Taxonomy of Events and Components in the Power Grid

Technical description for work packages 3.1 and 3.2 of the ELVIRA Project

Technical Report HS-IIT-TR-18-001

Editor: Manfred Jeusfeld

2018-01-31

Authors

Manfred Jeusfeld (University of Skövde)

Yuning Jiang (University of Skövde)

Jianguo Ding (University of Skövde)

Yacine Atif (University of Skövde)

Daniel Haglund (Combitech AB)

Christoffer Brax (Combitech AB)

EUROPEAN UNION
Internal Security Fund ISF

Vi driver ett projekt som bidrar till att stärka EUs inre säkerhet.
http://his.se/elvira

EXECUTIVE SUMMARY

This document reports a technical description of ELVIRA project results obtained as part of Work-package 3.1&3.2 entitled “Taxonomy of Critical Infrastructure (Taxonomy of events + Taxonomy of CI component and relationship)”. ELVIRA project is a collaboration between researchers in School of IT at University of Skövde and Combitech Technical Consulting Company in Sweden, with the aim to design, develop and test a testbed simulator for critical infrastructures cybersecurity.

We investigate how power grid components are described in the literature and in documentations of power grid modeling & simulation tools, in particular related to the Nordic32 model. Based on this analysis, we organized group meetings, where we jointly created a taxonomy of power grid components. This taxonomy was extended by concepts for IT components to cover the cyber-physical nature of the power grid.

The taxonomy of components was extended by a taxonomy of events. This taxonomy merges two existing event modeling approaches. The first one represents events as a multidimensional object (time, places, agent, input, and output). The second one models causal relations between event types. In our taxonomy, we thus differentiate between ‘Event’ (= actual event) and ‘EventType’ (class of events).

We use the ConceptBase system to represent both the taxonomy of components and the taxonomy of events. Further, ConceptBase is used to represent actual or hypothetical power grids by instantiating the component taxonomy. The Nordic32 model has been used to validate the expressiveness of the taxonomy.

Table of Contents

Table of Figures:

1. Tools Selection

Conceptual modelling is based on rationale analysis of ontology. According to Tom Gruber (1992), “an ontology is a specification of a conceptualization”, it also works as a statement for certain logical theories. In the Elvira project, we generate ontological definitions for both components and events from a knowledge-level perspective (Newell, 1982). The ontology was adopted to have clear and uniform definitions for entities and their relationships in the power-grid system. Those definitions could be invoked in different models and be used for models transferring, and also work as a basis for model integration framework.

After comparing and testing several existing conceptual modelling tools, such as IDEF-based Modelling, Protegé (https://protege.stanford.edu/), and ConceptBase Modelling, we eventually decided to use the ConceptBase system (http://conceptbase.sourceforge.net/), because it allows to represent meta classes, classes, and instances in a uniform way. It also has a sophisticated query language to analyze large (graphical) models. Besides, we use a single central server to maintain the ontology and to provide multi-user access to the taxonomy for all project members.

1.1 Features of ConceptBase

ConceptBase is a multi-user deductive database system with an object-oriented (data, class, metaclass, meta-metaclass, etc.) data model, which makes it a powerful tool for metamodeling and engineering of customized modeling languages. The system is accompanied by a highly configurable graphical user interface that builds upon the logic-based features of the ConceptBase server. Compared to other tools, it has the following features that distinguishes it and makes it suitable for the purpose of the ELVIRA project:

Unlimited metaclass hierarchies. ConceptBase can represent information at the data level (example data, traces of process executions etc.), the class level (schemas, process definitions etc.), the metaclass level (constructs of modeling languages), the meta-metaclass level (constructs for defining modeling languages), and so forth. There are no fixed hierarchies but rather the notion of instantiation between two objects, one being the instance, the other the class. There are also classes whose instances span over several metaclass hierarchies. For example, the class *Proposition* has all objects as instances, regardless of their abstraction level.

Uniform object representation. All objects are represented in a uniform quadruple data structure called P-facts (or "propositions") pioneered by the developers of the O-Telos knowledge representation language. Objects, their attributes, specializations, and instantiations are all represented as P-facts. By this, for example, attributes of objects can have attributes and are instances of other attributes. Specialization between attributes is fully supported.

Logical expressions. Deductive rules, integrity constraints, and queries are expressed in first-order logic formulas. Internally, the system transforms them into Horn clauses interpreted by a Datalog-based evaluation machine. Logical expressions in ConceptBase can range over class of objects regardless of its type (node vs. link) and its abstraction level. Datalog is known for being the most robust computational system for evaluating logical expressions.

Active rules. Active rules update the database or call external routines as a reaction to events. The execution follows the Event-Condition-Action scheme (ECA). Updates to the database are formulated via Tell/Untell/Retell commands. External routines (e.g. for invoking a script to send an email) can be incrementally added to the database using a simple Prolog-based programming interface.

Functional expressions. ConceptBase supports functional and arithmetic expressions to define computation within models. Functions can be recursively defined much like in functional programming. For example, the length of the shortest path between two nodes is defined as the minimum of the lengths of the shortest path between the successors of the start node and the end node. Functional expressions are particularly useful for defining complex metrics on models. Aggregation functions such as SUM and AVG are predefined.

The O-Telos knowledge representation language is the data model underlying the ConceptBase system. It has a single base relation (P-fact P(o,x,n,y)) by which concepts at any abstraction layer (instance, class, meta class, etc.) can be defined. Deductive rules and integrity constraints extend the computational model to that of a deductive database.

