Knowledge-based Engineering of Automation Systems using Ontologies and Engineering Data

Matthias Glawe[1], Christopher Tebbe[2], Alexander Fay[1] and Karl-Heinz Niemann[2]

[1]*Institute of Automation Technology, Helmut Schmidt University / University of the Federal Armed Forces, Holstenhofweg 85, 22043 Hamburg, Germany*

[2]*Faculty I – Electrical Engineering and Information Technology, University of Applied Sciences and Arts Hannover, Ricklinger Stadtweg 120, 30459 Hannover, Germany*

Keywords: Ontology, Knowledge-based Engineering, Semantic Web, CAEX, IT Security.

Abstract: Ontologies provide an effective way for describing and using knowledge of a specific domain. In engineering workflows the reusability and quick adoption of knowledge is needed for solving several tasks in efficient ways. Engineering data is mostly structured in hierarchical documents and exchange formats, but is not represented in ontologies. Therefore a connection between engineering data and the knowledge in ontologies is needed. In this article we present a *bridge* concept for connecting engineering data with an OWL-based ontology. For this we use an example ontology containing security knowledge of automation systems.

1 INTRODUCTION

The engineering of an automation system for a production plant or a building requires significant effort and a dedicated workflow, organised in subsequent phases (Vogel-Heuser et al., 2014). Within this engineering workflow, some tasks require creativity, whereas other phases are characterised to be repetitive and tedious tasks, which makes the use of software tools advisable (Frank et al., 2012). Especially for the repetitive tasks, the use of knowledge-based support tools has proven to be advantageous in terms of effort savings, time savings, and quality assurance (Strube et al., 2011, Runde and Fay, 2011, Legat et al., 2013). If the engineering expert's knowledge can be formulated in IF-THEN statements, these rules can be applied on the engineering data of the current engineering project. These form the facts[1] of the current problem, from the perspective of the knowledge-based systems (Russell and Norvig, 2010; Fulcher and Jain, 2008).

In the engineering workflow, rule-based systems can not only support the creation of engineering results (e.g. Schmidberger and Fay, 2007; Güttel et al., 2008) but also the analysis of engineering results regarding e.g. completeness, reliability (e.g. Christiansen, 2011) and safety (e.g. Schreiber, 2007). Recently, the analysis of the automation system's design regarding IT security threats has become more important due to increased vulnerability of commercial-of-the-shelf components and standard communication technologies in automation systems. Therefore, IT security analyses, denoted as security analyses in the following, of the automation system have to be conducted. These analyses should take place during the design of the automation system as well as along its operation (IEC 62443-2-1). Security analyses might be highly complex and time consuming, as many details regarding the automation system and the implementation environment have to be considered. Likewise, they require a significant amount of up-to-date security knowledge. Due to resource limitations, however, in practice there is not always time to conduct such an analysis, and the required knowledge is often not available, particularly in small and medium-size companies.

In the research project INSA[2], a tool to support security analyses during the engineering and opera-

[1]"The term facts mean information that is considered reliable. Expert systems draw inferences using facts." (Giarratano 2005, p. 72).

[2]ut.hsu-hh.de/insa

tion of automated systems has been developed. This tool exploits the capabilities of a rule-based system on the combination of engineering data and security ontologies. This combination concept is discussed in this paper.

The rest of the paper is structured as follows. In Chapter 2 the chosen data formats for representing facts about the automation system and the domain knowledge for security analyses are introduced. Chapter 3 provides a concept for connecting the facts, given in an engineering data format, with the security domain knowledge represented in the ontology. A design concept for a suitable user interface is illustrated in Chapter 4 to provide an intuitive possibility for creating complex rules. Finally, a short conclusion and an outlook to further development and possible implementations of the presented concepts for other engineering tasks are given in Chapter 5.

2 DOMAIN & TECHNOLOGIES

In this chapter a brief introduction to the domain of security, engineering data and ontologies is given. In our work we designed an exemplary automation system to proof our methods and concepts. A part of this exemplary automation system is shown in Figure 1.

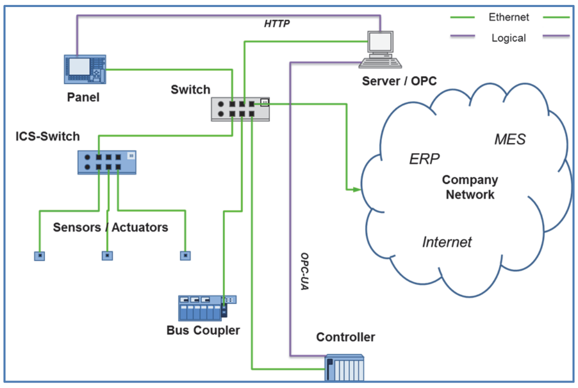


Figure 1: Part of an automation system.

