# An IEC 62443-security oriented domain specific modelling language

Jolahn Vaudey, Stéphane Mocanu, Gwenaël Delaval, Eric Rutten

# An IEC 62443-security oriented domain specific modelling language

Jolahn Vaudey Stéphane Mocanu
Gwenaël Delaval Eric Rutten
*Univ. Grenoble Alpes, CNRS, Inria, Grenoble INP*, LIG, 38 000 Grenoble, France*
**Institute of Engineering Univ. Grenoble Alpes*
firstname.lastname@inria.fr

***Abstract*—As the historically isolated industrial control systems become increasingly connected, the threat posed by cyberattacks soars. To remedy this issue, industrial standards dedicated to the cybersecurity of ICS have been developed in the last twenty years, namely the IEC 62443 series. These standards provide guidelines to the creation and maintenance of a secure ICS, from the concept phase to its eventual disposal. This standard notably assume a specific Zone/Conduit model for systems, as a basis for building the security program. This model currently lacks computer-aided design tools, which are essential to the adoption of a standard. In this paper, we will present a domain specific modeling language, able to describe IEC 62443 compliant systems. Our main contributions are the DSL's syntax, which tries to formalize the informal model found in the standard, and the validation rules applied to it that ensure the described installations are secure by design, according to a set of hypotheses.**

## I. Introduction

### A. Industrial control systems

Industrial control systems (ICS) are networked systems composed of hardware and software components that control a physical industrial process. Typically, ICS are implemented by either Supervisory Control And Data Acquisition (SCADA) systems, or Distributed Control Systems (DCS). The physical processes' control loops are usually executed by Programmable Logic Controllers (PLC), rugged computers commonly used in the industry. In both SCADA and DCS, most of the control is delegated to these PLC, and collected data is centralized mostly to monitor the system. Typically, SCADA is used when monitoring multiple geographically distant plants, and DCS on critical plants that need real time monitoring, but their functionalities tend to converge nowadays.

### B. Growing cybersecurity concerns

For historical reasons, these systems communicate internally using mostly insecure protocols, such as Modbus, and for a long time this was not an issue as accessing the system required physical on-site presence. However, ICS nowadays are increasingly connected, and this has lead to the rise of security concerns and cyberattacks [McLaughlin et al.(2016)], including the well-known Stuxnet worm in 2010. These high profile attacks led to a growing awareness of these security issues, and research about attack detection, prevention or mitigation in ICS is now a hot topic. Compared to traditional Information Technology (IT) systems, Operational technology (OT) ones such as ICS have different security priorities. In the CIA (Confidentiality, Integrity, Availability) triad, ICS need availability first and foremost, rather than confidentiality in typical IT systems. Indeed, as a physical process is managed, a control loss can cause dramatic results, both financial, if the attacker manages to break key components, and potentially life threatening. To avoid such loss of control, systems should be as resilient to such attacks as possible, both by reducing the attack surface using a secure architecture and by detecting and reacting to security breaches.

### C. Cybersecurity standards

The main standard concerning the security of information systems is ISO/IEC 27001. It is widely adopted across the industry, and tries to encompass all aspects of information security, which naturally includes cybersecurity. This standard is associated with a certification, which needs to be maintained by passing regular review and audits proving that the security measures taken evolve alongside the threat landscape. While most large companies possess information systems that are critical to their business, the certification is mostly used by pure IT enterprises [Mirtsch et al.(2020)]. As for ICS, their

specificities evoked in the previous paragraph have led to the creation of specific standards, derivative of ISO/IEC 27001.

IEC 62443 is such a set of standards dedicated to the cybersecurity of Industrial Automation and Control Systems (IACS), which can be seen as a specialized expansion of ISO/IEC 27001. The standards try to encompass all possible components of IACS, including ICS, be they SCADA or DCS, which entails PLCs, Remote terminal units (RTU), sensors, actuators, monitoring/diagnostic systems, Human-Machine interfaces (HMI), for example. Guidelines are provided for the creation and maintenance of secure ICS, ranging from the initial concept phase up to their recycling or disposal. IEC 62443 employ a specific Zone/Conduit model for ICS, which will be described in more details in Section II.

As of now, there is a lack of tools dedicated to modeling the architecture of IEC 62443 compliant systems in their finished state, once the zone partitioning and subsequent risk assessments are done. While this kind of representation comes after the initial conception, and therefore cannot be used in the design phase, it would allow for easier, or even automated, modifications of the architecture. This paper will showcase our contribution, a domain specific modeling language (DSL) describing instances of ICS in this standard's Zone/Conduit model. Section III will present this DSL's conception and implementation, while in Section IV an example ICS will be described, and modeled using the DSL.

## II. IEC 62443

As we aim to describe IEC 62443 compliant systems, we will mostly be interested in its first part, 1-1 [IEC(2009)], that defines concepts and models used in the rest of the standards. Part 3-2, which gives requirement for the design and risk assessment phase of an IACS creation, and Parts 3-3 and 4-2, which details requirements pertaining to security levels, were also used to further define the domain model the DSL describes, and their relevant parts will thus be presented.

### *A. Part 1-1: Terminology, concepts and models*

This first document contains the definitions of terms and abbreviations used in the rest of the standard, a description of the current situation in ICS and overall trends, OT cybersecurity concepts, and several models used to represent industrial control systems. General guidelines are given for a Threat-risk assessment, including lists of typical risks and threats. The standard recommends a cyclic approach to the development of the security management system, rather than a project-based one, on the basis that the threat landscape evolves at a quick pace, and thus security policies in place needs to adapt constantly. In this article, we will mostly be interested in describing systems that follows the assumed architecture in IEC 62443, a zone/conduit model. We will now describe the involved concepts.

