

A Meta Model for a Comprehensive Description of Network Protocols Improving Security Tests

Steffen Pfrang¹ ^a, David Meier¹ ^b, Andreas Fleig¹ and Jürgen Beyerer^{1,2}

¹Fraunhofer IOSB, Karlsruhe, Germany

²Vision and Fusion Laboratory, Institute for Anthropomatics, Department of Informatics, Karlsruhe Institute of Technology, Germany

Keywords: Meta Model, Network Protocols, Packet Structure, Protocol Behavior, Security Testing, Industrial Automation, IACS.

Abstract: Modern industrial automation and control systems (IACS) are highly interconnected via Ethernet. Performing security tests to detect possible vulnerabilities in IACS is one of the measures requested by the IEC 62443 series of standards in order to improve their security. However, security testing tools and frameworks which exceed the power of random fuzzing require precise network protocol definitions. Unfortunately, those definitions vary greatly from tool to tool. Additionally, their creation and maintenance is time-consuming and error-prone. In consequence, especially common IACS protocols like Profinet IO or OPC UA are seldom to never implemented. To overcome this issue, this work proposes and implements an approach of a generic meta model for a comprehensive description of arbitrary network protocols. An important use case of this meta model is the export of network protocol definitions for different testing tools.

1 INTRODUCTION

Security tests on IACS gain an increasing significance the more industrial devices get interconnected via Ethernet. Emerging concepts like Industrial Internet of Things (IIoT) and Industrie 4.0 are strong drivers for new connections of IACS even to the Internet. Security tests aim at detecting vulnerabilities in the network protocol implementations or in the design of the devices which might stop production processes or even harm humans. Once detected, manufacturers and integrators of IACS are enabled to patch their systems and close the respective security gaps.

Security tests are conducted utilizing specialized tools and frameworks. Their purposes range from protocol analysis to fuzzing to stress tests with a very high transmission rate. Except simple fuzzing tests with random data as payload, the tools depend on precise network protocol definitions. Unfortunately, these definitions vary greatly from tool to tool. This leads to the situation that especially common IACS protocols like Profinet IO (PNIO) or OPC UA are seldom to never implemented in these tools.

To solve this problem, this work proposes a generic and universal meta model for a comprehensive description of arbitrary network protocols. The general idea of this meta model is that network protocols can be specified in a standardized way as an instance of the meta model. Then, exporters allow for creating network protocol definitions specific to the respective testing tools. Moreover, the meta model enables several other valuable applications in the domain of security analysis.

The main contributions of this paper are

- a use case analysis of a meta model for network protocols,
- a requirement analysis for such a meta model,
- the design of the meta model, and
- the evaluation of the meta model by conducting a security test on real IACS devices.

The rest of the document is organized as follows: In section 2, background information on network protocol design, industrial automation protocols and security testing is given. A broad use case analysis is made in section 3, the section closes with the formulation of requirements which the developed meta model has to fulfill. The meta model design is presented in section 4. Its implementation is evaluated in section 5 by

^a  <https://orcid.org/0000-0001-7768-7259>

^b  <https://orcid.org/0000-0003-0660-8087>

combining several use cases with a real industrial protocol and common industrial devices in a security test. The paper finally concludes with section 6 including possible future work.

2 NETWORK-BASED COMMUNICATION AND SECURITY TESTING

This sections motivates the need for a meta model describing network protocols comprehensively. It starts with industrial automation protocols and moves on to the design of common network protocols. Then, security testing as a means to analyze security properties is introduced briefly, next a reference protocol and finally the deduced implications.

2.1 Industrial Automation Protocols

Modern industrial facilities are based on highly automated processes that are controlled by IACS. Widespread network protocols in this area are PNIO and OPC UA.

The PNIO communication suite is used to enable real-time Ethernet-based communication in industrial networks, for example between programmable logic controllers (PLC) and sensors or actuators. It consists of several sub-protocols. The PNIO protocol design is described in *Technical Specifications*. They are published as IEC standard and formatted as PDF documents with several hundreds of pages (IEC, 2019a) and (IEC, 2019b). Meanings of options and state transitions are depicted in tables and diagrams.

The OPC UA protocol stack is designed for machine-to-machine communication. It features different communication architectures like server/client and publish/subscribe. OPC UA is also published as IEC standard in 15 parts as PDF documents. The mapping how data and information are transferred between OPC UA Servers and Clients is specified in part 6 (OPC Foundation, 2017). In addition to tables and diagrams, the information model and type definitions are supplied as XML schema definitions.

2.2 Protocol Design

Network communication protocols are defined by two main properties: *Packet structure descriptions* and the *protocol behavior*.

The packet description defines the structure of the individual protocol data units (PDUs) – also referred to as *messages* – that are transmitted over the network.

PDUs can be further dissected into data fields. Every field consists of data coded in a certain way and can influence the meaning of other fields, the PDU and the overall protocol state. Packet structure descriptions vary from simple structures and protocols with only a single packet type to more complex protocols with different kinds of PDUs and packet fields that influence the structure of every individual PDU.

The protocol behavior describes the communication flow of a protocol. This includes the connection initiation, data flow, control messages, as well as the connection closing. Besides stateful protocols, there exist also stateless protocols which do not hold any information about previous messages.

In most cases, PDUs transmitted over a network not only consist of a single protocol but are consisting of multiple parts, each representing a protocol. This layering approach is formalized, for example, in the ISO/OSI-model (ISO, 1994) or the Internet protocol (IP) stack (Braden, 1989). The PNIO stack aligned to the IP stack is shown in Figure 1. Protocols in shaded boxes are defined in the PNIO standard, non-shaded boxes represent protocols that are used by PNIO. While the configuration and management protocol (CM) is based on IP and UDP, the real-time protocol (RT) is based directly on Ethernet. The discovery and configuration protocol (DCP) is also based on Ethernet and is used to discover and configure PNIO-compliant devices in a sub-network. DCP enables, for example, to find all available devices using a *DCP Identify Request*.