The automation system consists of Ethernet-based intelligent sensors/actuators, further sensors/actuators connected to a bus coupler and a controller (PLC = Programmable Logic Controller), which runs the automation system.

This controller communicates automation data via OPC-UA, an automation specific communication standard, to an industrial server. This server provides this data via a WebServer for other users. In this case a panel collects this data by a HTTP communication.

The elements of the automation system are connected by industrial Ethernet and have a connection to the companies' network to communicate with other instances like a MES (Manufacturing Execution System), an ERP (Enterprise Resource Planning System) or the Internet. This structure is used in the following chapters to illustrate specific elements of our approach.

2.1 Security of Automation Systems

While security mechanisms have been present in Home- and Enterprise-IT for a long time, many industrial automation systems do not have an acceptable level of IT-Security up to now. This originates in the fact that automation systems used to consist of proprietary components which were not in focus of hackers and had (nearly) no connection to external systems. A change in mind was needed when modern automation systems adopted more and more standard IT techniques, like the Ethernet and standard software. For example the industrial server in our example runs a Windows-based operating system while the operation system of the controllers is Linux-based.

Especially for small- and medium-sized companies it is hard to reach an acceptable level of security in their automation system due to low knowledge about security and the high effort for implementing security in automation systems. In our research we focus on the creation of a rule-based system to support companies in the execution of a security analysis for their automation systems.

For the creation of a rule-based system, knowledge of the specific domain has to be gathered for the current problem. It is necessary to analyse the structure of the domain knowledge that shall be represented, as a basis for the rule-based system. In the case of security knowledge for automation systems, this information can be found for example in (IEC 62443-2-1).

The structure of an automation system is the initial information needed to execute a security analysis. Within this information assets must be defined. These assets describe all elements that should be protected. Assets and their environment are subject to threats that can reduce the security of the automation system. The implementation of protection measures mitigates these threats and improves the security of the automation system. We described this in a generic model of a security analysis. A similar model is briefly introduced by (Valenzano, 2014).

2.2 Engineering Data in CAEX

In order to acquire asset data as the input for a security analysis and therefore as facts of a knowledge-based system, we decided to use existing engineering exchange data formats. One of the most recognized standards for exchanging engineering data is the XML-based data format *Computer Aided Engineering Exchange (CAEX)* (IEC 62424). A current implementation of CAEX is represented by the data format AML defined by the AutomationML e.V. (www.automationml.org). For AML, various implementations are available, e.g. an editor for creating AML files and a .NET-based engine for manipulating AML files.

The CAEX Model mainly consists of four libraries, which are defined as follows:

- *InstanceHierachyLibrary (IHL)*
  An InstanceHierarchy (IH) is a description of a specific hierarchy of components from top-level plant (the automation system) down to single components (*InternalElements*, IEs) (e.g. the Controller) with interfaces (*ExternalInterfaces*, EIs) and relations (InternalLinks, ILs) between these interfaces.
- *SystemUnitClassLibrary (SUCL)*
  The reusable *SystemUnitClasses* (SUCs) define the component types with their respective technical realizations. In our example a SystemUnitClassLibrary was created which contains all possible components of the automation system. These *SystemUnitClasses* were used to create the *InternalElements* in the *InstanceHierachy*. A component-catalogue from a particular vendor is a different example of a *SystemUnitClassLibrary*.
- *RoleClassLibrary* (*RCL*)
  In the *RoleClassLibrary, RoleClasses* (RCs) are defined as abstract description of component requirements. Therefore standard *RoleClassLibraries* are provided by the standard and can be complemented by the user. In our example the Panel fulfils the Role "HMI" (Human-Machine-Interface), which describes abstract control equipment without giving any information about the specific component.
- *InterfaceClassLibrary* (*ICL*)
  The *InterfaceLibrary* consists of reusable *InterfaceClasses* (ICs) for specifying connection points of RCs, SUCs, and interface types of EIs.

Regarding knowledge-based systems, *RoleClassLibrary*, *SystemUnitClassLibrary* and *InterfaceClassLibrary* contain domain knowledge about possible parts of the automation system, while the *InstanceHierachyLibrary* contains facts about the specific project under investigation.

2.3 Ontologies

Besides the information about automation systems, the security knowledge must also be formalized. A possible approach for the required formalization is the use of ontologies. In computer science the term ontology defines an explicit and formal specification of a conceptualization of a part of the real world (Antoniou and van Harmelen, 2012). Ontologies have already proven to be useful for the engineering of automation systems (e.g. Runde and Fay, 2011, Linnenberg et al., 2013). Donner and Ekelhart (Donner, 2003; Ekelhart et al., 2007) have shown that security knowledge can be described in form of ontologies, which allows the processing of queries by using existing query languages such as SPARQL.

Based on the aforementioned works, we also decided to use ontologies for our task. Specifically in the research project INSA, the "Web Ontology Language" (OWL) (W3C 2014 OWL) has been applied to describe the structure and content of the security knowledge in form of an ontology. OWL is specified as a Semantic Web standard by the W3C. It can be interpreted by humans as well as machines. The main part of our ontology consists of security threats, protection measures, their properties and the relationships between them.