*1) Zones and conduits:* High security needs imply restrictive policies, and it is usually not feasible to apply them to the complete system. Thus, the standard assumes that components that have the same security requirements are grouped together in security zones.

- The components within a zone can be physical devices, informational assets, or applications.
- A zone can be physical if the assets share a physical location, or it can be a purely virtual construct.
- A zone can be considered trusted, or untrusted (if it connects to the outside, typically).
- We will assume that components are able to communicate freely within a zone, as long as this communication respects the security requirements.
- A zone may contain sub-zones, which must meet the security requirements of their parent zone. If a component needs to communicate with an asset outside the zone, this communication needs to pass through a conduit.

Conduits are technically zones, set of components sharing security requirements, but as they are used to allow communications between zones, they are additionally defined by the set of zones they link.

- Conduits can be physical/virtual and trusted/untrusted, dedicated to communications.
- Conduits can be contained within a zone, but they cannot contain sub-zones themselves.
- Conduits also contain a set of channels, which are the communications links established within it. They are compared to cables (channels) within a pipe (conduit) in the standard. These channels are defined by a set of components they link. A channel inherits the security requirements of its parent conduit. It can be considered trusted/untrusted by the zones it connects, if this security level is sufficient/insufficient compared to the zone's target level.
- Untrusted communication needs to be checked in some manners before acceptation.

Figure 1 gives a representation of such an ICS, divided into three layers: an enterprise network connected to the internet, a more secure demilitarized zone (DMZ)

that filters unwanted communication and hosts critical services, and two zones dedicated to supervisory control, that contains PLCs and HMIs, and manage the physical processes. The defense in depth objective of IEC 62443 is here in plain sight: to interfere with the physical process, a remote attacker has to go through multiple layers of firewalls, all with different security rules.

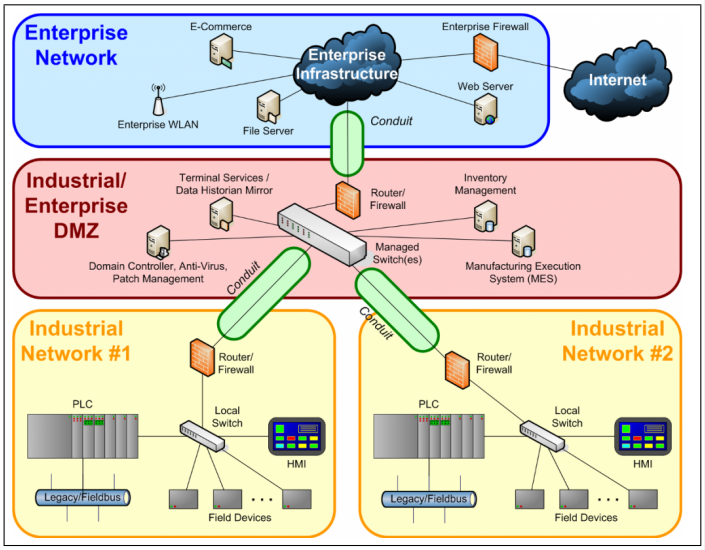


Fig. 1. Example of an ICS using the Zone/Conduit model. From Cisco blogs [Amirault(2021)].

*2) Security levels:* In this first part of the standard, security levels (SLs) are defined as a qualitative description of a zone security. The initially proposed definition gives a single security level for a zone, ranging from 1 to 3 and corresponding to a low/medium/high security. This is very vague, and we prefer another definition, used in more recent parts [IEC(2013)], [IEC(2019)] of the standard for the implementation. In this second definition, a security level ranges from 0 to 4, with 0 meaning that no specific protection is implemented, 1 that the system is protected against accidents, and 2 to 4 that the system is able to defend itself from small/medium/large scale attacks. Usually, we will use vectors of seven security levels, each corresponding to a foundational requirement (FR). These FRs are a refinement of the CIA model specific to IACS, defined in Figure 2. Assets and zones are given security levels with different meanings, which are described in Figure 3.

For zones and conduits to reach their target security level, the passive capability of their components may not be enough. In this case, countermeasures are added to raise the achieved security level. Those can be technical (devices or software), administrative (usage policies) or physical (locked areas).

- **Access Control (AC)**: Control access to selected devices, information or both to protect against unauthorized interrogation of the device or information.
- **Use Control (UC)**: Control use of selected devices, information or both to protect against unauthorized operation of the device or use of information.
- **Data Integrity (DI)**: Ensure the integrity of data on selected communication channels to protect against unauthorized changes.
- **Data Confidentiality (DC)**: Ensure the confidentiality of data on selected communication channels to protect against eavesdropping.
- **Restrict Data Flow (RDF)**: Restrict the flow of data on communication channels to protect against the publication of information to unauthorized sources.
- **Timely Response to Event (TRE)**: Respond to security violations by notifying the proper authority, reporting needed forensic evidence of the violation, and automatically taking timely corrective action in mission-critical or safety-critical situations.
- **Resource Availability (RA)**: Ensure the availability of all network resources to protect against denial of service attacks.

Fig. 2. Foundational requirements definitions ([IEC(2009)])

For Zones, Conduits and Components, different security level vectors are defined:

- **Target SL**: The actual security requirement for the zone/conduit, according to the risk assessment.
- **Capability SL**: Defined for countermeasures and components within a zone/conduit, and measure how much it contributes to the security of the zone/conduit.
- **Achieved SL**: The current security level applied in the zone/conduit, computed using the inherent security properties of components and the countermeasures applied, and potentially other factors. This should always be kept equal or above the target SL.