O-Telos exhibits an extreme usage of Datalog:

- there is only one base relation P(o,x,l,y) used for all objects, classes, meta classes, attributes, class membership relationships, and specialization relationships as well as for deductive rules, integrity constraints and queries.
- the semantics of class membership, specialization, and attribution is encoded by around 30 axioms which are either deductive rules or integrity constraints
- deductive rules ranging over more than one classification level (instances, classes, meta classes, etc.) are partially evaluated to a collection of rules (or constraints) ranging over exactly one classification level

The latter feature is the key to define the semantics of abstraction principles like specialization. For example, the inheritance of class membership is defined by the rules

```
forall o,x,c P(o,x,in,c) ==> In(x,c)
forall x,c,d In(x,c) and Isa(c,d) ==> In(x,d)
```

The first rule derives the predicate In(x,c) from P-facts P(o,x,in,c). The second one then defines class membership inheritance. Given the extension of the predicate Isa(c,d), one can partially evaluate such a formula. For example, if Isa(student,person) is part of the extension, then the rule is partially evaluated to

```
forall x In(x,student) ==> In(x,person)
```

Hence, the O-Telos knowledge representation language combines the uniformity of the P-facts data model with the expressive power of defining deductive rules. The deductive rules can be used to query/analyze a given model (e.g. to find which components are connected to each other with the same nominal voltage, see section 2), to derive properties and relations, to compute graph metrics such as the cyclomatic number, and to define consistency constraints such as constraints of proper configurations of power grids.

The metamodeling feature allows to model concepts at multiple abstraction levels, e.g. component types vs. component instances. Since rules and constraints can be formulated at any abstraction level, ConceptBase supports the definition of dedicate modeling constructs including their semantics. Examples are reported in (Jeusfeld et al., 2009).

In ELVIRA, we use ConceptBase to represent the component as well as event ontologies plus their instances. Instances of the component taxonomy are actual or hypothetical power grids such as Nordic32 (Walve, 1993). ConceptBase can maintain multiple such power grids in parallel as instances of the taxonomy. The taxonomy is thus also a domain-specific language (Tolvanen and Kelly, 2009) to represent power grids. Likewise, the event taxonomy can be used to instantiate event types to form actual event instances (occuring at a given time, location, and involving specific power grid components). While Protegé also allows to represent instances (called individuals), the system quickly reaches its limits when a high number of rules is incorporated (Terkaij, 2017).

2. Taxonomy of Power-Grid Components

Taxonomies are a special form of ontologies (Gruber, 1992). In ELVIRA, a taxonomy of components is set up to represent power components and IT (“cyber”) components, and also their possible relations such as power connections and data flows. In addition, the ontology is augmented by rules to analyze an example power grid model which is Nordic32 power model.

2.1. Top View

In Figure 1, the taxonomies of components are presented from a top view. In the figure, links with white arrowheads denote subclasses, black links denote associations. “Thing” is the top class that represent every existing entity in the power network system. “Component” is the subclass of “Thing” with data connections between components. Both “CyberComponent” and “PhysicalComponent” are subclasses of “Component”, among which “CyberComponent” either monitors or controls, or is embedded in “PhysicalComponent”. Besides, “PhysicalComponent” could have physical connections among each other. “PowerGridComponent” is a subclass of “PhysicalComponent”. Power connections between power grid components allow to pass electrical energy.

Note that we do not restrict data connections to cyber components. By defining them as a feature of “Component”, any component (including physical computers) can be a node in the data connection grid.

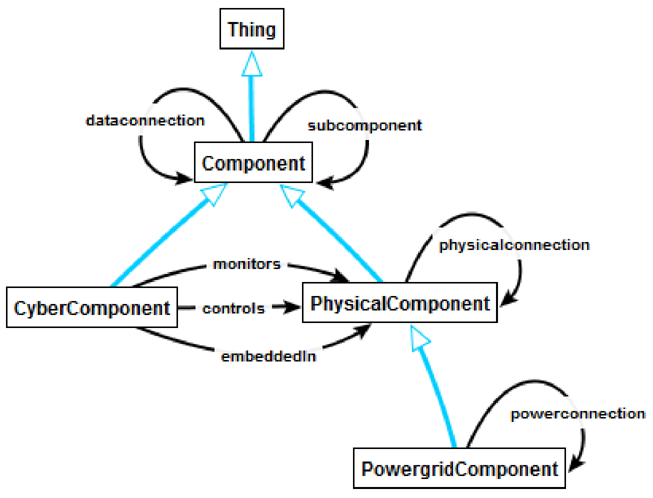

Figure 1. Taxonomy of Components (top view).

The relation “subcomponent” allows to decompose any component (cyber or physical) into parts. For example, a substation may contain a number of transformers, busbars, circuit breakers and shunts. Likewise, a monitoring information system may contain several software modules, like a local database, as parts.

Physical connections express that a physical component is permanently coupled with another component. Most if not all power connections are also physical connections. Such information is relevant for ELVIRA because if two parts are connected, then they can influence each other’s states. For example, if a component is physically destroyed, then connected components may become dysfunctional as well. Similar rules may hold for the “subcomponent” and “dataconnection” links.

2.2. Taxonomy of Power Components

In Figure 2, the taxonomy of power components is presented. For example, “SubStation”, containing 4 different subclasses (“CollectorSubStation”, “DistributionSubStation”, “TransmissionSubStation” and “SwitchingStation”), is a subclass of “PowerGridComponent”.