Regarding the inference capabilities of the expert system, it is necessary that a reasoner (inference mechanism) can execute an analysis in a limited time frame. This is why we used OWL DL as a subset of the full OWL language. The appendix DL stands for description logic, which means OWL DL is comparable to the well-studied description logic. OWL full is known to be undecidable (http://www.w3.org/TR/owl-ref/), whereas OWL DL, with its restrictions, is decidable. This permits efficient reasoning support.

In order to use rules which can easily describe the relations between concepts of the real world (assets, threats and protection measures in our case), we used the Semantic Web Rule Language (SWRL). SWRL enables the usage of Horn-like rules (W3C 2014 SWRL) for OWL ontologies. This is another reason for using OWL DL because SWRL is based on OWL DL and RuleML and combines them. The implication rules of SWRL are perfectly usable to represent security knowledge in our ontology. Contrary to OWL, SWRL is not a W3C standard, but has the status of a W3C Member Submission and is used in many applications. The syntax of rules in SWRL is:

```
antecedent => consequent
```

This allows to define all dependencies on the "antecedent"-side (IF-part) and the resulting consequence on the "consequent"-side, which only applies if the "antecedent" is satisfied (THEN-part).

Contrary to our project, the works of (Donner, 2003) and (Ekelhart et al., 2007) analyse Enterprise-IT-Security and are not based on an import document, which contains the assets. In their works, it is supposed that security threats are analysed by human experts before the ontology can be evaluated. From our point of view, the generation of security threats from a given system infrastructure (assets) appears to be the most difficult task. Therefore, in our approach, the structure of the automation system is exported as a CAEX file during the engineering process, and this file is imported into the rule-based system.

3 COMBINATION CONCEPT

3.1 Basic Concept

As described before, the security domain knowledge is given in OWL/SWRL. This includes knowledge about possible threats and protection measures as well as rules, which infer new knowledge from the given facts. The facts about the automation system and the domain knowledge about possible components of the automation system are represented in CAEX. Thereby facts of the automation system are represented in the IHL, while domain knowledge about possible components of an automation system given in the SUCL, RCL and ICL. For executing the rules, a connection between the CAEX structure and OWL/SWRL is needed. Therefore, we developed a concept to connect all elements of the CAEX model with the OWL ontology by a small number of explicit and well defined operators, usable in SWRL rules. These operators are based on OWL Properties (object and data properties) and are used as part of the SWRL Rules in such a way that all relevant parts of the CAEX model could be addressed. Therefore the CAEX meta model, defined in (IEC 62424), gives an idea which elements have to be addressed. Before defining the specific operators a basic operator structure is defined. The operators should always look like:

```
COP_AOP_SOP(SWRL_Att1, SWRL_Att2)
```

In this notation, `COP` is a compare operator to define the sort of comparison between the CAEX element and the SWRL attribute. Up to now, compare operators for equality (`has`) and non-equality (`hasNOT`) as well as operators like more (`hasMore`) or less (`hasLess`) for version numbers or countable attribute values have been implemented. The further operators `in` and `notIN` check whether a given element is part of an enumeration which is represented in the ontology. `AOP` represents the address operator which defines the elements of the CAEX Meta Model (*RoleClassLibrary*, *SystemUnitClassLibrary* and *InterfaceClassLibrary*) to look for. It is the most important part of the operator which points to specific positions in the CAEX model. For supplementary information, the `AOP` operator can, in some cases, be extended by the supplementary operator `SOP`. The SWRL attributes `SWRL_Att1` and `SWRL_Att2` can either be a variable pointing to an element of the CAEX Model, an OWL individual, or the value which should be checked. The `AOP` and the optional `SOP` operators will be described in the following chapter.

3.2 CAEX Operators

In this chapter the operators pointing to the parts of the CAEX model are explained. As the facts for the current problem are given in the *InstanceHierachy*, the *InternalElements* in this library have to be checked. Due to this, the operators should be able to address all elements of these *InternalElements*. For a better understanding Figure 2 shows a part of the InstanceHierachy from our example describing the industrial server.

