Rule-based Code Generation in Industrial Automation: Four Large-scale Case Studies applying the CAYENNE Method

Heiko Koziolek, Andreas Burger, Marie Platenius-Mohr, Julius Rückert, Hadil Abukwaik
heiko.koziolek@de.abb.com
ABB Corporate Research Center Germany
Ladenburg

Raoul Jetley, Abdulla PP
ABB Corporate Research Center India
Bangalore

ABSTRACT

Software development for industrial automation applications is a growing market with high economic impact. Control engineers design and implement software for such systems using standardized programming languages (IEC 61131-3) and still require substantial manual work causing high engineering costs and potential quality issues. Methods for automatically generating control logic using knowledge extraction from formal requirements documents have been developed, but so far only been demonstrated in simplified lab settings. We have executed four case studies on large industrial plants with thousands of sensors and actuators for a rule-based control logic generation approach called CAYENNE to determine its practicability. We found that we can generate more than 70 percent of the required interlocking control logic with code generation rules that are applicable across different plants. This can lead to estimated overall development cost savings of up to 21 percent, which provides a promising outlook for methods in this class.

KEYWORDS

Software design and implementation, code generation, IEC 61131-3, industrial automation, case study, model-driven development

ACM Reference Format:
Heiko Koziolek, Andreas Burger, Marie Platenius-Mohr, Julius Rückert, Hadil Abukwaik and Raoul Jetley, Abdulla PP. 2019. Rule-based Code Generation in Industrial Automation: Four Large-scale Case Studies applying the CAYENNE Method. In *ICSE '20: International Conference on Software Engineering, May 23–29, 2019, Seoul, South Korea.* ACM, New York, NY, USA, 10 pages. https://doi.org/TBD

1 INTRODUCTION

Software to control power production, chemical plants, or oil refineries is often programmed using IEC 61131-3 programming languages and cyclically executed on real-time controllers [18]. This IEC standard defines five programming languages, both graphical (e.g., function block diagrams) or textual (e.g., structured text). Control engineers design and implement this kind of software based on semi-formal requirements specifications from process experts [16].

ICSE '20, May 23–29, 2019, Seoul, South Korea

ACM ISBN TBD…$15.00
https://doi.org/TBD

Nowadays, process control systems, where such software is embedded, comprise million lines of code and feature a distributed infrastructure spanning controllers, local servers, edge gateways, and cloud-backends [23]. There are more than 140.000 process control systems installed world-wide with a total market volume of more than 15 BUSD [13].

Despite initial tool-assisted automation, designing and implementing control logic for process control systems still remains a largely manual process involving significant human labor [16]. The costs for this work contribute to the high engineering costs for such systems, which are typically in the range of several hundreds of thousands USDs for a single project, but may also go up to several million USDs for very large installations. Some of the involved human work is repetitive, developing similar software repeatedly, by re-encoding known patterns. Additionally, manually translated requirements may lead to communication problems between customers and suppliers.

For these reasons, a higher level of automation and code generation for software applications in industrial automation is highly desired [29]. Over the last three decades, practitioners created software libraries to reuse functionality and tools to automatically import list-based specifications [16]. Researchers proposed more than a dozen methods for automated IEC 61131-3 code generation [20]. One class of methods proposes to first model the customer requirements using UML models with automation-specific profiles and then use code generators to create IEC 61131-3 control logic [17, 27, 28]. However, process engineers are usually not familiar with the UML notation, and the additional notation requires keeping models and code synchronized. Another class of methods suggests a rule-based approach to extract knowledge required for code generation directly out of semi-formalized customer requirements [12, 15, 26]. These methods have not been tested in realistic settings, therefore their scalability and robustness are unknown.

The contribution of this paper is an analysis of the practicability of a rule-based control logic generation approach using four large-scale case studies. In each case we generated IEC 61131-3 control logic code from the requirements specifications of actual industrial plants using our rule-based method called CAYENNE (**C**reating **A**rchitectures for rapidl**Y** **EN**gi**NE**ering control systems). It involves translating customer 'piping and instrumentation diagrams' (P&IDs) into a topological plant model, and then executing code generation rules expressed in a novel domain-specific notation. The CAYENNE rule engine traverses the topological models and identifies specific patterns of pipes and instruments that trigger code generation rules.

We found from the case studies that we can generate a particular type of control logic, namely interlocking logic [22], by more than 90 percent using a rule-based approach. We expect that potentially more than 70 percent of these rules mined in these case studies are potentially reusable across different plants, therefore providing great potential for code generation in future cases. We estimated that this form of code generation could lower the efforts for requirements specification, control logic and process graphics engineering, as well as testing by approximately 21 percent. This can lead to cost savings of hundreds of thousand USD in a given project. Domain experts confirmed our findings and provided positive feedback. More case studies may be needed to address additional corner cases and refine the rule set.