Each layer can have various ties to upper or lower layers. For example, the type field in the Ethernet packet determines the protocol used as its payload. While IP uses the type *0x0800*, DCP and CM use *0x8892* and LLDP uses *0x88CC*.

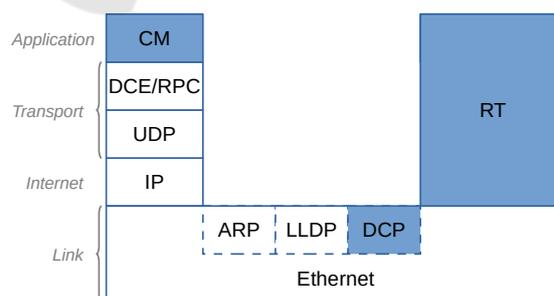


Figure 1: The PNIO stack aligned to the IP stack.

When interpreting the binary data transmitted by a PDU, it can be dissected into its different protocol parts. Each part consists of a header, containing protocol and payload information, and the payload itself. Some protocols, like Ethernet, also have a footer following the payload that can contain, for example, er-

ror detection data. The payload can be the PDU of another protocol, consisting again of header and payload, and so on.

Internet protocols like IP or UDP are described in so-called *Requests for Comments (RFC)* or *Internet Standards* and published by the Internet Engineering Task Force (IETF). The format of those documents is specified in (Flanagan and Ginoza, 2014). It consists of plain text or XML pages; packet structures and state diagrams are formatted as *ASCII art*.

2.3 Security Testing

Security testing aims at detecting vulnerabilities which can be exploited by attackers. In case of IACS connected to networks, flaws in network protocols are a main gateway for attacks. Part 4-1 (IEC, 2018) of the IEC 62443 standard consequentially requires industrial product suppliers to conduct different kinds of security tests during the life-cycle of their devices. Generally, security considerations should be part of every network protocol design process. Nevertheless, the results need to be verified. Also the security properties of existing protocols actively used need to be assessed and validated.

According to (Felderer et al., 2016), security testing can be classified in four categories: Model-based security testing (MBST), code-based testing and static analysis (CBTSA), penetration testing and dynamic analysis (PTDA) and security regression testing (SRT). The application to network protocol analysis will be presented in the following. While MBST techniques analyzes the specification, CBTSA and PTDA analyze implementations of a network protocol. SRT is related to both areas.

MBST. Deals with the analysis of the protocol specification. It presumes the existence of architectural and functional models of the network protocol. As already depicted, most protocols are specified in plain text, diagrams and tables. If a protocol is modeled manually, there is no guarantee of completeness and correctness. So in the strict sense, results of this method are only valid for the particular model, not the original specification.

CBTSA. Focuses on the source code of a protocol implementation. Manual or automated code reviews are common means, the latter is called static code analysis. These approaches are defined as white-box security testing techniques.

PTDA. Is a black-box approach. Testing is performed in practice by using a live system running

the protocol implementation. There is no need to have insights into the implementation, only messages transferred via Ethernet are regarded. PTDA includes manual penetration testing, automated scanning for known vulnerabilities, dynamic analysis of data sanitization, and fuzzing.

A widespread tool for a manual network packet analysis is *Wireshark*¹. Wireshark dissects network packets in its layers and fields. Additionally, for some protocols like TCP it provides a basic state analysis which aggregates several packets to a session. Wireshark uses packet definitions that are programmed in C.

Network packet crafting can be performed using *Scapy*², an interactive packet manipulation program. It supports both packet definitions and a state model. Scapy, as well as its packet definitions, is programmed in Python. Besides a manual usage, Scapy can be used as a library in Python scripts. Industrial security testing or exploit frameworks like *ISF*³ or *ISuTest* (Pfrang et al., 2017) employ Scapy packet definitions.

Fuzzing can be used to automatically test network protocol implementations. The effectiveness of fuzzing can be improved if the network protocol definition is available to the fuzzing tool. Then, fuzzing tools like *ISuTest* purposefully change values in the PDU fields, including creating invalid combinations of values (Pfrang et al., 2018).

High loads of network traffic can lead to denial of service. Packet generators can be used to test which loads are acceptable for a device and how it reacts to overload. There are several different types of packet generators available. Their maximal load ranges from slower ones with Scapy definitions to faster ones with native network interface support, or even hardware-based generators. *Ostinato*⁴ is a packet generator and network traffic generator which can be extended by packet definitions. Unimplemented protocols can be added to Ostinato as userscripts. They are implemented in QtScript.

SRT. Is intended to ensure that changes that are made to a system do not decrease its security. In regard to network protocols, this could happen if a specification is, for example, extended by new options or features. Then both the specification and implementation should be subject to a new security test. In order to minimize the effort needed for these tests, automated testing would be most beneficial.

¹<https://www.wireshark.org/>

²<https://scapy.net/>

³<https://github.com/dark-lbp/isf>

⁴<https://ostinato.org/>

2.4 Example Network Protocol

The *Example Network Protocol* (ENP) is an artificial network-based communication protocol. It was introduced in (Pfrang et al., 2019b), motivated by the need of a reference protocol targeting tool developers to design, implement and test their tools against. The authors examined common Internet protocols as well as specific industrial automation protocols like PNIO and OPC UA. They developed the ENP to contain every observed peculiarity with the idea in mind that a tool that is able to handle the ENP is able to handle every imaginable network protocol (Pfrang et al., 2019a).

