mary loaned her car to John."

pre situation	Mary	hasInPossession	her car
	John	notHasInPossession	her car

post situation	Mary	notHasInPossession	her car
	John	hasInPossession	her car

-Meeting subclassOf: StaticEvent

"The static event class where people meet each other, usually intentional and for some purpose."

Class mappings:
closeMatch: fn:Come_together
closeMatch: fn:Assemble
closeMatch: fn:Social_event
closeMatch: sumo:Meeting

Role mappings:
meeting-participant: Party_1, Party_2, fn:Attendee, fn:Host, fn:Individuals, fn:Group, fn:Configuration
meeting-place: fn:Place

Assertions:

during situation	meeting-participantatPlace	meeting-place
	meeting-participantinMeeting	true

EXAMPLES:

"The Republicans convened in New York to discuss the program."

during situation	the Republicans	atPlace	New York
	the Republicans	inMeeting	true

"John meets Marie in New York"

during situation	John	atPlace	New York
	Marie	atPlace	New York
	John, Marie	inMeeting	true

"The whole group attended the party"

during situation	the whole group	inMeeting	true

-Merging subclassOf: InternalChange

"The subclass of InternalChange where two entities are merged into a whole."

Class mappings:
closeMatch: fn:Amalgamation
closeMatch: fn:Cause_to_amalgamate
closeMatch: sumo:Combining

Role mappings:
merging-theme_1: fn:Part_1, fn:Parts
merging-theme_2: fn:Part_2
merging-theme_3: fn:Whole

Assertions:

pre situation	merging-theme_1	exist	true
	merging-theme_2	exist	true
	merging-theme_3	exist	false
post situation:	merging-theme_1	exist	false
	merging-theme_2	exist	false
	merging-theme_3	exist	true

EXAMPLES:

"In 1980, EBC merged with KPN into KPN-BC."

pre situation	EBC	exist	true
	KPN	exist	true
	KPN-BC	exist	false
post situation	EBC	exist	false
	KPN	exist	false
	KPN-BC	exist	true

"John blended the herbs and the eggs."

pre situation	the herbs and the eggs	exist	true
post situation	the herbs and the eggs	exist	false

-Motion subclassOf: DynamicEvent
"The subclass of DynamicEvent where some entity moves."

Class mappings:
closeMatch: fn:Motion
closeMatch: sumo:Motion

No assertions are defined for this class.

-Paying subclassOf: FinancialTransaction
"The subclass of FinancialTransaction where some financial asset is given in exchange for some item or in discharge of a debt."

Class mappings:
closeMatch: fn:Commerce_pay

For the assertions and role mappings, see: FinancialTransaction.

EXAMPLES:

"Ford paid Chrysler 40.000 dollar for John's idea."

pre situation	Ford	notHasInPossession	John's idea
	Chrysler	hasInPossession	John's idea
	Ford	hasInPossession	40.000 dollar
	Chrysler	notHasInPossession	40.000 dollar
post situation	Ford	hasInPossession	John's idea
	Chrysler	notHasInPossession	John's idea
	Ford	notHasInPossession	40.000 dollar
	Chrysler	hasInPossession	40.000 dollar
during situation	John's idea	hasValue	40.000 dollar

"Mary paid the bill."

pre situation	Mary	hasInPossession	the bill
post situation	Mary	notHasInPossession	the bill

-Placing subclassOf:Translocation

“The subclass of Translocation where some entity is put in a new location.”

Class mappings:
closeMatch: fn:Placing
closeMatch: sumo:Putting

For the assertions and role mappings, see: Translocation.

EXAMPLES:

“While thinking of Mary, John put the flowers in a vase.”

pre situation	flowers	notAtPlace	in a vase
post situation	flowers	atPlace	in a vase

"Mary loaded all her belongings in the car."

pre situation	her belongings	notAtPlace	in the car
post situation	her belongings	atPlace	in the car

"The sea deposited dead fish on the beach."

pre situation	dead fish	notAtPlace	on the beach
post situation	dead fish	atPlace	on the beach

-QuantityChange subclassOf: InternalChange
“The subclass of InternalChange where some quantity is altered.”

Class mappings:
closeMatch: sumo: QuantityChange

No assertions are defined for this class.

-Removing subclassOf: Translocation
“The subclass of Translocation where some entity is taken away from its location.”

Class mappings:
closeMatch: fn:Removing
closeMatch: sumo:Removing

For the assertions and role mappings, see: Translocation.