Fig. 3. Security level vectors definitions

### *B. Part 3-2: Security risk assessment for system design*

Part 3-2 [IEC(2020)] of the standard defines zone conduit requirements (ZCR) for the design process of an IACS, which includes defining boundaries for the system under consideration (ZCR 1), partitioning it into zones and conduits (ZCR 3), performing risk assessments (ZCR 2, 4 and 5) and subsequently establishing

their target security level. While most of this process lies outside of the scope of system description, this part also explains how the security requirements, assumptions and constraints should be documented (ZCR 6), and therefore includes additional fields used to describe zones and conduits.

On top of what was already presented in previous parts, the following information should be added to zones and conduits:

- A list of accountable organizations
- A definition of the logical (and if applicable physical) boundaries
- Whether or not the zone contains safety assets, or is itself safety related.
- A list of access points (Doors, fences, ...) to the zone's physical location
- A list of data flows coming in and out of the zone. This is advised to be a list of tuple, each of the form (Source, Destination, Protocol).
- A list of applicable security policies.
- A description of assumptions and external dependencies, that are deemed necessary to attain the target security level.

### C. Parts 3-3 and 4-2: Security requirements

*1) System requirements:* Part 3-3 [IEC(2013)] of the standard defines the translations between qualitative foundational requirements security levels and concrete system requirements (SR) that need to be implemented in zones. These system requirements are divided, just like security levels, by foundation requirements, as shown in Figure 2. For each of these FRs, a table is given, explaining which measure needs to be in place to achieve a given security level (From SL1 to SL4).

*2) Component requirements:* In a similar manner, part 4-2 [IEC(2019)] defines the translations between qualitative foundational requirements security levels and concrete component requirements (CR) that need to be implemented in corresponding components. Some of these requirements apply to all components in a zone, while others are specific to certain kind of components. The document denotes four different assets archetypes, those being:

- Software applications
- Embedded devices
- Host devices
- Network devices

Typically, while system requirements are implemented while deploying the zone, CRs should be enforced when designing components, determining their inherent security capability.

## III. Domain specific language

### A. Motivation

IEC 62443 is a relatively recent and still evolving standard, with its first part being published in 2009 and the current last part in 2023. Along with the fact that sharing an ICS architecture currently in use is innately a risk, the consequence is that no examples of real zone/conduit architectures are shared online. Most instances of IEC 62443 systems found on the internet are merely illustrations of the standard's concepts, and none of them show both the architecture and the security levels at the same time, while both are fundamental to the norm. Several models can be found in the literature concerning different parts of the design process found in IEC 62443.

Some of them are used to perform the threat analysis found in part 3-2 [IEC(2020)], with [Da Silva et al.(2023)] and [Fockel et al.(2019)] using the Microsoft modeling tools and STRIDE, a model used to identify security threats, to do so. The former article proposes a database of templates for components commonly found in ICS and tries to automatically populate it based on configuration files, while the second describes a manual methodology that must be applied by developers and security experts and proposes a modified threat template compliant with IEC 62443. In a similar vein, the method found in [Eckhart et al.(2022)] performs multiple steps of the risk assessment also found in part 3-2, generates attack graphs and validates the zone partitioning. They propose a model written in AutomationML using an extendable library that allows for cybersecurity semantics to be defined.

Other models are used to check that security constraints are met. For instance, [Kulik et al.(2019)] uses a labeled transition system model, representing the studied system behaviour, and applies model checking techniques to verify that system requirements found in part 3-3 [IEC(2013)] are met. On the other hand, [Ehrlich et al.(2019)] proposes a TOSCA (Topology and Orchestration Specification for Cloud Applications) model, and ensures that system requirements are met whenever the supervised ICS is reconfigured. Both of these models are specific to cloud-based ICS. For component requirements found in part 4-2 [IEC(2019)], [Göttel et al.(2023)] presents a survey of available tools, but they do not propose a specific model.

There is however a lack of publicly available tool dedicated to modeling such systems in their finished state, once the zone layout and security levels are devised. This is where our contribution resides, after the threat/risk analysis, and the corresponding models mentioned, and at a higher abstraction level which uses the generic zone/conduit model described in the standard, as opposed to the mentioned model checking techniques.

This contribution consists of a domain specific language (DSL), capable of modeling systems divided in Zones/Conduits, as described in the standard. This model can be read and modified programmatically, and we initially created it to serve as a base for our use case, control reconfiguration. We will now go into the details of this language, starting with some reminders about DSL technologies.

### *B. Domain specific languages and model driven engineering*

Domain specific languages are modeling or programming languages dedicated to a specific use case [Mernik et al.(2005)], as opposed to general-purpose languages. Typically, DSLs can be either external or internal. Internal (or Embedded) DSLs are defined as a subset of a general-purpose language, and therefore use the same syntax. They can be understood as a kind of API. On the other hand, external DSL have their own grammar, and therefore their own parser. We will prefer external DSLs, as the freedom in the syntax design allows for code that is easier to understand and write.

DSLs are often used under a greater paradigm, model-driven engineering (MDE) [Schmidt et al.(2006)]. This approach starts by defining specific domain models, and subsequently generating source code based on the model. This level of abstraction allows for faster development times, and ensures that the output code works according to the model's specifications, by design. In this approach, DSLs can be used to represent instances of the domain model, and their parser, which translates a text following their grammar into an instance of the domain model, is classically automatically generated from the desired grammar. In turn, the obtained instance can be used to generate code specific to the system it describes.

DSLs have already been created to model Industrial systems architecture. For instance, one has been developed to model a specific Industry 4.0 architecture, the Reference Architecture Model Industry 4.0 [Binder et al.(2019)].