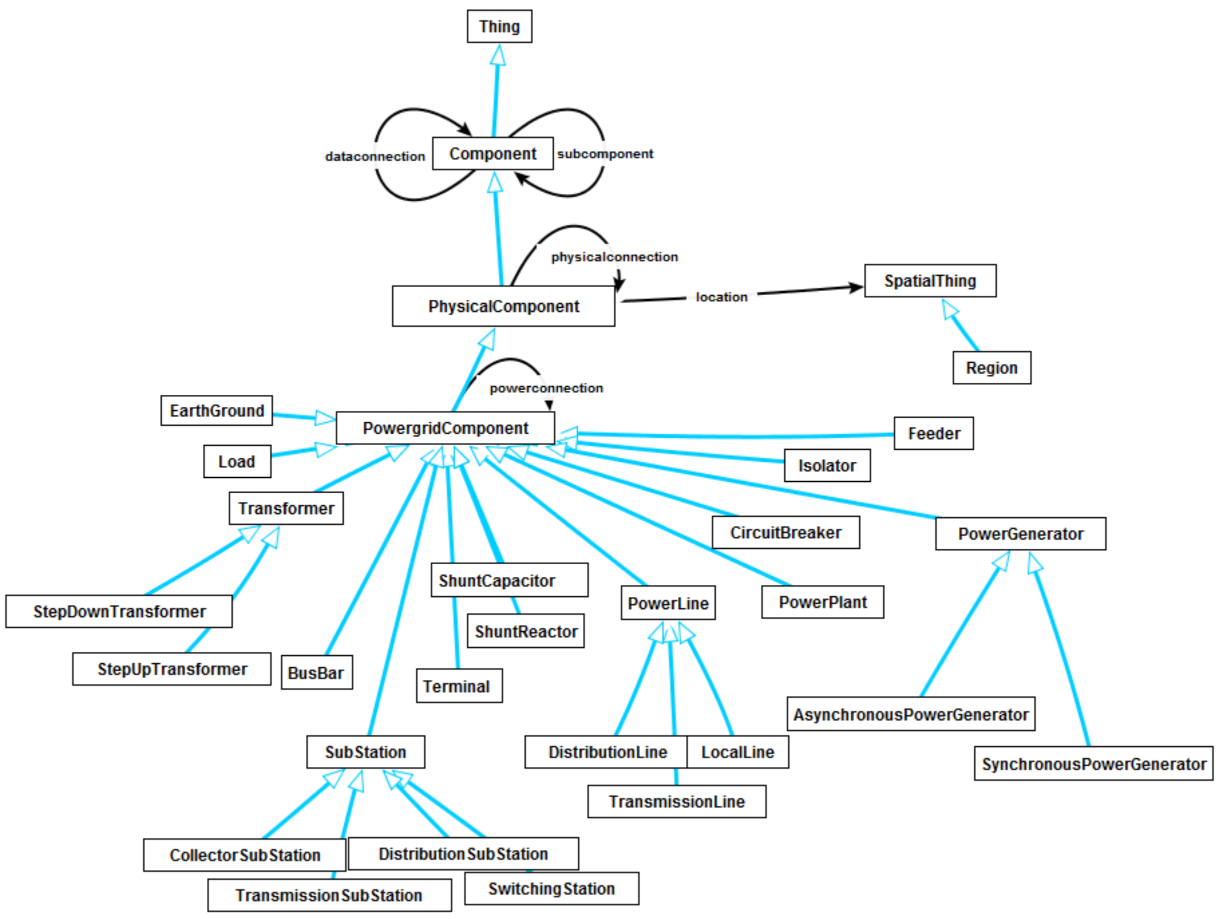

Figure 2. Taxonomy of Power Components.

We define rules to represent power components and their relationships, through which the data type and the properties of components could be classified and further identified. Two examples are shown below.

- Example 1: Power-Grid-Component Definition

```
PowergridComponent in Class isA PhysicalComponent with
  attribute
    powerconnection : PowergridComponent;
    nominalvoltage : String;
    nominalfrequency : String
  rule
    voltrule : $ forall p1,p2/PowergridComponent v/String
             ((p1 powerconnection p2)  or (p2 powerconnection p1)) and (p1 nominalvoltage v)  and
               not (p1 in Transformer) and not (p2 in Transformer) ==> (p2 nominalvoltage v) $
end
```

- Example 2: Transformer Definition

```
Transformer in Class isA PowergridComponent with
  attribute
    lowervoltage : String;
    uppervoltage : String
  rule
    voltrule1 : $ forall t/Transformer pc1,pc2/PowergridComponent v1,v2/String
                   (pc1 nominalvoltage v1) and (pc2 nominalvoltage v2) and (v1 < v2)
               ==> (t lowervoltage v1) $;
    voltrule2 : $ forall t/Transformer pc1,pc2/PowergridComponent v1,v2/String
                        (pc1 nominalvoltage v1) and (pc2 nominalvoltage v2) and (v1 < v2)
                    ==> (t uppervoltage v2) $
  end
```

The first rule (voltrule) transitively propagates the nominal voltage of a power grid component to all its neighbors. The propagation is terminated at transformers. Note that the rule is recursive. The other two rules derive the upper and lower voltage of a transformer. Additional constraints can be used to specify that any power grid component has a single nominal voltage, except for transformers. These constraints allow to trace flaws on a power grid model. The rules and constraints are compiled by ConceptBase into executable code, allowing to query the consequences of those rules. Further details on the rule and constraint language are described in (Jeusfeld, 2018).

2.3. Taxonomy of Cyber Components

In Figure 3, the taxonomies of cyber components are presented. Cyber components are embedded in physical components, and are used to monitor and control physical components. The taxonomy is extensible at any time. Both classes “CyberComponent” and “PhysicalComponent” are shown because cyber component could be a separate entity or an entity that is embedded in certain physical component”. For instance, a specific firmware is embedded in a router, which is a sub-component of a local area network, which itself is part of some substation.