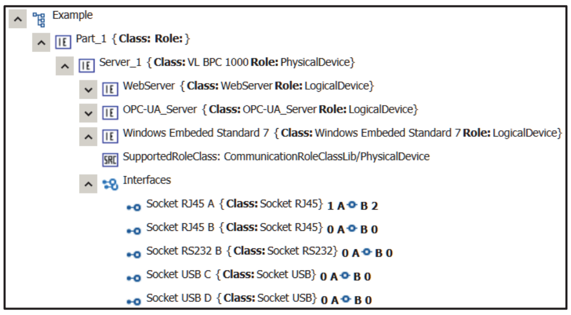

Example
Part_1 {**Class:** **Role:**}
Server_1 {**Class:** VL BPC 1000 **Role:** PhysicalDevice}
WebServer {**Class:** WebServer **Role:** LogicalDevice}
OPC-UA_Server {**Class:** OPC-UA_Server **Role:** LogicalDevice}
Windows Embeded Standard 7 {**Class:** Windows Embeded Standard 7 **Role:** LogicalDevice}
SupportedRoleClass: CommunicationRoleClassLib/PhysicalDevice
Interfaces
Socket RJ45 A {**Class:** Socket RJ45} **1 A B 2**
Socket RJ45 B {**Class:** Socket RJ45} **0 A B 0**
Socket RS232 B {**Class:** Socket RS232} **0 A B 0**
Socket USB C {**Class:** Socket USB} **0 A B 0**
Socket USB D {**Class:** Socket USB} **0 A B 0**

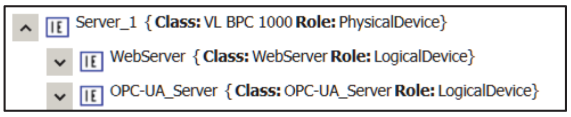
Figure 2: CAEX Representation of an industrial server.

As the *InternalElements* are instances of the *SystemUnitClasses*, an operator is needed which checks, if the *InternalElement* is the instance of a *SystemUnitClass*. For this reason the `AOP` operator `CAEXClass(?var, "Classname")` is defined, which checks, if an *InternalElement* `?var` is an instance of the *SystemUnitClass* `"Classname"`. In this case `has` is the `COP` operator equals. Due to the hierarchical structure of the *SystemUnitClassLibraries*, `"Classname"` can be the name of a specific *SystemUnitClass* or a group of *SystemUnitClasses*. In our example the class "VL BPC 1000" is a child class of the class "Hardware". Due to this the operator `has_CAEXClass(?x, "Hardware")` would also return true for the industrial server.

In the same way like described for *SystemUnitClasses* the operators `CAEXRole(?var, "Rolename")` and `CAEXInterface(?var, "Interface")` are defined to check the *RoleClass* and the *Interfaces* of an *InternalElement*. Deviating from the *SystemUnitClass* for *RoleClasses* it is necessary to differentiate between *RoleClasses* and *SupportedRoleClasses*. For this reason, the operator can be expanded by the `SOP` `"Supported"`.

Additionally *InternalElements* can contain attributes. Attributes are described by a name and the associated value. A general definition for all possible attributes is not possible, because attributes can be defined by the users without restrictions. By this, an operator is needed which can be defined generally but adopted dynamically to the existing attributes. Therefore an operator was defined as `CAEXAttribute_AttName(?var,"AttVal")`. This operator can match all attributes by just adding the appropriate attribute name as SOP. Because an attribute can be added to Interfaces, as well as *InternalElements*, a comparable operator is needed to access attributes of an *Interface*. This operator was defined as `CAEXInterfaceAttribute_AttName(?var, "AttName")`. By this a combination of checks for *Interfaces* and the concerning attributes of this interfaces is possible. For example a check like `has_CAEXInterface(?x,"Socket_RJ45")`, `has_CAEXInterfaceAtribute_Used(?x,true)` checks for all *InternalElements* with at least one RJ45 interface which is used. This operator will return true for the industrial server if one of the RJ45 sockets is used.

After having defined all operators to check the parts of an *InternalElement*, it is necessary to enable the rules to represent the structure of the automation system depicted in the *InstanceHierachy*. For this two more operators are needed to connect *InternalElements*.



Server_1 { **Class:** VL BPC 1000 **Role:** PhysicalDevice}
WebServer { **Class:** WebServer **Role:** LogicalDevice}
OPC-UA_Server { **Class:** OPC-UA_Server **Role:** LogicalDevice}

Figure 3: InternalElement including InternalElement.

Due to the hierarchical structure of CAEX, *InternalElements* can include other *Internal Elements*. The example in Figure 3 shows a controller used in an automation system which includes two other *InternalElements* representing the firmware of the controller and an OPC-UA Client. This enables the controller to communicate automation data to an associated OPC-UA server. To check for such structures the operator `CAEXInternalElement(?var1,?var2)` has been defined for checking whether the *InternalElement* represented by ?var2 is part of the *InternalElement* represented by ?var1. A check for a part of the structure, shown in Figure 3, looks like `has_CAEXClass(?x,"VL_BPC_1000")`, `has_CAEXClass(?y,"WebServer")`, `has_CAEXInternalElement(?x,?y)`.

In a similar way the operator `CAEXInternalLink(?var1,?var2)` has been defined to check for *Internal Links*. *InternalLinks* connect two *Interfaces* to each other, thus representing physical or logical connections or other types of relationships.