### *C. Assumptions*

Some parts of the model used in the standard are vague, or even contradictory. For instance, in part 1-1[IEC(2009)], it is both written that "A conduit can have subconduits" (6.5.5), and that "a conduit is not made up of subconduits" (6.5.6). As such, we needed to make a few assumptions to fill in the blanks.

For this particular case, as shown in the previous parts, conduits will not be able to contain subconduits. On another note, we assume that applications need to be explicitly run on a device within their zone.

The achieved security level of a zone/conduit will only be computed using the capability of its components and countermeasures. More precisely, the achieved SL vector will be the pairwise maximum between :

- the pairwise minimum of all component capabilities
- the pairwise maximum of all countermeasures capabilities

This rule assumes that the potential security levels SL0, SL1, SL2, SL3 and SL4 behave as the integers 0,1,2,3 and 4 respectively. An example illustrating this computation can be found in Figure 4

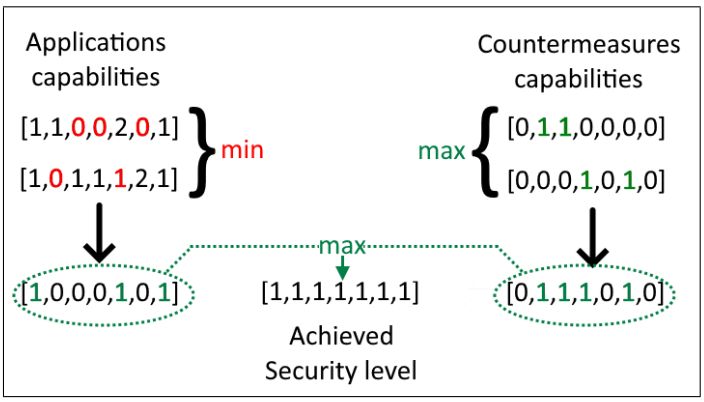


Fig. 4. Achieved security level vector computation

Theoretically, this achieved level also degrades over time, and depends on other parameters. As our model is a static representation, such evolutions should be taken into account by reducing the capabilities of obsolete components / countermeasures.

We also took inspiration from the security level model found in [Ehrlich et al.(2019)] that allows for some security level in capability vectors to be designated as Not Relevant (NR), which excludes them from the achieved level computation.

Finally, communication-wise, we suppose that components within a zone can freely communicate, as long as this communication respects the measures befitting of

the zone security levels. However, it is not assumed that sub-zones can innately communicate with their parent, it must be explicitly indicated via a conduit. Also, we assume that communications between components may only pass through at most one conduit.

### D. Implementation

The DSL has been implemented using Eclipse Modeling Framework (EMF) [Steinberg et al.(2008)] and Xtext [Eysholdt and Behrens(2010)], from the Eclipse suite. Both EMF and Xtext follow a model driven engineering paradigm, generating code based on a given model. EMF allows us to generate java classes representing a metamodel corresponding to the standard's by editing a graphical class diagram. This generated code can easily be modified and extended. Xtext takes a grammar and this java metamodel as input and generates a parser, which translates texts that follows the grammar into corresponding instances of the supplied metamodel. Xtext also generates a runtime integrated development environment (IDE) and several hooks that, for instance, allow verifying properties that are not purely grammatical by navigating the abstract syntax tree instead. Our model follows the abstract grammar found in Figure 5, which will be referenced in the rest of this section. This grammar is provided in a formalism akin to Backus-Naur, while ignoring most keywords used in the concrete implementation. This section will ignore the additional zone/conduit informal information found in Section II-B for the sake of brevity, as they do not change the shape of the model and are mostly implemented as lists of String.

*1) Conduits and zones:* An instance of the model contains a set of zones, which may either be "normal" zones, or conduits. Zones are identified by their names, which must be unique. A given zone also has a security level vector as a target, and contains a set of components, and a set of countermeasures. Classical zones may contain subzones (which may themselves be either classical zones or conduits), while conduits may contain channels. Additionally, zones are either trusted or untrusted, and physical or virtual, these properties being represented as two boolean fields.

*2) Components:* Components are either devices or applications. All components are identified by their name and possess a capability security vector. They may also possess a "needed" security level vector, which must always be pairwise lower than the target level of the zone the component resides in. To allow descriptions with more depth, the possibility to describe application dependencies and component attributes/constraints has been added.

To roughly represent critical interconnections, applications within the model have a field named "communicatesWith", that lists every other application or device that it needs to communicate with to run properly. According to hypotheses regarding communication, this means that all of these components must be within one conduit of the application. These communication needs are not necessarily symmetrical: Let's say we have two applications $a_1$ and $a_2$, if $a_1$ has $a_2$ listed as a dependency, $a_1$ cannot run if $a_2$ is not accessible, but $a_2$ can.

Every component in the model is also given a list of properties, or attributes (both terms will be used). These are String-value associations, expressed as $propName = value$, that can be used to give more details about the component. The values in question may be either boolean, integer, float or string constants. Applications are given a list of constraints, conditions that applies to the device running them. These constraints may either require that the running location possess a given property, regardless of its value/type, or that this attribute satisfies a comparison to a given value. This value can be a constant, one of the application's properties, or one of the device's. In any case, both the property and the value it is compared to must be compatible type-wise. The comparison operators are the usual ones, and strings are assumed to follow an alphabetical ordering.

Devices are further detailed by three boolean fields, indicating respectively whether they are a network device, a host device and/or an embedded device. It was assumed that these categories are not mutually exclusive. As of now, these are used in the process translating the security level to specific component security requirements.

*3) Countermeasures:* Countermeasures are identified by their names, and described by a capability security level vector. As a reminder, according to the hypotheses, countermeasures raise the achieved security levels of the zone they reside in up to their own level.