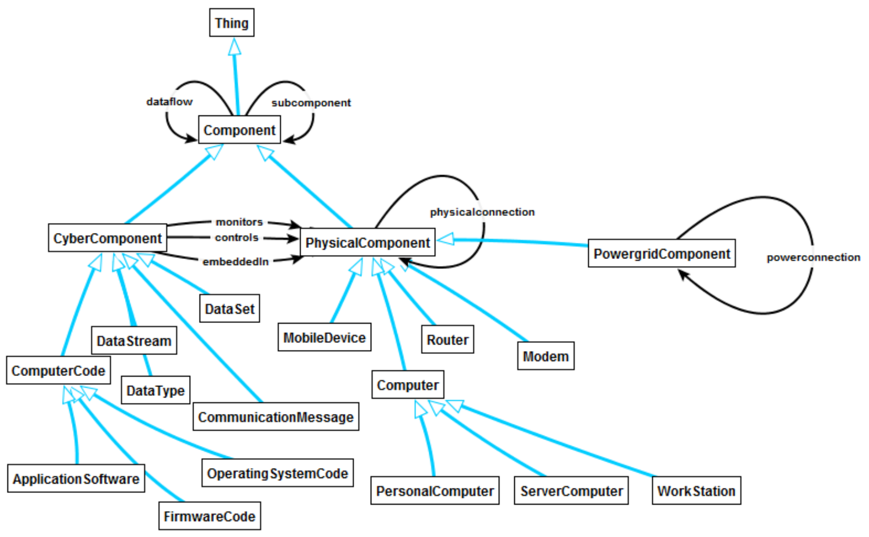

Figure 3. Taxonomy of Cyber Components.

3. Taxonomy of Events

In Figure 4, taxonomies of events are shown. Both event and event type are presented. The “EventType” class models causal and other relations between event types, e.g. how likely is it that a fire in a substation causes a transformer to fail. “Event” stands for actual events recorded at some time (either point of time or time interval).

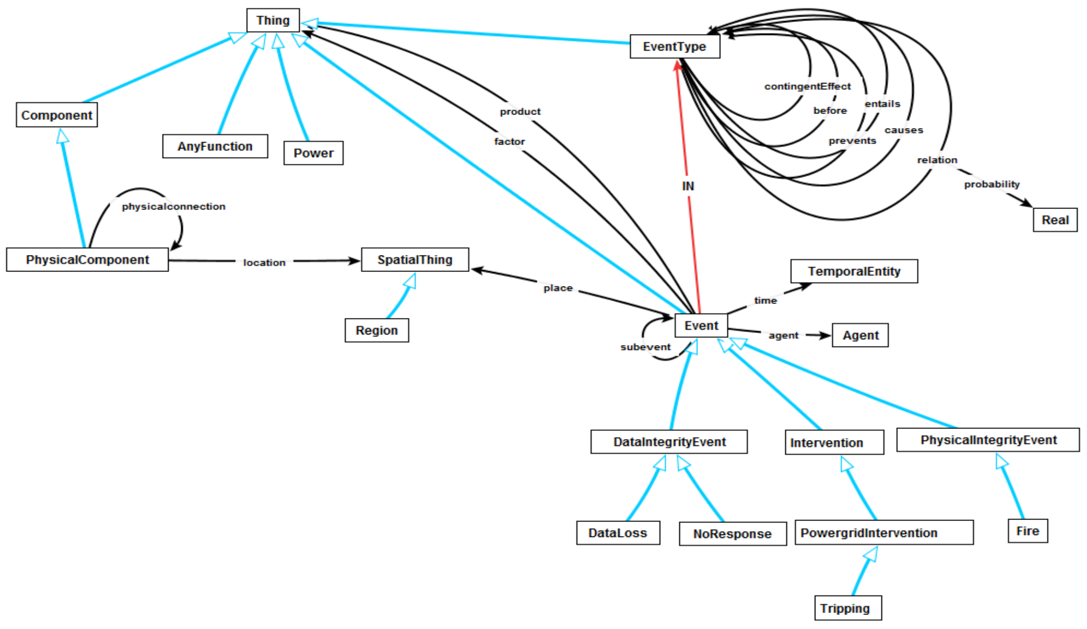

Figure 4. Taxonomy of Events.

The ELVIRA taxonomy of events combines two existing event ontologies. The upper part (“EventType”) encodes the circumstantial event ontology (Segers et al., 2017). It allows to specify causal and other relations between event types. An event type (“EventType”) stands for a set of actual events (“Event”) that share some properties, e.g. data integrity events involving routers in a substation that use a specific firmware. Relations between such event types store empirical knowledge, for example that a data integrity event on such routers can lead to actuating a circuit breaker in the substation and then lead to power loss on power lines.

The class “Event” encodes the event ontology of (Raimond and Abdallah, 2007). This ontology is concerned with describing actual events. In their view, such events are *arbitrary qualification of a space-time by a cognitive agent* (i.e., an agent that observes space-time). As a consequence, an event has the dimensions “place”, “time”, and “agent”. In addition, an event has “factors” (anything that contributed to the event from the viewpoint of the observing agent) and a “product” (anything that is the result of the event). By default, all these relations are multi-valued. There may be several factors (any instance of the most general class “Thing”) and also several products. In particular, any component of the component taxonomy can be a factor of an event, and any event may could be the product of an event.

The two ontologies are linked via the “IN” relation. This relation has been introduced in (Jeusfeld and Neumayr, 2016). The meaning is that any instance of “EventType” shall be a subclass of “Event”, and any subclass of “Event” shall be an instance of “EventType”. This allows us to use the relations of “EventType” to express empirical knowledge at the level of instances of “EventType”. For example, the event that the firmware of a programmable logic controller (PLC) that controls a circuit breaker of a substation has been compromised leads with some likelihood to unexpected power shut-downs of that circuit breaker. We can express such rules for event types, and regard actual event instances from the past as empirical evidence backing this rule.