3.3 Combining Operators and Implementation

With the given operators `(AOP)` and their supplementing operators `(SOP)`, all facts given in an *InstanceHierachy* can be addressed. The combination with the compare operators `(COP)` leads to the complete operator. Although not all combinations of `AOP` and `COP` are possible, the compare operators `has` and `hasNOT`- are combinable with all CAEX operators.

Contrary to this, the compare operators `hasMore` and `hasLess`, which check whether a value is greater or lower than a given value, can only be interpreted for countable values like version numbers as a possible attribute. For `CAEXInternalElement` and `CAEXInternalLink` the operators `in` and `NOTin` are useless. These relationships can either exist or not, i.e. they cannot have any other state. The possible combinations of compare operator and address operator are shown in Table 1.

Table 1: Possible combination of compare operator and address operator.

	CAEXClass	CAEXRole	CAEXInterface	CAEXAttribute	CAEXInerface Attribute	CAEXInternal Element	CAEXInternal Link
has	X	X	X	X	X	X	X
hasNOT	X	X	X	X	X	X	X
hasMore				X	X		
hasLess				X	X		
in	X	X	X	X	X		
NOTin	X	X	X	X	X		

With these operators defined a reasoner is needed to process the SWRL rules. As shown before, the operators (OWL properties) directly address parts of the CAEX Model (using OWL Literals). By this, new methods are needed to check all parts of a rule, containing operators of our concept. AutomationML, as an implementation of CAEX, is used as part of the reasoner. The existing AutomationML Engine (Automation-ML 2015) is used to provide the functionalities needed to implement the defined operators. Up to now all operators can be implemented and connections to the OWL Elements can be evaluated. The rules created within our security ontology easily describe the relation between assets, threats and protection measures. For example a rule combines all hardware elements with an USB Interface with the threat TH4 that represents the threat "`storage elements are not properly checked before being used`" at this element. This rule can be represented as follows:

```
has_CAEXClass(?x,"Hardware"),
has_CAEXInterface(?x,"USB")
-> Threat(TH4)
```

With our additional methods, based on the AutomationML Engine, we are able to evaluate this rule and to infer corresponding OWL individuals (threats and protection measures). By this we can proof that our concept is able to connect CAEX models given in AML with ontologies given in OWL by using well defined OWL/SWRL operators. After an initial test of the designed methods, the operators are currently intensively tested to find potential needs for further adjustment of the implementation. By now only small adjustments have been required, and no further operators had to be defined.

3.4 Comparison to Existing Works

Another way to combine ontologies and engineering data is the conversion of CAEX to OWL. Theoretically OWL is able to represent all knowledge stored in CAEX. There already are works in this field like (Runde et al., 2009) and (Abele et al., 2013) which are converting CAEX structures to OWL to use existing query tools for solving some explicit tasks. The approach of (Abele et al., 2013) is aiming on answering single questions to check a CAEX Model for inconsistencies. Therefore the questions are given in existing query languages like SPARQL and are not using SWRL for a rule-based system. Also this approach provides the possibility to create SWRL Rules, the creation of rules and the preparation of the conversion from CAEX to OWL needs a high effort of preparation. The approach of (Runde et al., 2009) uses a knowledge-based system with SWRL rules for assisting the engineering in building automation. This approach can be adapted to other problem domains but intends much effort in the preparation of the transformation.

Due to the transformation to OWL both approaches can use existing reasoning or questioning tools and do not need additional reasoning tools. But both approaches are only transforming the CAEX Model to OWL and are applying checks. Concerning these facts these approaches should always be used to solve specific questions concerning the CAEX Model without creating changes to the CAEX Model.

On the other hand adapting changes to the CAEX Model is hard to be done. In the cases of (Abele et al., 2013) a transformation back to a CAEX Model is not intended and will lead to a loss of information. Therefore the adaption of information to the CAEX Model is not possible. The approach of (Runde et al., 2009) enables the transformation back to a CAEX File but assumes the creation of a special transformation, which contains meta information about the CAEX model and its transformation. The handling of these is complicated and prone to errors. By direct access to the CAEX Model our concept provides the possibility to use the CAEX Model after applying the knowledge in other tools without the risk of information loss. Also it allows a direct adoption of changes in the CAEX model without any additional effort.

4 USER INTERFACE CONCEPT

For checking special structures in automation systems, complex rules need to be created. Figure 4 shows a simple example of a possible part of an automation system which should be covered by the defined operators. In this example, a Programmable Logic Controller (PLC) communicates via an OPC-UA based communication with a Human-Machine Interface (HMI). One attribute of the OPC-UA communication is the attribute `"transport operator screen"`. This describes that information about the operator screen is transported from the PLC to a displaying unit. This attribute should be appended to this OPC-UA communication and set to `"true"`, if not existing.