*4) Channels:* Channels are identified by their names, and defined by the set of at least two components that they link. These channels are used to specify the precise end points of conduits within the linked zones, and will mostly be ignored in the examples, as this granularity is not suited to small instances.

### E. Validation rules

Not all properties can be enforced on a grammar level, and the following ones are checked using validation rules

```
model ::= zone*
zone ::= normalZone | conduit
normalZone ::= zoneName purple"target" slevels
    component* countermeasure* purple"subZones"
    zone*
conduit ::= zoneName purple"connects" zoneName*
    purple"target" slevels component* countermeasure*
    channel*
component ::= device | app
counterMeasure ::= cntmName purple"capability"
    slevels
device ::= deviceName purple"capability" slevels
    property*
app ::= appName purple"run" purple"on" (purple"nothing"
    | deviceName) purple"capability" slevels
    (purple"needed" slevels)? constraint*
    componentName* property*
channel ::= channelName purple"connects"
    componentName*
componentName ::= deviceName | appName |
    systemName
slevels ::= SL[7]
SL ::= SL0 | SL1 | SL2 | SL3 | SL4
property ::= propName value
value ::= bool | int | float | string
constraint ::= purple"optional"? propName
                (operation (value | (purple"self" |
                    purple"ref") propName))?
operation ::= = | <> | < | <= | > | >=
```

Fig. 5. Abstract DSL grammar. "*" denote that the rule can be repeated any number of times, including 0. Words in bold are keywords relevant to the field they precede.

(which can be understood as static semantics), applied on the abstract syntax tree (AST) returned by the parser. We will now describe these, which implements properties and hypotheses that were previously mentioned.

The first category of validation rule verify that the system description is well-formed. Violating one of these rules would lead either to a nonsensical model, to ambiguities, or would give redundant information.

- Every component, zone, countermeasure and channel has a unique name, when compared to other objects of the same type anywhere in the model. As conduits are zones, there cannot be a classical zone and a conduit sharing the same name.
- For a given component, every property has a unique name. Different devices may, and probably should have properties with identical names.
- In every list, elements are never referred to twice. For instance, zones are not connected multiple times by the same conduit, and application do not have duplicate dependencies.
- Zones can't be subzones of themselves, and a subzone can't contain it's parent. This is checked recursively to avoid container loops.
- Conduits do not connect themselves, as this connection is implicit.
- Channels connects components that are part of the zones their parent conduit connects, or part of the conduit itself. A single channel connects at least two components.
- Every application run on a device in its own zone, or does not run.

The other validation rules concerns security-related properties.

- Target security level of subzones are always pairwise equal or higher when compared to their parent's. This comes from the standard, which indicates that subzones must abide by their parent's security needs.
- For each zone, the target security level must be compatible with the achieved one, based on its components and countermeasures. The achieved security level is computed according to the assumptions found in section III-C.
- For every application currently running, each non-optional constraint is met. This entails checking that the tested property is present, that the constraint is correctly typed and that the comparison returns true when computed.
- Every application is at most one conduit away from the components it communicates with, under the hypothesis that a communication cannot pass through more than one conduit.

Finally, some validation rules are optional, and only raise warnings when violated. Such warnings will be given when:

- A running application does not meet an optional requirement.
- An application is not running on any device.

### *F. DSL Functionalities*

*1) Textual IDE:* The textual IDE's interface is shown in Figure 6. It is automatically generated by xtext using the grammar description and the programmed validations rules. This is accessed through an eclipse runtime application when opening a .miec file.

On the figure, several functionalities can be seen. To help write the instances, the language's keywords are highlighted, and suggestions can be given: in the example, in the device being described, these suggestions indicates that the keyword "capability" must be entered next. The IDE also indicates grammatical and validation errors. In the example, the aforementioned device declaration is underlined in red, as the capability feature is missing, while it's presence is mandatory within the grammar rules. In zone B, a validation rule error is showcased: The target security level is not reached, as the application appB lacks sufficient capability. In this case, both the zone target level and the component are underlined. Finally, optional constraints raise a warning when they are not met, as seen with the constraint on the model, underlined in yellow.

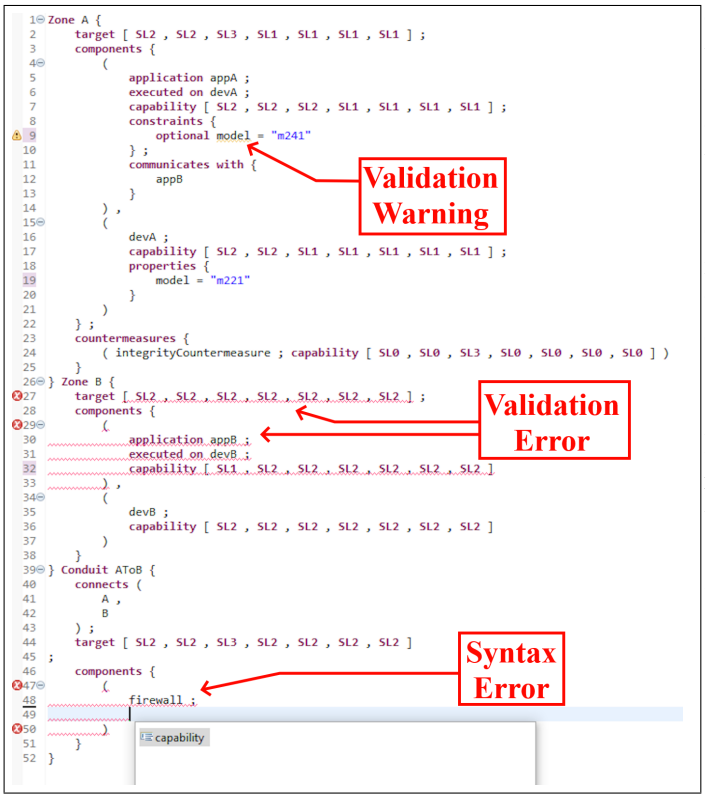


Fig. 6. Textual IDE illustration

*2) Graphical IDE:* The graphical IDE, designed using Sirius, can be used to visualize and edit DSL models. It is also accessed through the eclipse runtime application, by creating a representation file for a .miec file. The two zone example previously shown in textual form is represented in this graphical IDE in Figure 7.