4. Integrated ELVIRA Taxonomy

Figure 5 presents the integrated taxonomy for project ELVIRA. The integration is via two paths. First, the dimensions event taxonomy concepts are all subclasses of Thing. As a consequence, the factors and the results of an event can be any components in the component taxonomy (power grid components and cyber components). Second, events have a possible location (“SpatialThing”). This is the same class which is also used to specify where physical components are located.

The full source text of the integrated ELVIRA Taxonomy is listed in Appendix A.

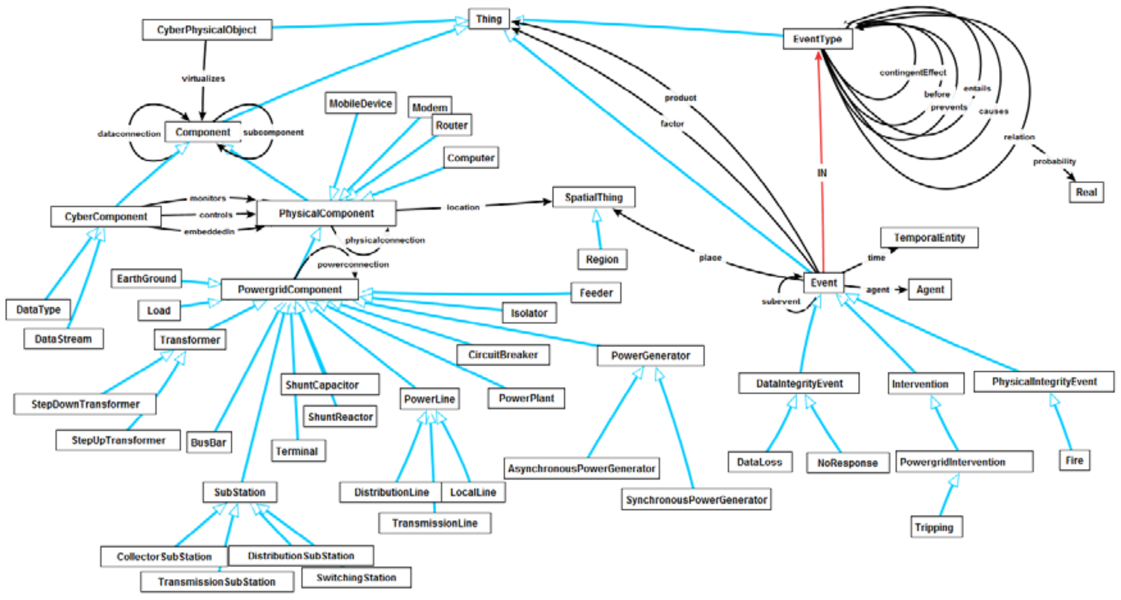

Figure 5. Integrated ELVIRA Taxonomy.

5. Nordic32 Model Representation Example

In Figure 6, a small part of Nordic32 (Walve, 1993) power grid model is presented as an example of conceptual modelling. In the figure, black rectangles symbolize 'bus bars' which are massive copper components to link power components, for instance power generators g19, g9, g10 and Transformer tf4011_1011. The figure is graphical view on its representation on ConceptBase and directly excerpted from the ConceptBase system.

The graphical view is a visualization of the the model. Internally, the system stores the models as an instance of the classes defined in the ELVIRA component taxonomy. For example, the element bb4071 is defined as:

```
bb4071 in BusBar with
  powerconnection
        pc1 : ld4071;
        pc2 : tl4071_4072a;
        pc3 : tl4071_4072b;
        pc4 : gr4071;
        pc9_2 : tl4071_4011;
        pc19_6 : tl4071_4012
end
iso19 in Isolator with
  powerconnection
        pc1 : bb4071
end
```

The first line "bb4071 in BusBar" declares it as an instance of "BusBar". Since BusBar is a specialization of PowerGridComponent", it can use the relation "powerconnection". In this case it is connected to the load "ld4071" and a number of other components. The relation "powerconnection" is a symmetric relation. As a consequence, it is also connected to the isolator "iso19".