During their analysis, the authors developed a packet field categorization which is based on the fields' lengths and parsing rules. Both can be either fixed or variable; the variability can depend or be defined by several different causes. Additionally, several protocol state related requirements were defined. Examples are triggers for and effects of state changes.

2.5 Conclusion

Network protocol specifications are rarely described in a machine-readable way. Security testing approaches, like model-based testing, essentially need complete network protocol models. Fuzzing gets more powerful if packet definitions are used instead of random data as payload. But also tools for manual security research, like Wireshark or Scapy, depend on the existence of protocol definitions. Unfortunately, most of the tools use different formats for their definitions – they cannot be easily interchanged. Especially in the industrial domain, protocols are seldom to never implemented in every tool. As security testing is essential for discovering vulnerabilities before attackers can exploit them, this work proposes a meta model for a comprehensive description of network protocols.

3 USE CASE ANALYSIS

A meta model for network protocols offers various kinds of use cases. The ones that are considered within this paper are depicted in Figure 2. A special focus is laid on the clarification how these use cases can help to improve security tests.

The big box represents the meta model. Inside the meta model, there are different model instances which represent different network protocols. For each of the protocols, the model instances contain both packet descriptions and protocol behavior. At the bottom left

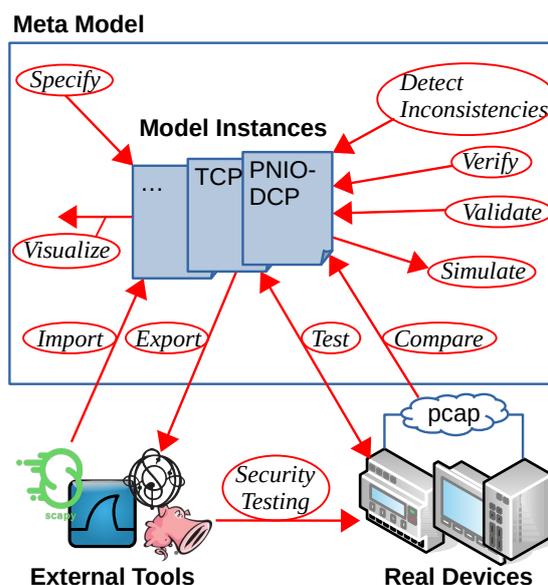


Figure 2: Use cases of the meta model.

side, external tools like Wireshark or Scapy are depicted. Each of those tools use different formats for their own protocol descriptions which are not compatible. At the bottom right side, real devices communicating over a network with the specific protocols are displayed. Recorded network traffic can be stored as a packet capture (pcap). *Italic terms* in red ovals with arrows represent use cases that are being described in the following two subsections. The section closes with the formulation of requirements which have to be fulfilled by the meta model design.

3.1 Basic Use Cases

Basic use cases deal with network protocols as an instance of the meta model. They are about the specification and the refinement of the protocol instance.

Specification of Network Protocols (Specify). In protocol engineering, network protocols can be specified as an instance of the meta model. Their definitions include both packet descriptions and protocol behavior. The knowledge of data types and dependencies between different packet fields lays the foundation for targeted security tests. The specifications are readable for machines as well as for humans. This allows for the use of different kinds of tools which supports the development process. In addition to the creation of new protocols, existing protocols can be specified formally.

Import of Existing Definitions (Import). Existing network protocol definitions from other tools can be

imported as an instance of the meta model. A complete and fully automated import of both packet definitions and state models is desirable but not realistic in every case. But any help of an at least partially automated import could improve the situation of security researchers.

Visualization of State Machines (Visualize). Definitions of stateful network protocols comprise a state model as a vital part. This state model does not only describe the states and transitions, but also explains the protocols processes. A machine-readable protocol instance allows for the automatic generation of those charts which improves the clarity of even complex state models. This can simplify the process for security researchers to understand the protocol in depth.

Detection of Protocol Inconsistencies (Detect Inconsistencies). Both existing and network protocols in development can be underspecified. There could exist mistakes like in the message definitions, or non-reachable states in the state machine. The machine-readability of a meta model instance of the given network protocol allows for automatic inconsistency checks. Once detected, those protocol implementations can be corrected which might close security issues. Besides that, security tests can be focused on discovered inconsistencies which might detect vulnerabilities.

Verification of Protocol Properties (Verify). Various kinds of complex protocol analyses and verification methods can be employed to an instance of a meta model. The machine-readability allows for a fully-automated processing. This could help to prove certain properties of network protocols like security features.

Export in Various Formats (Export). Instances of the meta model can be exported in network protocol definitions for different tools. Like in the case of the import of protocol definitions, a fully automated export is desirable. In practice, even an only partially automated export could facilitate the use of tools with the specific network protocol. The export feature is one of the most valuable parts for security testers since it enables the usage of arbitrary testing tools.

3.2 Advanced Use Cases

Advanced use cases make use of one or more basic use cases and add further functionality.

Conversion of Protocol Definitions. A combination of the import and export use case is the conversion of protocol definitions from one tool to another. It is not depicted explicitly in Figure 2. A common case is that a protocol definition exists only in Wireshark, but a security tester needs to craft network packets for example with Scapy.

Validation of External Protocol Definitions (Validate). Given a reference specification of a network protocol as instance of the meta model. Then protocol implementations from external tools can be imported and compared to the reference. This allows for the validation of external implementations.

Comparison with Recorded Traffic (Compare). A network protocol is defined as a protocol instance of the meta model. The network traffic of real implementations using this specific protocol can be recorded as packet capture. Then it is possible to check both the packet definitions and the protocol behavior. This feature can be used in two directions: Either one assumes the real implementation to be correct, then the meta model instance can be adapted. Or the protocol implementation of a device has to be changed. Additionally, security testers can verify a specific testing process and identify if a discovered anomaly is caused by the testing tool itself.