The potential actions are shown on the right within a palette, they are used to add or edit elements within the model. They can be activated by drag-and-dropping them

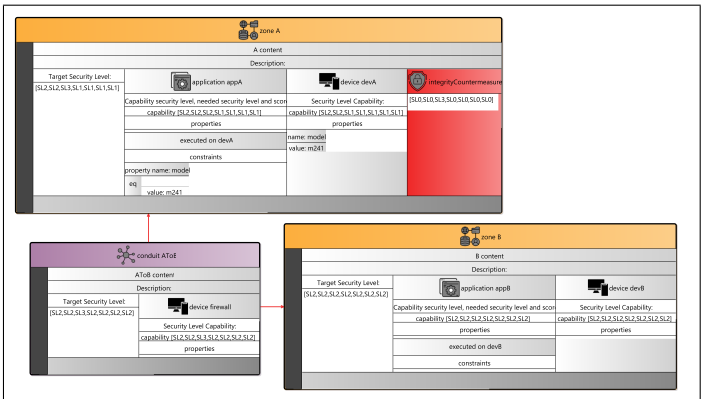

Fig. 7. Graphical IDE illustration

to corresponding areas, or by holding the mouse over the field you want to edit until their icons appears. In the example, two such icons appear on the "Zone A" at the top of the image, showing that one can add a sub-zone or a sub-conduit. The same effects could be obtained by drag and dropping these actions on Zone A. All such tools will not be enumerated, as they corresponds to the fields in the grammar found in Figure 5. The ability to copy-paste parts of the model is also implemented. All in all, the graphical IDE can be used interchangeably with the textual one. However, for clarity reasons, editing channels graphically is not supported.

*3) Security requirements:* Thanks to the parts of the standard evoked in section II-C, we are able to translate the security levels to security requirements. System requirements are given for each zone/conduit based on their target security level, while component requirements are given to each applications, and to each host, network or embedded devices, based on their capability.

## IV. Application

As an application, we will study a simple use case, and its translation to a DSL instance.

### A. System studied

We study a heavily simplified industrial control system, supervising a schematic chain of production. The system is divided into two parts, the first one manufacturing raw materials, and the second one sorting them based on their color. Both parts are controlled by a dedicated PLC implementing the local loop , and these two PLCs are supervized by a third one, that controls a Human-machine interface (HMI) and communicates with the rest of the (hypothetical) system. The two PLCs managing a physical process do not directly communicate.