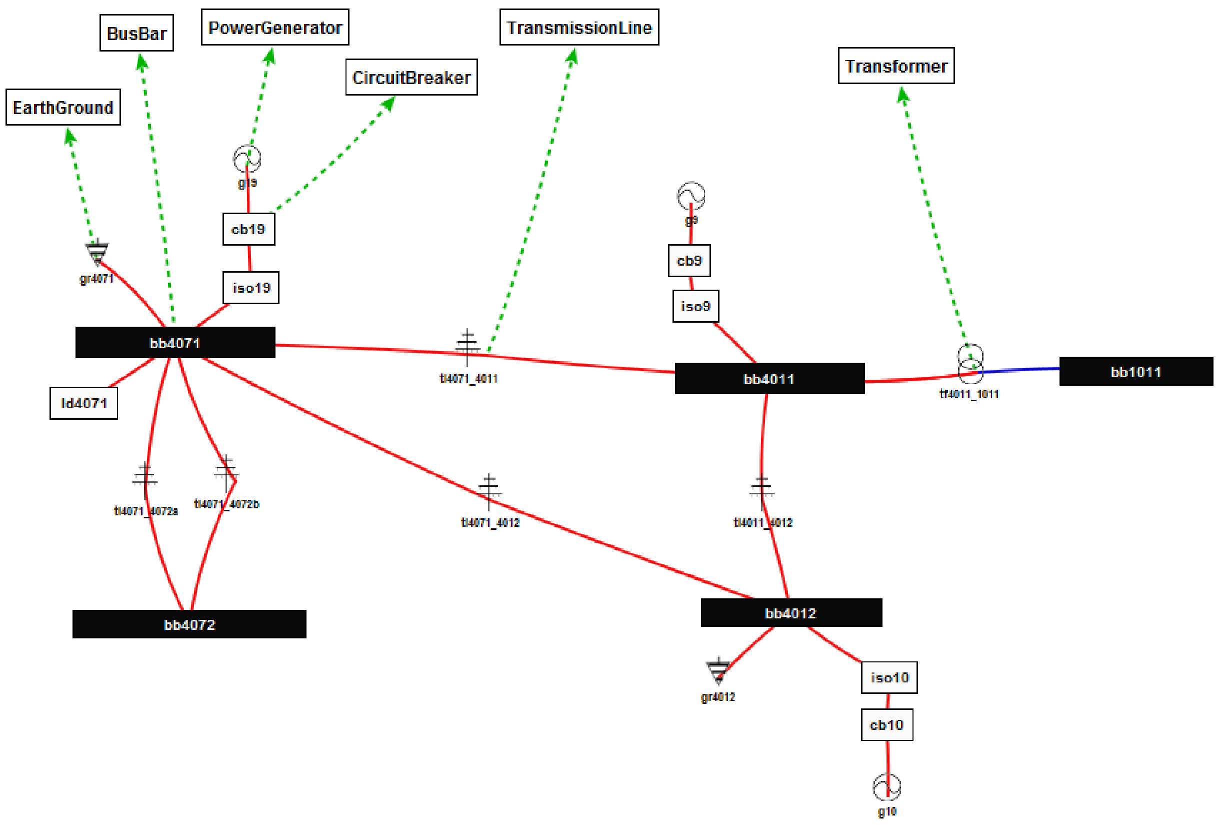

Figure 6. Excerpt of Nordic32 model as instance of the component taxonomy.

Power lines such as t4071_4011 are regarded as components just like transformers, circuit breakers, and so forth. They may physically occupy a much larger area than other component. The physical area shall be represented by the ‘location’ relation of “PhysicalComponent”. Geographic proximity can thus be represented in our power grid models and used for reasoning of cascading effects. For example, a fire (or other physical event such as tripping) in one component has a certain likelihood to propagate to neighboring components.

The links between power grid components are called “power connections”. A power connection allows a flow of electric power between two components. The power connection itself is however, not a component.

The full source text of the Nordic32 excerpt is presented in Appendix B.

6. Conclusions

This report describes the results of work packages 3.1 and 3.2 of the ELVIRA project (www.his.se/elvira). The result is an integrated taxonomy of events and components. The taxonomy is stored in the ConceptBase system and represent instances of components (power and cyber networks as well as instances of events. The system is thus usable as a knowledge base for the project members and can be an integral part of the software tools to be developed for establishing the current state of the grid and for managing empirical rules for event propagation.

The scientific contribution of this report can be summarized as follows:

- We unified two event ontologies, one for event types and the other for event instances.
- We integrated the event ontology with a taxonomy of power grid components that includes physical and cyber components.
- The component taxonomy has been tested as basis of a domain-specific modeling language (DSML) for power grid components. We used the Nordic32 dataset to check the expressive power of the DSML.

7. References

T.R. Gruber (1992). What is an ontology? Online http://www-ksl.stanford.edu/kst/what-is-an-ontology.html

M.A. Jeusfeld, M. Jarke, J. Mylopoulos (eds., 2009). Metamodeling for Method Engineering. Cambridge, MA, 2009. The MIT Press.

M.A. Jeusfeld, B. Neumayr: DeepTelos - Multi-level Modeling with Most General Instances. In Proc. 35th Int. Conf. on Conceptual Modeling (ER 2016), Gifu, Japan, Springer, LNCS 9974, Nov 14-17, 2016, pp. 198-211.

M.A. Jeusfeld (ed., 2018). ConceptBase.cc User Manual Version 8.0. Online http://conceptbase.sourceforge.net/userManual80/

Y. Raimond, S. Abdallah (2007). Event ontology. Online http://motools.sourceforge.net/event/event.html

R. Segers, T. Caselli, P. Vossen (2017). The circumstantial event ontology (CEO). Proceedings of the Events and Stories in the News Workshop, pages 37–41, Vancouver, Canada, August 4, 2017.

W. Terkaj (2017). OntoGui: A Graphical User Interface for Rapid Instantiation of OWL Ontologies. Proceedings of the Joint Ontology Workshops 2017, CEUR Workshop Proceedings, CEUR-WS.org/Vol-2050, 2017.

J-P. Tolvanen, S. Kelly (2009). MetaEdit+: defining and using integrated domain-specific modeling languages. Proceedings OOPSLA'2009, ACM, DOI http://doi.acm.org/10.1145/1639950.1640031.

K. Walve (1993). Nordic32A – A Cigre test system for simulation of transient stability and long term dynamics, Svenska Kraftnät, Sweden, March 15, 1993 (rev. March 01, 1994).