Support of Protocol Prototyping (Test). A protocol instance which fully describes a certain protocol allows for using a simulator. The simulator can send and receive network packets and additionally execute the state model. This enables fast prototyping of protocols.

Security Testing. Security research can profit from different features of the meta model. This ranges from the verification of protocols inside the meta model to tests of real devices applying external tools. An example is the setup in the evaluation: Several IACS will be tested with the external network packet generator Ostinato. Ostinato does not support the industrial network protocol by default, but the meta model enables it.

3.3 Meta Model Requirements

Based on the needs for the realization of the use cases presented above, the following requirements have been established and grouped into three requirement types: General meta model requirements, message specification requirements and state model requirements.

General Meta Model Requirements (GR). Describe universal requirements for the meta model design.

GR-1 Universal Applicability: Similar to an RFC, the meta model has to be capable of specifying arbitrary network protocols. It has to be possible to create a detailed model instance for any conceivable network protocol.

GR-2 Modularity: The meta model should follow a modular design to decrease its complexity and increase its adaptivity. This applies for both tools like an importer or an exporter, and parts of message or behavior definitions that different protocols have in common.

GR-3 Extensibility: Both the meta model itself and the protocol definitions, which are described as instances of the meta model, need to be easily extensible.

GR-4 Machine-Readability: Instances of the meta model need to be in a machine-readable format to enable automated and tool-supported usage.

GR-5 Human-Readability: Additionally, instances of the meta model have to be human-readable. This allows for an easy handling of meta model instances even without using complex tools.

Message Specification Requirements (MR). Represent the specific needs for implementing arbitrary network protocol messages as instances of the meta model.

MR-1 Message Type Specification: A network protocol may consist of different messages. Every message needs a unique message name, a message type indicator and a list of contained fields.

MR-2 Field Specification: A message can contain multiple fields. Every field needs a unique field name and can be either atomic or compound. The length specification is either fixed or variable. In the latter case, the length value always has to be specified within an external field, within the field itself or within a separate request. The parsing rule specification is either fixed or variable. A variable parsing rule has to be specified or named either within an external field, the field itself or by a separate request. Both the variable length specifications and the variable parsing rule specifications may be based on quite complex rules. The meta model has to be able to specify arbitrary and Turing-complete length and parsing rules.

MR-3 Underlying Protocols: To be able to realize today's protocol stacks, it has to be possible to reference the underlying protocol. On the one side, the meta model must be able to specify a certain underlying protocol. On the other side, it also has to be

possible to reference certain details within this underlying protocol.

State Model Requirements (SR). Describe requirements which are derived from the necessity of being able to implement the protocol behavior of arbitrary network protocols.

SR-1 State Specification: For each possible state of the protocol, there has to be a corresponding state in the state model.

SR-2 Transition Specification: Every possible change of the protocol state has to be specified by a corresponding transition in the state model.

SR-3 Transition Trigger: Each of those transitions has to have exactly one of the following triggers: (a) A certain message or the content of a certain field within a message, (b) a temporal condition like a timeout or a timed event, or (c) an action within the protocol execution which triggers a certain transition.

SR-4 Arbitrary Actions: The state model has to allow the execution of arbitrary, Turing-complete actions, including special constructs like conditional actions or loops.

SR-5 Roles: It has to be possible to specify different roles. The most common examples of such roles are client and server. Depending on the current role, the states and transitions may differ.

SR-6 Data Model: The state model has to offer some kind of data model which allows to access data objects or data structures. This is meant to be used to maintain all kinds of local and global information related to messages and states.

SR-7 Message Specification References: It has to be possible to reference parts of the message specifications of the corresponding meta model instance from inside the state model. For example, one of those actions may depend on some specific details of the message specifications.

SR-8 State Model References: It has to be possible to reference other model instances for arbitrary actions. It is particularly important to be able to access the underlying protocol since the protocol may depend on certain details of it.

4 META MODEL DESIGN

The meta model design comprises three parts as depicted in Figure 3: The meta model specification, meta model instances and the meta model framework. The specification deals with the concepts of how a network protocol can be described comprehensively as an instance of the meta model. Meta model instances are actual implementations of network pro-

protocols. The framework design contains the software components which allow users to interact with the meta model instances.

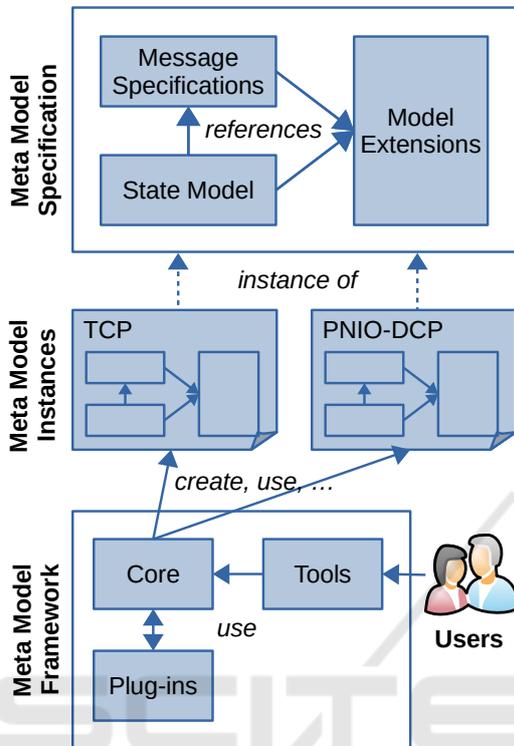


Figure 3: Meta model design overview.