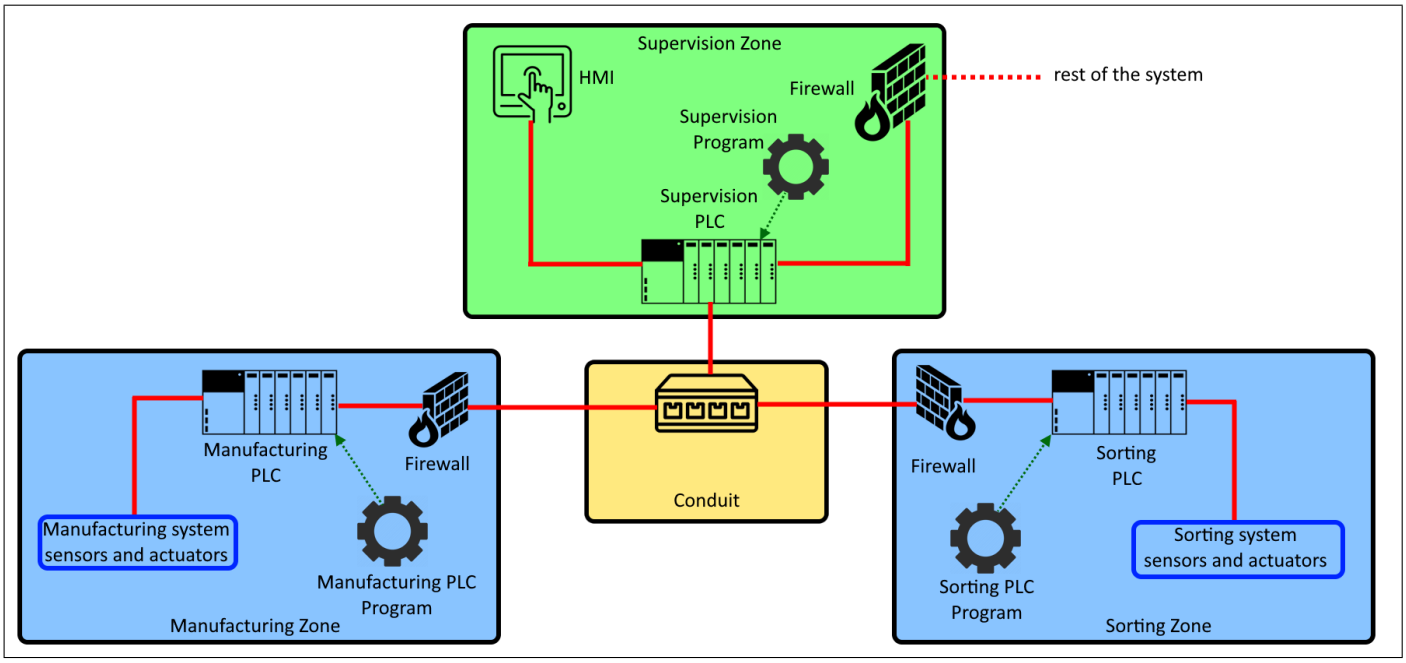


Fig. 8. Schema representing the example system as a Zone/conduit model

### B. Zone/conduit model

By design, this system lends itself well to a zone-based model. It is easily split into three zones: One for each part of the physical process, containing the local PLC and it's program, sensors and actuators, and one for the supervision that contains a PLC and the HMI. Additionally, there needs to be a conduit connecting these zones, and we will assume one links all three. This conduit will contain two channels, one linking the supervision to the manufacturing zone, and one to the sorter. A graphical view of this model is found in Figure 8.

Concerning foundational requirements, found in Figure 2, we will define the zones' targets for the example's sake. The first two zones will need a level of SL1 in Access Control and Use Control, restricting who can access it and which command is acceptable, and a level of SL2 in Timely Response to Events, as reacting quickly is a priority when managing a physical system. FRs related to confidentiality or integrity are assumed as not critical for low level data produced in these zones, and protection against DOS attacks unnecessary. For the supervision zone, access and use control will likewise need a level of SL1, and in this case data confidentiality as well. These security levels cannot be achieved by the PLCs alone, and in this example, firewalls are present as a countermeasure to filter accesses.

### C. DSL instance

This Zone/Conduit model is easily translated to the DSL format, which shares its architecture. The DSL instance can be found in Figure 9. Note that the conduit does not enforce any security level, as it only contains a switch, and is thus considered untrusted by all zones. This means that any information passing through it must be checked, by the firewalls for zones managing a process here, and by the PLC for the supervision. There are no errors or warnings reported, meaning the model is correct both syntactically, and validation-wise.

```
Zone Manufacturing {
    target [SL1,SL1,SL0,SL0,SL0,SL2,SL0];
    components {
        (
            device manufacturing_plc;
            capability [SL0,SL1,SL0,SL0,SL0,SL2,SL0]
        ),
        (
            application manufacturing_program;
            executed on manufacturing_plc;
            capability [SL0,SL1,SL0,SL0,SL0,SL2,SL0];
            communicates with {supervision_program}

        )

    };
    countermeasures {
        (
            manufacturing_firewall;
            capability [SL1,SL1,SL0,SL0,SL0,SL0,SL0]
        )
    }

}



Zone Supervision {
    target [SL1,SL1,SL0,SL1,SL0,SL0,SL0];
    components {
        (
            device supervision_plc;
            capability [SL0,SL1,SL0,SL0,SL0,SL0,SL0]
        ),
        (
            device supervision_hmi;
            capability [SL0,SL0,SL0,SL0,SL0,SL0,SL0]
        ),
        (
            application supervision_program;
            executed on supervision_plc;
            capability [SL1,SL1,SL0,SL0,SL0,SL2,SL0]
        )

    };
    countermeasures {
        (
            supervision_firewall;
            capability [SL1,SL1,SL0,SL1,SL0,SL0,SL0]
        )
    }

}
```

```
Zone Sorting {
    target [SL1,SL1,SL0,SL0,SL0,SL2,SL0];
    components {
        (
            device sorting_plc;
            capability [SL0,SL1,SL0,SL0,SL0,SL2,SL0]
        ),
        (
            application sorting_program;
            executed on sorting_plc;
            capability [SL1,SL1,SL0,SL0,SL0,SL2,SL0];
            communicates with {supervision_program}
        )

    };
    countermeasures {
        (
            sorting_firewall;
            capability [SL1,SL1,SL0,SL0,SL0,SL0,SL0]
        )
    }

}



Conduit LinkToSupervision {
    connects (
        Manufacturing,
        Sorting,
        Supervision
    );
    target [SL0,SL0,SL0,SL0,SL0,SL0,SL0];
    components {
        (
            switch;
            capability [SL0,SL0,SL0,SL0,SL0,SL0,SL0]

        )
    };
    channels {
        (
            supervision_manufacturing;
            connects {
                supervision_plc,
                manufacturing_plc
            }
        ),
        (
            supervision_sorting;
            connects {
                supervision_plc,
                sorting_plc
            }
        )
    }
}
```

Fig. 9. Textual DSL instance corresponding to the studied example

## V. Conclusion

In this paper, we have pointed to a lack of design tools for the models found in the cybersecurity standard for ICS IEC 62443. As we needed a formal representation of compliant systems for our use case, we designed and presented a new DSL, describing industrial control systems in a zone/conduit model compliant with the new standard IEC 62443. Instances can be edited using either a graphical or textual IDE, and are checked using validations rules that enforces semantic properties.

By itself, the DSL is descriptive, and does not have much functionalities beside creating instances. This creation process could be improved, potentially by adding predefined components, which could be derived from online devices catalogs. However, this enhancement would need readily available information about innate component security levels according to the IEC 62443 standard, which is not realistic at the moment. As for additional functionalities, models may be edited programmaticaly, allowing for ad-hoc expansions using the component attributes. Our main use case, for example, aims to design a reconfiguration controller allowing us to migrate programs away from compromised devices while sustaining a cyberattack. This controller is obtained by automatically deriving a constraint program, modeling the reconfiguration problem from the DSL description. The DSL models are modified by this reconfiguration, and constraints and properties are used to restrict where applications can be executed.

As for immediate perspectives, part 3-2 of the standard indicates that a zone and conduit drawing on the system should be produced to illustrate the partitioning, and producing it should be easy to automate based on the DSL description.