EXAMPLES:

"John removed all the evidence from the archive."

pre situation	the evidence	atPlace	the archive
post situation	the evidence	notAtPlace	the archive

"Mary evacuated the employees from the burning factory."

pre situation	the employees	atPlace	the burning factory
post situation	the employees	notAtPlace	the burning factory

"The Maserati was unloaded from the Boeing 747."

pre situation	the Maserati	atPlace	the Boeing 747
post situation	the Maserati	notAtPlace	the Boeing 747

"John removed all his books."

pre situation	-		
post situation	-		

-Renting subclassOf: Getting

"The subclass of Getting where a person gets something in possession from someone else for some period in exchange for money."

Class mappings:
closeMatch: fn:Renting
closeMatch: sumo:Renting

For the assertions and role mappings, see: ChangeOfPossession.

EXAMPLES:

"John leased his Peugeot from ELB."

pre situation	John	notHasInPossession	his Peugeot
	ELB	hasInPossession	his Peugeot
post situation	John	hasInPossession	his Peugeot
	ELB	notHasInPossession	his Peugeot

"Mary rented a room from an old lady."

pre situation	Mary	notHasInPossession	a room
	an old lady	hasInPossession	a room
post situation	Mary	hasInPossession	a room
	an old lady	notHasInPossession	a room

-RentingOut subclassOf: Giving

"The subclass of Giving where a person gives something in possession for some period in exchange for money."

Class mappings:
closeMatch: fn:Renting_out

For the assertions and role mappings, see: ChangeOfPossession.

EXAMPLES:

"The old lady rented a room to Mary."

pre situation	Mary	notHasInPossession	a room
	an old lady	hasInPossession	a room
post situation	Mary	hasInPossession	a room
	an old lady	notHasInPossession	a room

"Mary rented the garage out."

pre situation	Mary	hasInPossession	the garage
post situation	Mary	notHasInPossession	the garage

-Replacing subclassOf: IntentionalEvent

"The subclass of IntentionalEvent were someone or something is replaced with someone or something else in a specific role or function."

Class mappings:
closeMatch: fn:Replacing
closeMatch: fn: Take_place_of
closeMatch: fn:Change_of_leadership
closeMatch: sumo:Substituting

Role mappings:
replacing-entity_1: fn:Old, fn:Old_order, fn:Old_leader
replacing-entity_2: fn:New, fn:New_leader
replacing-entity_3: fn:Agent
replacing-function: fn:Role, fn:Function

Assertions:

pre situation	replacing-entity_1	hasFunction	replacing-function
	replacing-entity_2	notHasFunction	replacing-function
	replacing-entity_1	inFunctionFor	replacing-entity_3
	replacing-entity_1	inFunction	true
	replacing-entity_2	inFunction	false
post situation	replacing-entity_1	notHasFunction	replacing-function
	replacing-entity_2	hasFunction	replacing-function
	replacing-entity_2	inFunctionFor	replacing-entity_3
	replacing-entity_1	inFunction	false
	replacing-entity_2	inFunction	true

EXAMPLES:

"Peter replaced Mary by John as CEO of Apple."

pre situation	Mary	hasFunction	CEO of Apple
	John	notHasFunction	CEO of Apple
	Mary	inFunctionFor	Peter
	Mary	inFunction	true
	John	inFunction	false
post situation	Mary	notHasFunction	CEO of Apple
	John	hasFunction	CEO of Apple
	John	inFunctionFor	Peter
	Mary	inFunction	false
	John	inFunction	true

"Mary replaced her Ford Taunus for a Peugeot 205."

pre situation	Ford Taunus	inFunctionFor	Mary
	Ford Taunus	inFunction	true
	Renault 205	inFunction	false
post situation	Peugeot 205	inFunctionFor	Mary
	Ford Taunus	inFunction	false
	Peugeot 205	inFunction	true

"Vinyl was replaced by the compact disc in the early eighties."

pre situation	vinyl	inFunction	true
	compact disc	inFunction	false
post situation	compact disc	inFunction	true
	vinyl	inFunction	false

"Amsterdam installed Mary as the new mayor."

pre situation	Mary	notHasFunction	mayor
	Mary	inFunction	false
post situation	Mary	hasFunction	mayor
	Mary	inFunctionFor	Amsterdam
	Mary	inFunction	true

"The rebellion against the Lannisters."

pre situation	Lannisters	inFunction	true
post situation	Lannisters	inFunction	false*

**Note that, due to the lexical units associated to a FrameNet frame, the triggered assertions can be too strong.*

-Selling subclassOf: FinancialTransaction

`'The subclass of FinancialTransaction where some entity changes of ownership in exchange for money."