4.1 Meta Model Specification

The meta model specification consists of three conceptual parts: Message specifications, a state model and model extensions. While the first two parts follow an XML-based approach, the latter makes use of a Turing-complete programming language. A network protocol like TCP would need to be specified by creating instances of these three parts. This separation into different parts demands that the consistency within a meta model instance is ensured.

Message Specification. The basic idea for the specification of messages is to define an instance of a certain message as an XML document. That way, every message can be represented by a corresponding XML document. Figure 4 depicts an *Identify all request* of DCP in its XML representation.

To specify the format and characteristics of those XML documents, an XML schema is created. The most powerful XML schema language is *XML Schema Definition (XSD)* (W3C, 2012). It combines both an assertion-based and a grammar-based

```

1 <DCP>
2   <DCPHeader>
3     <ServiceId>5</ServiceId>
4     <ServiceType>
5       <ServiceTypeFlag>0</ServiceTypeFlag>
6       <ServiceTypeType>00</ServiceTypeType>
7     </ServiceType>
8     <Xid>3256714648</Xid>
9     <ResponseDelayFactor>0
10      </ResponseDelayFactor>
11   </DCPHeader>
12   <IdentifyAllReqPDU>
13     <AllSelectorBlock>
14       <Option>ff</Option>
15       <SubOption>ff</SubOption>
16       <DCPBlockLength>0000</DCPBlockLength>
17     </AllSelectorBlock>
18   </IdentifyAllReqPDU>
19 </DCP>

```

Figure 4: The DCP Identify all request denoted as XML document.

approach. There exist different external tools which allow for checking XML documents against an XSD schema.

Figure 5 shows excerpts of the corresponding XSD schema. Within the schema definition, messages and their fields employ grammar-based XSD rules. For example, line 2 defines that there has to be exactly one *DCPHeader*. The header has a field *ServiceId* (line 4) which is restricted (line 6) to a Byte. If the *ServiceId* is 5, the message has to be interpreted as *IDENTIFY request* (line 10).

More complex rules like dependencies between fields, however, make use of assertions. An example for an assertion determining the type of the PDU is printed in line 12ff. Applying XPath selectors, values of message fields are evaluated against a fixed value. Such rules have to be written for each PDU type.

If the needs exceed the capability of XSD assertions, like in the case of a hash calculation, external assertions will be used. They are marked as annotations and employ model extensions.

State Model. The state model is realized as *statechart XML (SCXML)* (W3C, 2015). SCXML is a simple, standardized way to specify all kinds of state models in XML documents. The meta model design employs *Harel statecharts* (Harel, 1987) which are slightly more powerful than classic finite state machines. For example, they easily allow the specification of sub-states or concurrent tasks. This even allows the specification of complex protocols.

Figure 6 illustrates excerpts of the statechart definitions of the ENP. The name of the protocol in-

```

1 <xs:schema [...]
2 <xs:element name="DCPHeader" minOccurs="1"
   maxOccurs="1">
3 [...]
4 <xs:element name="ServiceId"
   minOccurs="1" maxOccurs="1">
5 <xs:simpleType>
6 <xs:restriction
   base="xs:unsignedByte">
7   [...]
8 <xs:enumeration value="5">
9 <xs:annotation>
10 <xs:documentation>IDENTIFY
   </xs:documentation>
11 [...]
12 <xs:assert test="if (
   DCPHeader/ServiceId = 5 and
   DCPHeader/ServiceType/ServiceTypeFlag =
   0 and
   DCPHeader/ServiceType/ServiceTypeType =
   '00')
13 then (IdentifyAllReqPDU)
14 else true()">

```

Figure 5: Extracts of the corresponding DCP XSD.

```

1 <scxml [...] name="enp"
   initial="connecting">
2 <state id="connecting">
3 <transition type="external" event=
   "conn_ok" target="connected"/>
4 <transition type="external" event=
   "error_occ" target="error_occ"/>
5 <transition type="internal" event=
   "state_act">
6 <assign location="res" expr="fkt_connect
   (('localhost', 1337)"/>
7 <if cond="res == -1">
8 <raise event="error_occ"/>
9 </if>
10 </transition>
11 </state>

```

Figure 6: Extract of the state model of the ENP.

stance is *enp*, its initial state is *connecting* (line 1). In this state, three transitions are defined. In case of a *conn_ok* event, the transition to the state *connected* is triggered. If an error occurs, the target state *error_occ* will be aimed.

Harel statecharts execute actions – like Moore machines – within states. For that reason, actions have to be described in an *internal* event. This internal event (line 5) will be triggered once the state is reached and executes a function call. In case of an error (line 7), the external event defined in line 4 will be triggered.

Model Extensions. There are two different types of model extensions. The first one serves the re-usability of common types within the message specifications and the state model.

The second one is used whenever needed functionality exceed the features of XSD and SCXML. These features are implemented in *JavaScript*, a Turing-complete programming language, which is the standard in SCXML. However, the complexity of these extensions prevents automatic processing by conversion tools and, thus, should only be used when unavoidable.

4.2 Meta Model Framework

The meta model framework is structured in three conceptual parts: The core, tools and plug-ins.

The core is the central component of the framework. On the one hand, it represents the interface between the tools and meta model instances. It offers high-level functionality, like checking the validity of a model instance via an application programming interface (API). An example for a tool is the *pcap checker*, which interprets network messages from a *pcap* and compares both packets and protocol behavior with a given model instance. Tools are the only interface between a user and the meta model.

On the other hand, the core interfaces with plug-ins. These plug-ins enable the framework to import and export protocol definitions to and from a meta model instance. A plug-in contains code to both interpret and write one specific protocol definition language. It is independent from a specific meta model instance.