### A. Code availability

The source code can be found in an archive [Vaudey(2024)]. We aim to find better ways to share it, by exporting the eclipse installation alongside the source code.

## References


[Amirault(2021)] Antoine Amirault. 2021. Cisco Blog, IEC 62443 Post. https://blogs.cisco.com/security/securing-industrial-networks-what-is-isa-iec-62443. Accessed: 2024-04-11.

[Binder et al.(2019)] Christoph Binder, Christian Neureiter, Goran Lastro, Mathias Uslar, and Peter Lieber. 2019. Towards a standards-based domain specific language for industry 4.0 architectures. In *Complex Systems Design & Management: Proceedings of the Ninth International Conference on Complex Systems Design & Management, CSD&M Paris 2018*. Springer, 44–55.

[Da Silva et al.(2023)] Mike Da Silva, Maxime Puys, Pierre-Henri Thevenon, Stéphane Mocanu, and Nelson Nkawa. 2023. Automated ICS template for STRIDE Microsoft Threat Modeling Tool. In *ARES 2023 - 18th International Conference on Availability, Reliability and Security*. Benevento, Italy, 1–7. https://doi.org/10.1145/3600160.3605068

[Eckhart et al.(2022)] Matthias Eckhart, Andreas Ekelhart, and Edgar Weippl. 2022. Automated Security Risk Identification Using AutomationML-Based Engineering Data. *IEEE Transactions on Dependable and Secure Computing* 19, 3 (2022), 1655–1672. https://doi.org/10.1109/TDSC.2020.3033150

[Ehrlich et al.(2019)] Marco Ehrlich, Henning Trsek, Martin Gergeleit, Julius Paffrath, Kostyantyn Simkin, and Jürgen Jasperneite. 2019. Secure and Flexible Deployment of Industrial Applications inside Cloud-Based Environments. In *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. 1269–1272. https://doi.org/10.1109/ETFA.2019.8868978

[Eysholdt and Behrens(2010)] Moritz Eysholdt and Heiko Behrens. 2010. Xtext: implement your language faster than the quick and dirty way. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*. 307–309.

[Fockel et al.(2019)] Markus Fockel, Sven Merschjohann, Masud Fazal-Baqaie, Torsten Förder, Stefan Hausmann, and Boris Waldeck. 2019. Designing and Integrating IEC 62443 Compliant Threat Analysis. In *Systems, Software and Services Process Improvement*, Alastair Walker, Rory V. O'Connor, and Richard Messnarz (Eds.). Springer International Publishing, Cham, 57–69.

[Göttel et al.(2023)] Christian Göttel, Maëlle Kabir-Querrec, David Kozhaya, Thanikesavan Sivanthi, and Ognjen Vuković. 2023. Qualitative Analysis for Validating IEC 62443-4-2 Requirements in DevSecOps. In *2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 1–8.

[IEC(2009)] International Electrotechnical Commission IEC. 2009. *IEC 62443 Part 1-1: Terminology, concepts and models*. Technical Report. International electronical commission. https://webstore.iec.ch/preview/info_iec62443-1-1%7Bed1.0%7Den.pdf

[IEC(2013)] International Electrotechnical Commission IEC. 2013. *IEC 62443 Part 3-3: System security requirements and security levels*. Technical Report. International electronical commission. https://webstore.iec.ch/preview/info_iec62443-3-3%7Bed1.0%7Db.pdf

[IEC(2019)] International Electrotechnical Commission IEC. 2019. *IEC 62443 Part 4-2: Technical security requirements for IACS components*. Technical Report. International electronical com-

mission. https://webstore.iec.ch/preview/info_iec62443-4-2%7Bed1.0%7Db.pdf

[IEC(2020)] International Electrotechnical Commission IEC. 2020. *IEC 62443 Part 3-2: Security risk assessment for system design*. Technical Report. International electronical commission. https://webstore.iec.ch/preview/info_iec62443-1-1%7Bed1.0%7Den.pdf

[Kulik et al.(2019)] Tomas Kulik, Peter W. V. Tran-Jørgensen, and Jalil Boudjadar. 2019. Compliance verification of a cyber security standard for Cloud-connected SCADA. In *2019 Global IoT Summit (GIoTS)*. 1–6. https://doi.org/10.1109/GIOTS.2019.8766363

[McLaughlin et al.(2016)] Stephen McLaughlin, Charalambos Konstantinou, Xueyang Wang, Lucas Davi, Ahmad-Reza Sadeghi, Michail Maniatakos, and Ramesh Karri. 2016. The cybersecurity landscape in industrial control systems. *Proc. IEEE* 104, 5 (2016), 1039–1057.

[Mernik et al.(2005)] Marjan Mernik, Jan Heering, and Anthony M Sloane. 2005. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)* 37, 4 (2005), 316–344.

[Mirtsch et al.(2020)] Mona Mirtsch, Jan Kinne, and Knut Blind. 2020. Exploring the adoption of the international information security management system standard ISO/IEC 27001: a web mining-based analysis. *IEEE Transactions on Engineering Management* 68, 1 (2020), 87–100.

[Schmidt et al.(2006)] Douglas C Schmidt et al. 2006. Model-driven engineering. *Computer-IEEE Computer Society-* 39, 2 (2006), 25.

[Steinberg et al.(2008)] Dave Steinberg, Frank Budinsky, Ed Merks, and Marcelo Paternostro. 2008. *EMF: eclipse modeling framework*. Pearson Education.

[Vaudey(2024)] Jolahn Vaudey. 2024. DSL for modeling IEC 62443 compliant ICS instances, in a zone/conduit model. https://hal.science/hal-04573409