Class mappings:
closeMatch: fn:Commerce_sell
closeMatch: sumo:Selling

For the assertions and role mappings, see: FinancialTransaction.

EXAMPLES:

"In 2013, Ford sold 10.000 cars."

pre situation	Ford	hasInPossession	10.000 cars
post situation	Ford	notHasInPossession	10.000 cars

"The Catholic church auctioned off 20 churches to project developers."

pre situation	Catholic church	hasInPossession	20 churches
	project developers	notHasInPossession	20 churches
post situation	Catholic church	notHasInPossession	20 churches
	project developers	hasInPossession	20 churches

"Mary sold the plot of land to John for 10.000 dollar."

pre situation	Mary	hasInPossession	the plot of land
	John	notHasInPossession	the plot of land
	Mary	notHasInPossession	10.000 dollar
	John	hasInPossession	10.000 dollar
post situation	Mary	notHasInPossession	the plot of land
	John	hasInPossession	the plot of land
	Mary	hasInPossession	10.000 dollar
	John	notHasInPossession	10.000 dollar
during situation	the plot of land	hasValue	10.000 dollar

-Separating subclassOf: InternalChange

"The subclass of InternalChange where some whole is split into parts."

Class mappings:
closeMatch: fn:Becoming_separated
closeMatch: fn:Separating
closeMatch: sumo:Separating

Role mappings:

separating-theme_1: fn:Part_1, fn:Parts
separating-theme_2: fn:Part_2
separating-theme_3: fn:Whole

Assertions:

pre situation	separating-theme_1	exist	false
	separating-theme_2	exist	false
	separating-theme_3	exist	true
post situation	separating-theme_1	exist	true
	separating-theme_2	exist	true
	separating-theme_3	exist	false

EXAMPLES:

"The machine split the water into hydrogen and oxygen."

pre situation	hydrogen and oxygen	exist	false
	water	exist	true
post situation	hydrogen and oxygen	exist	true
	water	exist	false

"Mary divided the pile of cutlery into groups of six."

pre situation	groups of six	exist	false
	pile of cutlery	exist	true
post situation	groups of six	exist	true
	pile of cutlery	exist	false

"The auctioneer separated the hatchbacks from the saloons.*"

pre situation	the hatchbacks	exist	false
	the saloons	exist	false
post situation	the hatchbacks	exist	true
	the hatchbacks	exist	true

**Note that separating-theme_3 (the whole collection of cars) remains implicit in this example.*

"The partition of Germany in 1945."

pre situation	Germany	exist	true
post situation	Germany	exist	false

-StartingAnActivity subclassOf: IntentionalEvent

"The subclass of IntentionalProcess where someone intentionally starts an activity."

Class mappings:
closeMatch: fn:Activity_start

Role mappings:
activity: fn:Activity
activity-agent: fn:Agent

Assertions:

pre situation	activity	exist	false
post situation	activity	exist	true
	activity-agent	involvedIn	activity

"Ford started the production of the Taunus in 1979."

pre situation	production of the Taunus	exist	false
post situation	production of the Taunus	exist	true
	Ford	involvedIn	production of the Taunus

"The government began protecting the peat bogs in Ost-Friesland."

pre situation	protecting the peat bogs in Ost-Friesland	exist	false
post situation	protecting the peat bogs in Ost-Friesland	exist	true
	the government	involvedIn	protecting the peat bogs in Ost-Friesland.

-StaticEvent StaticEvent is the top node of the static event class hierarchy.
"A StaticEvent is an entity which is associated with a period of time where a set of propositions is true."

Class mappings:
closeMatch: fn:State

No assertions are defined for this class.

-Stealing subclassOf: Taking
"The subclass of Taking where a person takes something without permission of the owner."

Class mappings:
closeMatch: fn:Theft
closeMatch: sumo:Stealing

For the assertions and class mappings, see: ChangeOfPossession.

EXAMPLES:

"John shoplifted a sweater from the department store."

pre situation	department store	hasInPossession	sweater
	John	notHasInPossession	sweater
post situation	department store	notHasInPossession	sweater
	John	hasInPossession	sweater

"Marie stole a sweater from John."

pre situation	John	hasInPossession	a sweater
	Marie	notHasInPossession	a sweater
post situation	John	notHasInPossession	a sweater
	Marie	hasInPossession	a sweater

"Massive theft of documents from the Stasi archives."

pre situation	Stasi archives	hasInPossession	documents
post situation	Stasi archives	notHasInPossession	documents

-StoppingAnActivity subclassOf:IntentionalEvent
"The subclass of IntentionalProcess where some agent intentionally stops an activity."