5 EVALUATION

The evaluation of the meta model consists of three parts. The first part combines four use cases from the specification of a well-known network protocol to the analysis of a security test targeting real IACS. Those use cases were selected because the implementation of meta model tools was focused on the ones from which security testing benefits most. The second part examines if the meta model provides the needed functionality to specify arbitrary network protocols. The third part compares the meta model with related approaches. Finally, the section closes with a discussion of the results.

5.1 Use Case Evaluation

The use case evaluation consists of four steps which are depicted in Figure 7. It applies the use cases *Specify*, *Export*, *Security Testing* and *Compare*. Security testing is conducted with the packet generator Ostinato. Targets for the security test were two embedded PLCs and two PNIO gateways for analog and digital signals.

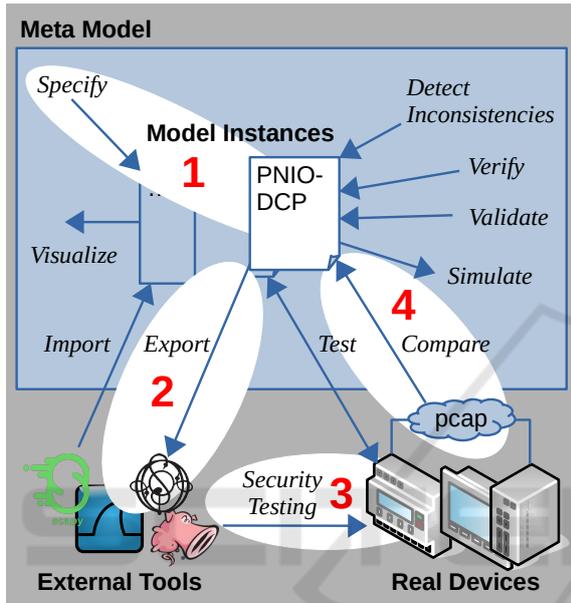


Figure 7: The use-case-based evaluation plan.

In step 1, the DCP protocol will be analyzed manually from the PNIO standard document and implemented as an instance of the meta model. DCP is a stateless protocol which means only a packet description is required. Excerpts of the resulting XML schema are depicted in figure 5 in the design section.

The second step will export the network packet definitions for Ostinato. This workflow is depicted in Figure 8. It starts with a user interaction (A) and runs the *Exporter* tool with the name of the meta model instance and the format that shall be created. The exporter then accesses the *DCP* model instance via the API of the *Core* (B). Making use of the export rules of the *Ostinato* plug-in (C), the corresponding *Ostinato* userscript is written to a file. Note that the generic exporter tool is able to export any model instance into any target file format for which export rules exist.

In step 3, a security test with *Ostinato* will be conducted on a test bed of IACS. The packet generator *Ostinato* is configured with the packet definitions of DCP as a userscript. Then, the devices to be tested are configured as targets of the test. Further settings concern the number of packets to be generated and the

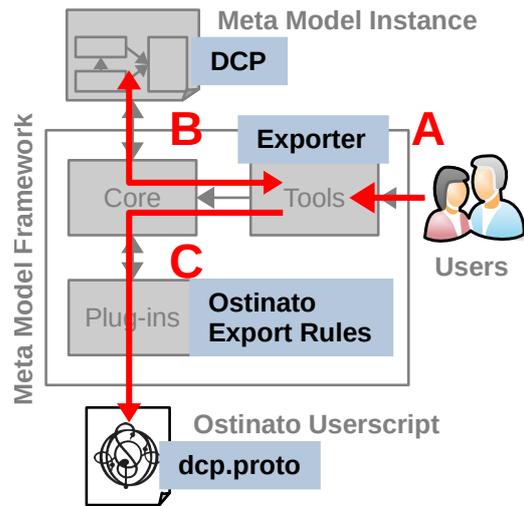


Figure 8: The workflow for exporting protocol definitions.

number of packets to be sent per second. In this case, none of the attacked IACS crashed – those devices are not vulnerable to a high load of DCP messages.

Step 4 concludes the evaluation with a proof of the correctness of the DCP model instance, the exporter tool and the *Ostinato* export plug-in. Taking a pcap of the security test as input, both a manual analysis with *Wireshark* and an automated analysis with the pcap-checker tool were conducted. The analysis with *Wireshark* showed that the network packets generated by *Ostinato* could be dissected as valid DCP messages without any error. Additionally, the devices answered the DCP identify request with a DCP identify response. This shows that the messages were processed correctly by the PNIO devices.

Finally, the pcap-checker tool was employed. It showed that the generated DCP messages were correct DCP messages according to the meta model instance of DCP.

5.2 Requirement Evaluation

The goal of the meta model is that it allows for a comprehensive modeling of every network protocol, regardless of its complexity. The evaluation of this goal can be reduced to the implementation of the ENP as a meta model instance.

Based on the requirement analysis, the satisfaction of both the message specification (MR) and state model requirements (SR) is also shown through the ENP instance.

Regarding the satisfaction of the general meta model requirements, universal applicability (GR-1) has been shown by both the implementation of the artificial ENP and the real DCP protocol, with the latter having been used for a security test. The modular-

ity (GR-2) is ensured by the design of plug-ins and tools as well as in common types for protocol definitions. The extensibility (GR-3) in respect to new protocol definitions is guaranteed by the design of meta model instances. The extensibility in relation to the meta model is ensured by the separation of the meta model specification and framework. Finally, the readability of meta model instances by both machines (GR-4) and humans (GR-5) is offered by the usage of XML as the description language.

5.3 Comparative Evaluation

There exist different other approaches which target only parts of the meta model's features. Some of them are only usable for packet definitions, others target solely the specification of a protocol's state model. In the following, six approaches are presented and compared to the meta model. The reasons for not being used within the meta model design get motivated.