The remainder of this paper is structured as follows. Section 2 first provides the necessary background for code generation in industrial automation. Section 3 conceptually explains the core idea of rule-based approaches, before Section 4 describes our particular implementation as the CAYENNE method. Section 5 elaborates on the case study methodology and means for data collection. Section 6 provide individual findings from each case, then Section 7 presents cross-case findings including statistics from rule mining and estimations for effort reduction. Section 8 briefly surveys related work, before Section 9 concludes the paper.

2 CONTROL LOGIC ENGINEERING

Process control systems (PCS) automate various industrial production facilities, such as chemical plants, power plants, paper mills, or oil refineries. Fig. 1 shows the generic components of a process control system. Numerous sensors for temperature, pressure, level, and flow collect data, e.g., from tanks, heat exchangers, or turbines. The sensors feed real-time controllers that compute control signals for various actuators, e.g., pumps, motors, conveyor belts. Engineering workstations allow programming and configuring field devices and controllers, while supervision workstations support human operators in production monitoring and alarm management. A PCS usually connects to a Manufacturing Execution System (MES) managing resources and scheduling processes, as well as an Enterprise Resource Planning System (ERP) managing accounting, procurement, and human resources.

PCS controllers often rely on real-time operating systems, such as Embedded Linux, FreeRTOS, or VxWorks. They cyclically execute control algorithms written in IEC 61131-3 programming languages. These include graphical languages (e.g., function block diagrams), as well as textual notations (e.g., structured text). Listing 1 shows the IEC 61131-3 Structured Text syntax similar to the Pascal programming language in a minimal code sample, where the filling level of a tank is kept below 50 cm by opening or closing its inlet valve. Such control logic keeps industrial processes within safe conditions. Control engineers compile and execute the code in a runtime environment (e.g., CODESYS [1], TwinCAT [2]). Control logic can include sophisticated calculations to optimize production flow.

Control engineers design and implement control logic based on requirements from engineering contractors. Fig. 2 depicts a typical - albeit simplified - engineering workflow for process control systems. An engineering contractor provides an I/O list, which

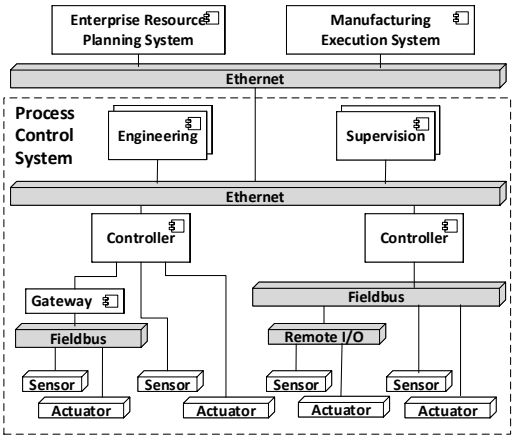

Figure 1: Abstracted Component View of a Process Control System

specifies the required input and output signals (analog or digital) for sensors and actuators. If needed, control engineers implement custom algorithms in new function blocks for special requirements (Step 1) and add them to their function block libraries. The I/O list can then be automatically mapped to function blocks in the library (Step 2), which creates control logic skeletons, e.g., the variable input and output lists of IEC 61131-3 (see Listing 1).

Listing 1: IEC 61131-3 Structured Text Code Sample

```
PROGRAM ValveControl
  VAR_INPUT
    TankLevel : REAL;
  END_VAR
  VAR_OUTPUT
    ValveOpen : BOOL;
  END_VAR

  IF (TankLevel > 50.0) THEN
    ValveOpen := FALSE;
  ELSE
    ValveOpen := TRUE;
  END_IF
END_PROGRAM
```

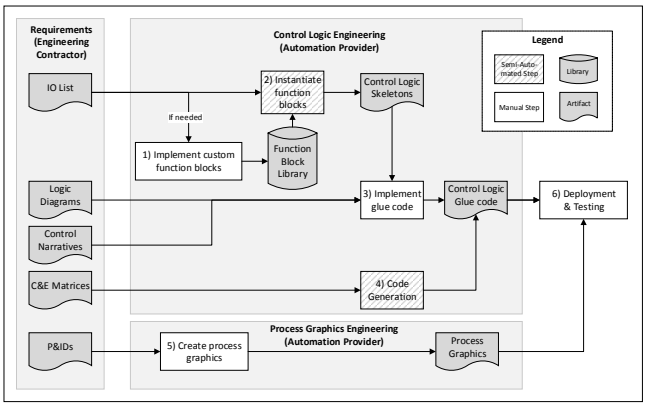

Figure 2: Conventional Workflow of Requirements Specification, Control Logic Engineering, and Process Graphics Engineering