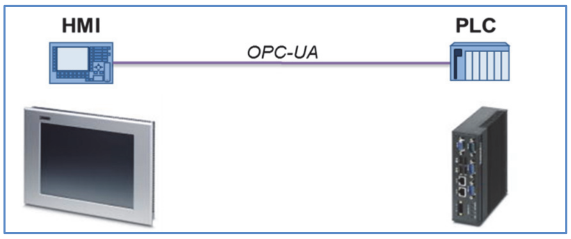


Figure 4: Part of an automation system.

Creating this rule, using the provided concept, it looks like depicted below.

```
   has_CAEXClass(?PLC, "PLC"),
has_CAEXInterface(?PLC, "OPC_UA"),
has_CAEXInternalLink(?PLC,?Comm),
has_CAEXClass(?Comm, "OPC-UA"),
has_CAEXInternal_Link(?Comm,?HMI),
has_CAEXInterface(?HMI, "OPC_UA"),
has_CAEXClass(?HMI, "HMI") ->
has_CAEXAttribute_
TransportOperatorScreen(?Comm, "true")
```

It is easy to understand that it is already difficult to create such rules in ontology editors like Protégé (http://protege.stanford.edu/). It becomes even more difficult when one tries to read or edit the XML code directly. Representing the SWRL rule in XML/RDF, the depicted rule contains of more than 100 lines of XML code. Editing such a complex rule manually it is likely to create syntactical wrong rules which are prone to errors.

Therefore a user interface concept was needed which matches the following requirements.

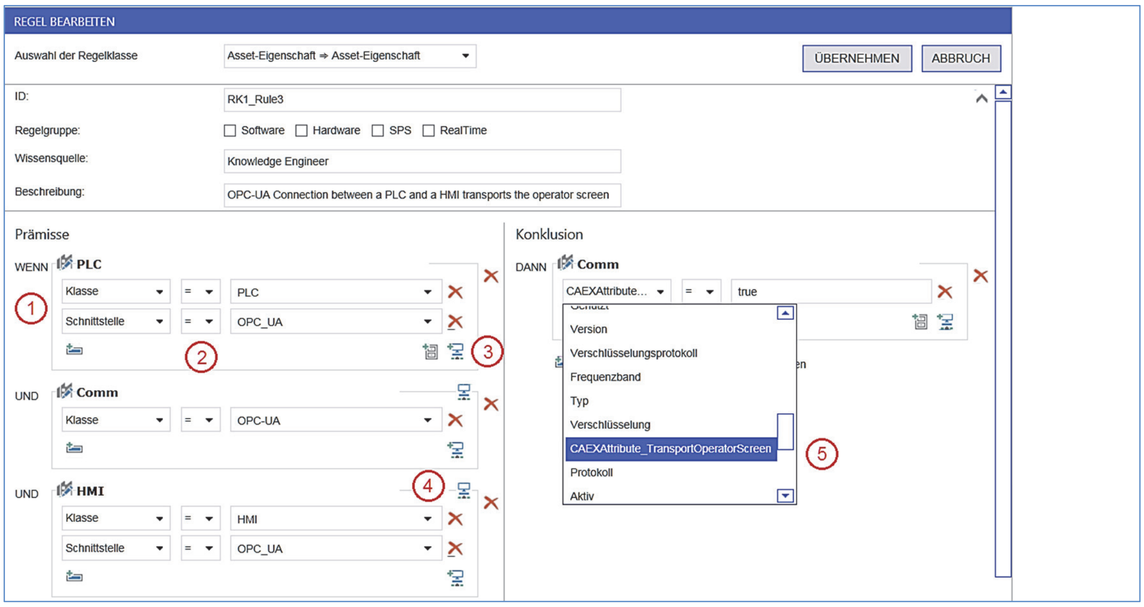
- Allow an intuitive creation of Rules which reference to a CAEX model.
- Ensure that the created rules are syntactical correct and formulated in the right way.
- Allow a visible connection between the variable, used as SWRL-Attribute and the CAEX Element.
- Reduce the needed knowledge about OWL/SWRL and CAEX when creating rules
- Provide all possible search elements and findable values to the user while creating the rule.
- Only allow to create permitted combinations of compare operators and address operators.

Following these requirements using a graphical user interface (GUI) for rule creation is the best solution to support the knowledge engineer. At this point we focus on rules just containing operators pointing to elements of the CAEX model. In our project we created rule parts for elements of the example OWL security ontology. Up to now an extension for variable knowledge from other knowledge domains given in OWL ontologies is undone. The GUI was developed for German small and medium-sized enterprises. For this reason, some wordings in the example view given in Figure 5, are shown in German.

To develop an intuitive interface for rule creation, it is necessary to show the *InternalElements* in a compact and sorted view, where all operators assigned to it stick together. This was developed in a way that one or more operators for one *InternalElement* can be added. This is shown on the left side (1) of Figure 5 were the antecedent elements of the described rule were shown. The elements are grouped by the *InternalElement* they represent. To prevent creation of ineligible combinations of compare operators and address operators, the compare operators are provided based on the address operator, selected before (2). For example: After choosing `"Class”"` (in German `"Klasse"` in Figure 5) only the permitted compare operators are selectable.