Class mappings:
closeMatch: fn:Activity_stop

Role mappings:
activity: fn:Activity
activity-agent: fn:Agent

Assertions:

pre situation	activity	exist	true
	activity-agent	involvedIn	activity
post-situation	activity	exist	false
	activity-agent	notInvolvedIn	activity

"Ford terminated the negotiations with Peugeot."

pre situation	negotiations with Peugeot	exist	true
	Ford	involvedIn	negotiations with Peugeot
post situation	negotiations with Peugeot	exist	false
	Ford	notInvolvedIn	negotiations with Peugeot

"John's treatment was discontinued."

pre situation	John's treatment	exist	true
post situation	John's treatment	exist	false

-Taking subclassOf: Getting

"The subclass of Getting where a person takes something without giving something in return."

Class mappings:
closeMatch: fn:Taking
closeMatch: sumo:UnilateralGetting

For the assertions and role mappings, see: ChangeOfPossession

EXAMPLES:

"The police seized financial documents from the private equity fund."

pre situation	the police	notHasInPossession	financial documents
	private equity fund	hasInPossession	financial documents
post situation	the police	hasInPossession	financial documents
	private equity fund	notHAsInPossession	financial documents

"Mary took a beer from the refrigerator."

pre situation	Mary	notHasInPossession	a beer
	the refrigerator	hasInPossession	a beer
post situation	Mary	hasInPossession	a beer
	the refrigerator	notHasInPossession	a beer

-Translocation subclassOf:Motion

"The subclass of Motion where physical objects or animate beings change from location."

Class mappings:
closeMatch: fn:Self_motion
closeMatch: fn:Cotheme
closeMatch: fn:Traversing
closeMatch: fn:Use_vehicle
closeMatch: fn:Intentional_traversing
closeMatch: fn:Ride_vehicle
closeMatch: fn:Travel
closeMatch: fn:Operate_vehicle
closeMatch: fn:Cause_motion
closeMatch: sumo:Translocation

Role mappings:
translocation-theme: fn:Self_mover, fn: Theme, fn:Driver, fn:Traveler, fn:Vehicle, fn:Escapee, fn:Cotheme, fn:Component, fn:Individuals.
translocation-source: fn:Source, fn: Undesirable_location
translocation-goal: fn:Goal, fn: Intended_goal, fn: Goal_area

Assertions:

pre situation:	translocation-theme	atPlace	translocation-source
	translocation-theme	notAtPlace	translocation-goal
post situation:	translocation-theme	atPlace	translocation-goal
	translocation-theme	notAtPlace	translocation-source

EXAMPLE:

"John drove from New York to Atlanta."

pre situation	John	atPlace	New York
	John	notAtPlace	Atlanta
post situation	John	atPlace	Atlanta
	John	notAtPlace	New York

-Transportation subclassOf:Transportation

"The subclass of Translocation where physical objects and animate beings together change from location and the physical object is not the means of translocation."

Class mappings:
closeMatch: fn:Bringing
closeMatch: fn:Delivery
closeMatch: sumo:Transportation

For the assertions and role mappings, see: Translocation

EXAMPLES:

"Mary brought her classic car from the US to England."

pre situation	her classic car	atPlace	US
	her classic car	notAtPlace	England
post situation	her classic car	atPlace	England
	her classic car	notAtPlace	US

"John flew Mary to the nearest hospital."

pre situation	Mary	notAtPlace	hospital
post situation	Mary	atPlace	hospital

"Russian gas deliveries to Europe."

pre situation	gas	atPlace	Russia
	gas	botAtPlace	Russia
post situation	gas	notAtPlace	Russia
	gas	atPlace	Europe

"The postman delivered a letter to Mary's mailbox."

pre situation	a letter	notAtPlace	Mary's mailbox
post situation	a letter	atPlace	Mary's mailbox

"The postman delivered a letter to Mary.*"

pre situation -
post situation -

**Note that 'Mary' is a 'Beneficiary' according to FrameNet. The fn:Beneficiary is not mapped to ESO translocation-goal.*

-Working subclassOf: StaticEvent
"Static event where someone is doing work. "

Class mappings:
closeMatch: fn:Working_a_post
closeMatch: fn:Work

Role mappings:
working-entity: fn:Agent

Assertions:

during situation	working-entity	works	true

EXAMPLES:

"John works hard on a new book."

during situation	John	works	true

"John and Mary manned the front desk."

during situation	John and Mary	works	true