Appendix A: Sources of the ELVIRA taxonomies

```
{*
* Module: System-oHome-ElviraTaxonomy
* Listed for: jeum@HS_P_000648_hs_local_amd64_Windows7 at 2018-01-25 11:58:07.736
(UTC)
*}
{* 2017-11-29 10:51:27.658 *}



Thing
end
TemporalEntity
end
TimePoint isA TemporalEntity
end
TimeInterval isA TemporalEntity
end
SpatialThing
end
Agent
end

Event with
  comment
       ref : "http://motools.sourceforge.net/event/event.html"
  attribute
       subevent : Event;
       time : TemporalEntity;
       agent : Agent;
       product : Thing;
       factor : Thing;
       place : SpatialThing
end

EventType with
  comment
       ref : "Roxane Segers, Tommaso Caselli, Piek Vossen: The Circumstantial Event
Ontology (CEO). Proceedings of the Events and Stories in the News Workshop, pages
37-41, Vancouver, Canada, August 4, 2017."
  attribute
       relation : EventType;
       causes : EventType;
       before : EventType;
       entails : EventType;
       prevents : EventType;
       contingentEffect : EventType
end

EventType!relation with
  attribute
       probability : Real
end

Event with
  IN
       c : EventType
end

Person isA Agent
end

Organization
end
```

```
EventType in Class
end

Component isA Thing with
  attribute
       subcomponent : Component
end

PhysicalComponent isA Component
end
CyberComponent isA Component
end

PhysicalComponent with
  attribute
       location : SpatialThing;
       physicalconnection : PhysicalComponent
end

Component with
  attribute
       input : Thing;
       output : Thing
end
Power isA Thing with
  attribute
       voltage : Integer
end

DataType
end
PowergridComponent isA PhysicalComponent
end
CircuitBreaker isA PowergridComponent
end
PowerLine isA PowergridComponent
end
DistributionLine isA PowerLine
end
TransmissionLine isA PowerLine
end
PowerGenerator isA PowergridComponent
end
SubStation isA PowergridComponent
end

CollectorSubStation isA SubStation with
  comment
       c1 : "A collector substation can contain either a step-up or step-down
transformers"
end

DistributionSubStation isA SubStation with
  comment
       c1 : "contains step-down transformers"
end

SwitchingStation isA SubStation
end
TransmissionSubStation isA SubStation
end
Transformer isA PowergridComponent
end
StepUpTransformer isA Transformer
end
StepDownTransformer isA Transformer
end
PowerNetwork isA PowergridComponent
```

```
end
TransmissionNetwork isA PowerNetwork
end
DistributionNetwork isA PowerNetwork
end

Component with
  attribute
       owner : Organization
end

AnyFunction isA Thing
end
PhysicalFunction isA AnyFunction
end
CyberFunction isA AnyFunction
end
DataTransmission isA CyberFunction
end
PowerSupply isA PhysicalFunction
end
PowerTransmission isA PhysicalFunction
end
PowerDistribution isA PhysicalFunction
end
PowerMonitoring isA AnyFunction
end

CyberPhysicalObject isA Thing with
  comment
       definition : "A cyber-physical object is always the virtual representation of
the combination of a physical component and some cyber component.It has a physical
part plus some software (mostly embedded software or firmware"
end

CyberComponent with
  comment
       definition : "A cyber-component is a component that is a data source or a
data sink, e.g. sending and receiving data from a SCADA-like system"
end

CyberPhysicalObject with
  attribute
       virtualizes : Component
end

CyberComponent with
  attribute
       embeddedIn : PhysicalComponent
end

Event isA Thing
end
EventType!contingentEffect isA EventType!relation
end
EventType!before isA EventType!relation
end
EventType!prevents isA EventType!relation
end
EventType!entails isA EventType!relation
end

EventType!causes isA EventType!relation
end

PowergridComponent with
  attribute
       powerconnection : PowergridComponent
```

```
end

PowergridComponent!powerconnection isA PhysicalComponent!physicalconnection
end

CyberComponent with
  attribute
       monitors : PhysicalComponent
end

PhysicalIntegrityEvent isA Event
end
Fire isA PhysicalIntegrityEvent
end
DataIntegrityEvent isA Event
end
DataLoss isA DataIntegrityEvent
end
NoResponse isA DataIntegrityEvent
end
Intervention isA Event
end

PowergridIntervention isA Intervention
end

Tripping isA PowergridIntervention with
  comment
       definition : "Tripping in a power plant/station occurs whenever fault
happens. it's a protective measure which essentially isolates the important devices
from the faulty section and thus saving it from getting destroyed (due to excessive
current or voltage across it)";
       ref1 : "https://www.quora.com/What-does-tripping-of-power-plant-electrical-
line-generator-power-grid-mean?no_redirect=1#!n=12"
end

Terminal isA PowergridComponent with
  comment
       definition : "A point at which a conductor from a power line comes to an end
and provides a point of connection to an external network, e.g. a customer"
end

BusBar isA PowergridComponent with
  comment
       definition : "A busbar is a metallic components connecting several electrical
components like power lines"
end

ShuntCapacitor isA PowergridComponent
end
ShuntReactor isA PowergridComponent
end

ShuntCapacitor with
  comment
       purpose : "Used to correct power factors"
end
CyberComponent with
  attribute
       controls : PhysicalComponent
end
Nordic32simple in Module
end
Isolator isA PowergridComponent
end
EventType isA Thing
end
Feeder isA PowergridComponent
```

```
end
Load isA PowergridComponent
end
EarthGround isA PowergridComponent
end
Region isA SpatialThing
end



LocalLine isA PowerLine
end
DataType isA CyberComponent
end
DataStream isA CyberComponent with
  attribute
       elementType : DataType
end
DataType isA Thing
end
DataStream isA Thing
end


PowergridComponent with
  attribute
       nominalvoltage : String;
       nominalfrequency : String
end

PowergridComponent in Class
end
$ forall p1,p2/PowergridComponent v/String
                           (p1 powerconnection p2) and (p1 nominalvoltage v) ==> (p2
nominalvoltage v) $
end


Transformer with
  attribute
       lowervoltage : String;
       uppervoltage : String
end


PowergridComponent with
  rule
       voltrule : $ forall p1,p2/PowergridComponent v/String
                           ((p1 powerconnection p2)  or (p2 powerconnection p1)) and
(p1 nominalvoltage v)  and
                    not (p1 in Transformer) and not (p2 in Transformer) ==> (p2
nominalvoltage v) $
end



Transformer in Class with
  rule
       voltrule1 : $ forall t/Transformer pc1,pc2/PowergridComponent v1,v2/String
                                  (pc1 nominalvoltage v1) and (pc2 nominalvoltage
v2) and (v1 < v2)
                           ==> (t lowervoltage v1) $;
       voltrule2 : $ forall t/Transformer pc1,pc2/PowergridComponent v1,v2/String
                                  (pc1 nominalvoltage v1) and (pc2 nominalvoltage
v2) and (v1 < v2)
                           ==> (t uppervoltage v2) $
end
```

```
PowergridComponent with
  rule
       pccolorrule1 : $ forall pc/PowergridComponent!powerconnection
p/PowergridComponent v/String
                                 (From(pc,p) or To(pc,p)) and (p nominalvoltage v)
and (v < "250kV")
                          ==> (pc gproperty/edgecolor "10,10,210") $
end

ComputerCode isA CyberComponent with
  comment
       definition1 : "Any computer-executable code"
end

OperatingSystemCode isA ComputerCode
end
FirmwareCode isA ComputerCode
end
ApplicationSoftware isA ComputerCode
end
DataSet isA CyberComponent
end
CommunicationMessage isA CyberComponent
end

Computer isA PhysicalComponent
end


Modem isA PhysicalComponent
end
Router isA PhysicalComponent
end
ServerComputer isA Computer
end
WorkStation isA Computer
end
PersonalComputer isA Computer
end
MobileDevice isA PhysicalComponent
end



ShortBusBar isA BusBar
end
LongBusBar isA BusBar
end

ShuntReactor with
  comment
       definition1 : "stores energy in the magnetic field"
end

ShuntCapacitor with
  comment
       definition1 : "A shunt capacitor bank stores energy in the electric field due
to polarization of the dielectric material"
end


SynchronousPowerGenerator isA PowerGenerator
end
AsynchronousPowerGenerator isA PowerGenerator
end
```

```
PowerPlant isA PowergridComponent
end


Component with
  attribute
       dataconnection : Component
end

Load with
  comment
       comment1 : "The Load attached is typically attached to a BusBar and is more a
parameter for powerflow simulation rather than a physical component"
end
PowerPlant isA PowerGenerator
end
```