Especially the operators for checking *InternalLinks* and the hierarchy of *InternalElements* are complex. Larger rules to check complex structures with more than two *InternalElements* make rules more complex and hard to understand. To mitigate this problem on the lower right side of an *InternalElement* (3) two buttons allow an intuitive generation of these structures. The existing structures are shown on the upper right side (4) of an *InternalElement*. This is shown at the assets `"Comm"` and `"HMI"`.

It is an important requirement to use elements that can be inferred only. This prevents rules from pointing to elements not contained in the CAEX model. For solving this problem we assume that the parts of the CAEX model belonging to the domain knowledge fulfil all requirements which the



Auswahl der Regelklasse: Asset-Eigenschaft ⇒ Asset-Eigenschaft

ID: RK1_Rule3

Regelgruppe: ☐ Software ☐ Hardware ☐ SPS ☐ RealTime

Wissensquelle: Knowledge Engineer

Beschreibung: OPC-UA Connection between a PLC and a HMI transports the operator screen

Prämisse

WENN PLC
Klasse = PLC
Schnittstelle = OPC_UA

UND Comm
Klasse = OPC-UA

UND HMI
Klasse = HMI
Schnittstelle = OPC_UA

Konklusion

DANN Comm
CAEXAttribute... = true

Version
Verschlüsselungsprotokoll
Frequenzband
Typ
Verschlüsselung
CAEXAttribute_TransportOperatorScreen
Protokoll
Aktiv

Figure 5: Knowledge Engineers view for rule creation.

knowledge engineer needs to create the rules. Otherwise these elements must be added to the libraries (*SystemUnitClassLibrary*, *RoleClassLibrary*, *InterfaceClassLibrary*). Before starting to create rules, these libraries are searched for all possible elements. Only these elements can be chosen in the drop-down fields. This is shown in Figure 5 on the right side (5) where the attribute `"TransportOperatorScreen"` is chosen as part of the consequent (German: `"Konklusion"`) of the rule.

To keep the knowledge base decidable, only rules using existing individuals from the ontology in both body and head are used. These are so called "dl-safe-rules" (Staab and Studer, 2009). Based on an OWL DL ontology, a reasoner is able to infer new knowledge in a finite time. In INSA this allows to infer protection measures for given assets. For keeping our ontology decidable in this view we enabled our knowledge engineering component to add all used operators as object respectively data properties to our ontology. The object properties are used to point to existing individuals and the data properties point to the CAEX model using Literals.

By the described system a GUI for the knowledge engineer was designed. This GUI is easy to learn and enables the knowledge engineer to create rules in short terms. In first tests knowledge engineers of the domain of security knowledge can use the GUI after half an hour of introduction. Depending on the complexity of the rule to create, they needed up to a few minutes to create it.

Parts of the knowledge base had to be created using Protégé. After the implementation and usage of the GUI the experts found it easier and more comfortable to create new rules or find and edit existing rules. They needed up to 50% less time for the creation of new rules especially for complex rules.

5 VALIDATION & CONCLUSION

To validate our concepts we created a small ontology containing out of approximately 50 threats and protection measures represented in OWL and 20 Rules given in SWRL. We applied this knowledge base to the CAEX Model representing the automation system shown in Figure 1 to identify the appropriate threats and protection measures for this automation system. In the result a security concept based on the knowledge base and the CAEX Model was generated in less than a second. We proofed, that basing on the given knowledge the security concept was generated correct. In a further step a much larger knowledge base with 200 threats and protection measures and nearly 400 rules was created by security experts. The much more complex security concepts generated with this knowledge base are currently reviewed by security experts to validate their correctness.

Up to now we have shown a concept for connecting engineering data given in CAEX with security knowledge represented in OWL by using

SWRL rules with specific operators. Additionally we presented a user concept for an easy creation of these specific rules basing on the domain knowledge from the CAEX model. In addition to that we have shown a current implementation and the feasibility of main parts from the described concept.

Intensive testing is performed to identify possible optimization potential. In a further step, we plan to extend our concept to other engineering tasks. It is also necessary to enable the GUI to access variable OWL ontologies as a basis for the implementation in other tools (for example the editor for AutomationML files). With such an implementation, engineers can easily create checks, supplements or manipulation to large CAEX files with just creating some rules. This supports the execution of extensive and error-prone tasks, making them more easy and manageable.

ACKNOWLEDGEMENTS

The authors would like to thank the Federal Ministry of Economic Affairs and Energy (BMWi) and the participating partners in the INSA project for their funding and support.

REFERENCES

Vogel-Heuser, B., Diedrich, C., Fay, A., Jeschke, S., Kowalewski, S., Wollschlaeger, M., Göhner, P., 2014, Challenges for Software Engineering in Automation. *Journal of Software Engineering and Applications*, 7, pp. 440-451.