YANG. YANG (M. Bjorklund, 2010) (Bjorklund, 2016) is a data modeling language. It was developed to model configuration and state data for the network management protocol NETCONF (R. Enns, 2006). While YANG itself only describes data models, JSON or XML can be used to describe data instances. To describe data models, YANG supports XPath 1.0 expressions.

The specification of simple fields with fixed length and fixed parsing rules – like a simple integer field – is easily possible using YANG's primitive data types. But the specification of fields with variable lengths or parsing rules is much more complex. Especially because it is not possible to define real custom types in YANG, but only to derive custom types from the pre-defined ones. So even the specification of simple type definitions are much more complex compared to the presented XML-based approach of the meta model. This especially contradicts the requirement of human-readability (GR-5), and also increases complexity for the automated processing of the meta model (GR-4).

PacketTypes. The authors of PacketTypes (McCann and Chandra, 2000) introduced a type system similar to type systems in programming languages like C. It is particularly designed to specify messages of network protocols.

In PacketTypes, there are no primitive data types, but only one generic data type called *bit*. Thus, content restrictions of any kind have to be specified using the *where*-clause of PacketTypes. That way, an 8 bit integer field can be specified using the two clauses

$field\#value \geq 0$ and $field\#value < 256$. Unfortunately, the authors only discuss some basic examples like this. Therefore, it would be necessary to heavily extend this approach to be able to specify arbitrary restrictions or dependencies between fields as required.

Protocol Buffers. Google released Protocol Buffers (protobuf) in 2008 aiming at a simple mechanism for serializing structured data. Its protocol compiler is written in C++, but is able to output source code in currently ten different programming languages (Google, 2008). Protobuf defines its own syntax to define messages and types in *.proto*-files. They are mainly based on messages and fields and can include options for the compiler.

Protobuf does not provide any possibility to model the state of a network protocol, missing all State Model Requirements (SR) of the Meta Model. The protobuf compiler also produces its own binary format of the serialized data, meaning it cannot be used to output arbitrary binary network data. This would be needed to create or dissect real communication data.

Formal Description Techniques (FDT). Standardized by organizations like ISO and CCITT, FDTs are developed since the 1970s. They aim at specifying formally a whole system's features, functionality and behavior. Major examples for FDTs are *Estelle* (Extended Finite State Machine Language), *Lotos* (Language Of Temporal Ordering Specification) and *SDL* (Specification and Description Language). They are broadly described and compared in (Turner et al., 1993).

Since FDTs focus on specifying systems' behavior, messages and specialized data types are less developed. But the main reason for not using FDTs within the meta model is the absence of current implementations and tools.

Model Checking. Regarding the state model of network protocols, there are multiple approaches like (Musuvathi et al., 2004), aiming to specify network protocols for model checking purposes. The basic idea is to perform automated tests on those models, utilizing formal verification methods. This requires to create a formal model of the network protocol.

While the very structural and mathematical representation of models like this is necessary to perform those formal verifications, they also make the models quite complex and especially less human-readable. Additionally, while such a representation is perfectly suitable to perform automated checking, it is much more complex to interpret those models. Therefore,

similar to YANG, both the human-readability (GR-5) and the automated processing (GR-4) of the meta model is much more complex with such models compared to the XML-based approach of the meta model.

Object-oriented Petri Nets. For the specification of the protocols state model, Petri nets are a conceivable alternative for the Harel statecharts used in the meta model. In general, Petri nets are especially suitable for the specification of more complex procedures such as parallel actions. Compared to classic state machines, they are more powerful. On the contrary, Petri net based representations are generally more complex. Additionally, it is possible to extend Petri nets by object-oriented aspects. This allows the easy specification of network protocol messages within the specification of the state model. So this approach allows a combined model for both parts of the protocol specification.

For the description of such object-oriented Petri nets (OPNs), there exist multiple different approaches. Two examples for available languages are *LOOPN* (Lakos and Keen, 1991) and the more advanced version *LOOPN++* (Lakos and Keen, 1994). Unfortunately, many of the object-oriented aspects of OPNs like polymorphism or dynamic binding do not offer a great advantage for the specification of network protocols, but introduce additional complexity to the model. Thus, in many cases, a representation using finite state machines is much simpler, and therefore both easier to read for humans (GR-5) and easier to process automatically (GR-4).

5.4 Discussion

In combination, all three parts of the evaluation have shown that the meta model is powerful enough to specify arbitrary network protocols. Particularly, it has been shown that the meta model enabled an existing security testing tool to test a protocol which was not implemented before. That is a very valuable feature in the industrial domain because many domain-specific network protocols are seldom to never implemented in widespread tools. Additionally, the design of the meta model framework allows for exporting every model instance for a given testing tool, requiring only a single plug-in.

As the meta model is a new approach, the implementation currently supports only one industrial communication protocol and one security testing tool. But through the design of the meta model, new protocols and tool support can be easily added.

The evaluation showed that export features need little to no manual work in order to create runnable

protocol specifications. Import features tend to be more complicated because external code has to be interpreted. Nevertheless, even a small support in defining PDU field types is better than having to start from scratch.

Finally, it turned out that machine-readability is necessary, but not sufficient for a protocol instance of the meta model. Different assertions, rules and, optionally, model extensions have to be combined manually. This can be a time-consuming and error-prone task. Particularly, automatic completeness and correctness checks are complicated since there is not yet a defined order and usage of the constraints. For example, the occurrence 1 of the field X can be ensured with the rule *minOccurs=1* and *maxOccurs=1* or an assertion which checks if the number of fields X equals 1 . The same applies to the model extensions in JavaScript. Since the chosen programming language has to be Turing-complete by design, it is challenging to determine which functionality a method implements.