Appendix B: Sources of Nordic32 partial model

```
{*
* Module: System-oHome-ElviraTaxonomy-Nordic32simple
* Listed for: jeum@HS_P_000648_hs_local_amd64_Windows7 at 2018-01-25 11:55:49.656
(UTC)
*}
{* 2017-11-29 10:51:28.009 *}
bb4072 end
ld4071 end
tl4071_4011 end
gr4012 end
tl4071_4012 end
tl4011_4012 end
tf4011_1011 end

g19 in PowerGenerator end
bb4071 in BusBar end
cb19 in CircuitBreaker end
iso19 in Isolator end

g19 with
  powerconnection
       pc1 : cb19
end
cb19 with
  powerconnection
       pc1 : iso19
end
iso19 with
  powerconnection
       pc1 : bb4071
end

bb4072 in BusBar end
ld4071 in Load end

bb4071 with
  powerconnection
       pc1 : ld4071
end

tl4071_4072a in TransmissionLine end
tl4071_4072b in TransmissionLine end

bb4071 with
  powerconnection
       pc2 : tl4071_4072a;
       pc3 : tl4071_4072b
end
tl4071_4072a with
  powerconnection
       pc1 : bb4072
end
tl4071_4072b with
  powerconnection
       pc1 : bb4072
end

gr4071 end

bb4071 with
  powerconnection
       pc4 : gr4071
end
```

```
gr4071 in EarthGround end

EXTERNALREGION in Region
end

bb4073 in BusBar end
bb4011 in BusBar end

g9 in PowerGenerator end
cb9 in CircuitBreaker end

iso9 in Isolator end
tl4071_4011 in TransmissionLine end

g9 with
  powerconnection
       pc9_1 : cb9
end
cb9 with
  powerconnection
       pc9_1 : iso9
end
iso9 with
  powerconnection
       pc9_1 : bb4011
end
tl4071_4011 with
  powerconnection
       pc19_5 : bb4011
end

ld4072 in Load end
g20 in PowerGenerator end
cb20 in CircuitBreaker end
iso20 in Isolator end

g20 with
  powerconnection
       pc20_1 : cb20
end
cb20 with
  powerconnection
       pc20_1 : iso20
end
iso20 with
  powerconnection
       pc20_1 : bb4072
end

bb4012 in BusBar end
gr4012 in EarthGround end
g10 in PowerGenerator end
cb10 in CircuitBreaker end
iso10 in CircuitBreaker end

g10 with
  powerconnection
       pc10_1 : cb10
end


cb10 with
  powerconnection
       pc10_1 : iso10
end
iso10 with
  powerconnection
```

```
       pc10_1 : bb4012
end
tl4071_4012 in TransmissionLine with
  powerconnection
       pc10_3 : bb4012
end
tl4011_4012 in TransmissionLine with
  powerconnection
       pc10_4 : bb4012
end

bb1011 in BusBar end
bb4071 with
  powerconnection
       pc9_2 : tl4071_4011;
       pc19_6 : tl4071_4012
end
bb4011 with
  powerconnection
       pc9_3 : tl4011_4012
end
bb4012 with
  powerconnection
       pc10_2 : gr4012
end
bb4011 with
  powerconnection
       pc9_6 : tf4011_1011
end
bb4072 with
  powerconnection
       pc20_2 : ld4072
end
tf4011_1011 in Transformer with
  powerconnection
       pc1011_1 : bb1011
end
bb4011 with
  powerconnection
       pc10 : tf4011_1011
end

g19 with
  nominalvoltage
       voltage : "400kV"
end

bb1011 with
  nominalvoltage
       voltage : "200kV"
end
```