Frank, T., Eckert, K., Hadlich, T., Fay, A., Diedrich, C., Vogel-Heuser, B, 2012, Workflow and decision support for the design of distributed automation systems. IEEE Int. Conf. on Industrial Informatics (INDIN), Beijing, China.

Strube, M., Runde, S., Figalist, H., Fay, A.: Risk Minimization in Modernization Projects of Plant Automation – a Knowledge-Based Approach by means of Semantic Web Technologies, IEEE International Conference on Emerging Technologies and Factory Automation, 2011

Runde, S., Fay, A., 2011, Software Support for Building Automation Requirements Engineering - An Application of Semantic Web Technologies in Automation, IEEE Transactions on Industrial Informatics, Volume 7, Issue, 4 pp. 723-730.

Giarratano, J.C., Riley, G.D., 2005, *ExpertSystems – Priciples and Programming,* Course Technology, Thomson Learning, Boston

Legat, C., Lamparter, S., Vogel-Heuser, B., 2013, Knowledge-Based Technologies for Future Factory Engineering and Control In: *Service Orientation in Holonic and Multi Agent Manufacturing and Robotics 2013,* Springer, Berlin

Russel, S., Norvig, P., 2010, *Artifical Intelligence: A modern approach*, Pearson, Boston

Fulcher, J., Jain, Lakhmi, 2008, *Computational Intelligence: A Compendium*, Springer-Verlag

Schmidberger, T., Fay, A., 2007, A rule format for industrial plant information reasoning, IEEE Int. Conf. on Emerging Technologies and Factory Automation (ETFA), Patras, Greece.

Güttel, K., Weber, P., Fay, A., 2008, Automatic generation of PLC code beyond the nominal sequence, IEEE Int. Conf. on Emerging Technologies and Factory Automation (ETFA), Hamburg, Germany.

Christiansen, L., Fay, A., Opgenoorth, B., Neidig, J., 2011, Improved Diagnosis by Combining Structural and Process Knowledge, IEEE Int. Conf. on Emerging Technologies and Factory Automation (ETFA), Toulouse, France.

Schreiber, S., Schmidberger, T., Fay, A., May, J., Drewes, J., Schnieder, E., 2007, UML-based safety analysis of distributed automation systems, IEEE Int. Conf. on Emerging Technologies and Factory Automation (ETFA), Patras, Greece.

IEC 62443-2-1, 2010, Security for industrial automation and control systems - Establishing an industrial automation and control system security program, IEC, www.iec.ch

Valenzano, A., 2014, Industrial Cybersecurity – Improving Security Through Access Control Policy Models, In *IEEE Industrial Electronics Magazine June 2014*, IEEE

Antoniou, G., van Harmelen, F., 2012, A semantic web primer. MIT Press, Cambridge, Mass.

Runde, S., Dibowski, H., Fay, A., Kabitzsch, K., 2009, Semantic Requirement Ontology for the Engineering of Building Automation. IEEE Int. Conf. Emerging Technologies and Factory Automation (ETFA), Mallorca, Spain.

Linnenberg, T., Mueller, A. W., Christansen, L., Seitz, C., Fay, A., 2013, OntoEnergy - A lightweight ontology for supporting energy efficiency tasks. Int. Conf. Knowledge Engineering and Ontology Development (KEOD), Faro, Portugal.

World Wide Web Consortium, 2014, OWL. Web Ontology Language. http://www.w3.org/2001/sw/wiki/OWL

Donner, M., 2003, Toward a security ontology, In *IEEE Security and Privacy*, IEEE

Ekelhart, A., Fenz, S., Klemen, M., Weippl, E., 2007, Security Ontologies: Improving Quantitative Risk Analysis, In *IEEE Proceedings of the 40th Hawaii International Conference on System Sience,* IEEE

World Wide Web Consortium, 2014, SWRL. A Semantic Web Rule Language Combining OWL and RuleML, http://www.w3.org/Submission/SWRL/

IEC 62424, 2008, IEC 62424:200x - Specification for Representation of process control engineering requests in P&I Diagrams and for data exchange between

P&ID tools and PCE-CAE

Runde, S., Fay, A., Wutzke, W.-O., 2009, Knowledge-based requirement-engineering of building automated systems by means of semantic web technologies, In *IEEE International Conference on Industrial Informatics (INDIN),* Cardiff, U.K.

Abele, L., Legat, C., Grimm, S., Müller, A., 2013, Ontology-based Validation of Plant Models. In *IEEE 11th International Conference,* IEEE

Staab, S., Studer, R., 2009, Handbook on ontologies. International handbooks on information syste*ms*. Springer, Berlin.

AutomationML, 2015, AML-Engine version 3.1, https://www.automationml.org/o.red/uploads/dateien/1427889211-AutomationML%20Engine_v3.1.zip, last checked 06.07.2015.