6 CONCLUSION AND FUTURE WORK

This work proposed a generic and universal meta model for a comprehensive description of arbitrary network protocols. A special focus was laid on security testing for the industrial domain since IACS testing suffers most from missing network protocol definitions.

A broad use case analysis led to the formulation of requirements for such a network protocol meta model. Since no known approach proved to be sufficient, a new solution was designed. It is based on well-known XML specifications which allow for the reuse of existing tools and tool chains.

During the evaluation, a special reference protocol (ENP) was implemented using the meta model, as well as a typical industrial network protocol (DCP). As a result, a testing tool which did not support this specific protocol before was enabled to test embedded IACS. This clearly showed the possible benefits of the meta model.

There are many more use cases in the domain of security testing which can profit from additional protocol specifications. Examples are the verification of protocol properties or the support of protocol prototyping. Beyond that, other domains, like intrusion detection, could benefit from protocol definitions, especially in the industrial domain.

Future work includes the implementation of additional export rules in order to supply more testing

tools. Also more protocols will have to be implemented as instances of the meta model. The pool of available network protocols could be increased with import functionality from other tools. Furthermore, new tools like a visual editor for network protocols would facilitate working with the meta model.

The most important task is indeed the design of an abstraction layer for the constraints regarding field types and dependencies between fields. These constraints are defined by multiple assertions, rules, and optionally by model extensions. This abstraction layer could be applied as specific comments or annotations within the different XML documents of a meta model instance. It combines a set of assertions and rules, and supplies the upper layer with a semantic description of its functionality. For example, it ensures that *field X* is an integer and has to be interpreted as length of *field Y*. Meta model tools could implement this abstraction layer and help to improve the completeness and correctness of meta model instances.

ACKNOWLEDGEMENTS

This work was supported by the German Federal Ministry of Education and Research within the framework of the project *KASTELSKI* in the Competence Center for Applied Security Technology (KASTEL).

REFERENCES

- Bjorklund, M. (2016). The YANG 1.1 Data Modeling Language. RFC 7950.
- Braden, R. (1989). Requirements for Internet Hosts - Communication Layers. RFC 1122.
- Felderer, M., Büchler, M., Johns, M., Brucker, A. D., Breu, R., and Pretschner, A. (2016). Security testing: A survey. In *Advances in Computers*, volume 101, pages 1–51. Elsevier.
- Flanagan, H. and Ginoza, S. (2014). RFC Style Guide. RFC 7322.
- Google (2008). Protocol buffers. <https://developers.google.com/protocol-buffers/>. Online; accessed 2020-01-06.
- Harel, D. (1987). Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, 8(3):231 – 274.
- IEC (2018). *Security for industrial automation and control systems - Part 4-1: Secure product development lifecycle requirements*. International Electrotechnical Commission (IEC), Geneva, Switzerland.
- IEC (2019a). Industrial communication networks - Fieldbus specifications - Part 5-10: Application layer service definition - Type 10 elements. IEC 61158-5-10:2019.
- IEC (2019b). Industrial communication networks - Fieldbus specifications - Part 6-10: Application layer protocol specification - Type 10 elements. IEC 61158-6-10:2019.
- ISO (1994). *ISO/IEC 7498-1:1994 - Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model*. International Organization for Standardization (ISO), Geneva, Switzerland.
- Lakos, C. A. and Keen, C. D. (1991). Modelling layered protocols in loopn. In *Proceedings of the Fourth International Workshop on Petri Nets and Performance Models PNPM91*, pages 106–115.
- Lakos, C. A. and Keen, C. D. (1994). *LOOPN++: A new language for object-oriented Petri nets*. Department of Computer Science, University of Tasmania.
- M. Bjorklund, E. (2010). YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF). RFC 6020.
- McCann, P. J. and Chandra, S. (2000). Packet types: abstract specification of network protocol messages. *ACM SIGCOMM Computer Communication Review*, 30(4):321–333.
- Musuvathi, M., Engler, D. R., et al. (2004). Model checking large network protocol implementations. In *NSDI*, volume 4, pages 12–12.
- OPC Foundation (2017). *OPC Unified Architecture Specification Part 6: Mappings*. OPC Foundation, version 1.04 edition.
- Pfrang, S., Giraud, M., Borcharding, A., and Meier, D. (2019a). Example network protocol. <https://github.com/stepfr/ExampleNetworkProtocol>. [Online; accessed 2019-11-03].
- Pfrang, S., Giraud, M., Borcharding, A., Meier, D., and Beyerer, J. (2019b). Design of an example network protocol for security tests targeting industrial automation systems. In *Proceedings of the 5th International Conference on Information Systems Security and Privacy - Volume 1: ForSE*, pages 727–738. INSTICC, SciTePress.
- Pfrang, S., Meier, D., Friedrich, M., and Beyerer, J. (2018). Advancing protocol fuzzing for industrial automation and control systems. In *Proceedings of the 4th International Conference on Information Systems Security and Privacy - Volume 1: ForSE*, pages 570–580. INSTICC, SciTePress.
- Pfrang, S., Meier, D., and Kautz, V. (2017). Towards a modular security testing framework for industrial automation and control systems: Isutest. In *Proceedings of the 22nd IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2017*.
- R. Enns, E. (2006). NETCONF Configuration Protocol. RFC 4741.
- Turner, K. J. et al. (1993). *Using formal description techniques: an introduction to Estelle, LOTOS and SDL*, volume 154. Wiley New York.
- W3C (2012). Xml schema definition. [Online; accessed 2019-11-17].
- W3C (2015). State chart xml. [Online; accessed 2019-11-17].