Glue code then needs to be implemented manually, based on informal logic diagrams and textual control narratives (Step 3). Another means for glue code requirements specification are Cause-and-Effect (C&E) matrices, which are tables linking specific signals to specify interlocks. As an example for an interlock, the signal representing an alarm condition in a pipe (e.g., high pressure), could be linked to the signal for venting a vessel, so that the automation system avoids unsafe conditions. Engineering contractors usually specify C&E Matrices and control engineers use code generation tools to translate their logic to IEC 61131-3 (Step 4). Finally, control engineers complete the control logic with manual additions and continue in the deployment and testing phase (Step 6). In parallel, so-called piping and instrumentation diagrams (P&IDs) from the engineering contractor serve as a visual template for a manual specification of process graphics (Step 5), that can be later used as HMI for human operators.

While the conventional process involves some form of code generation already in Step 2) and 4), there is potential for additional automation. Logic diagrams and control narratives could be turned into more formal specifications that allow automatic processing. C&E matrices contain a lot of interlocks following simple patterns (e.g., if a valve is closed, shut down a preceding pump to avoid over-pressure), but are today always specified from scratch. Finally, P&IDs already contain substantial knowledge about the topology of the industrial process (e.g., vessels, piping structures, control requirements), which is today not subject to automatic processing but only manual interpretation.

3 CAYENNE RULE-BASED ENGINEERING

The core idea of our proposed CAYENNE approach is to introduce a rule base with domain-specific, reusable rules to automate simple, re-occurring design and implementation tasks for control logic. A rule engine applies pre-specified rules on the requirement documents and can thus automatically generate parts of the IEC 61131-3 code. A first instantiation of this idea in the typical engineering workflow (Fig. 2) suggests to *generate* C&E matrices using information from P&IDs thus eliminating the need to specify interlocks from scratch for each plant.

P&IDs contain topological information needed for interlock specification and are today created with commercial CAD tools. While engineering contractors currently often provide these drawings only as paper printouts or PDF files, which are hard to process, many CAD tools have recently started to support object-oriented or table-based serialization formats (so-called "smart P&IDs"). The DEXPI initiative [3] (Data Exchange in Process Industries) is working on a standardized XML-format for P&IDs based on ISO 15926 and is supported by all major P&ID CAD tool vendors. Version 1.3 of the DEXPI specification will be released in 2019. Such a representation now enables knowledge extraction from smart P&IDs by applying domain-specific rules.

Fig. 3 shows an abstracted example for the knowledge extraction. The P&ID contains a vessel B101, pump P106, three valves (V102, V104, V105), as well as a level sensor (LS+/B102), and a temperature sensor (TIC/B103). Alarm limits are specified for the level sensor B102 (not shown) indicating the allowed low and high filling levels. A common interlocking rule is depicted on the right hand side of Fig. 3. For CAYENNE, we have developed a streamlined grammar for a rule specification language, which allows an efficient formulation and communication of new rules as well as semantic checks. Rules are specified in a generic way without referring to specific instances. A rule engine can process an object-oriented, smart P&ID model and potentially match each rule many times for different instances.

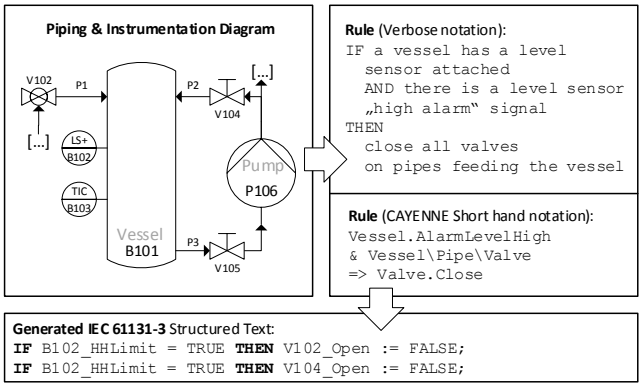

Figure 3: Example P&ID, domain-specific rule in verbose and short-hand notation, and generated IEC61131-3 ST.

In the example, the rule engine identifies all vessels in the P&ID (B101) and checks whether they have a connected level sensor (B102) with an alarm limit specified. It then traverses the model from the vessel via each feeding pipe (P1 and P2) to connected valves (here: V102, V104). The rule engine can then use these instance references to generate a C&E matrix linking the level sensor alarm signal with the valve closing signal. The rule engine optionally generates C&E matrices as an intermediate representation to facilitate manual review and approval by the engineering contractor, since a typical plant may contain thousands of interlocks. The C&E matrix is then processed by a code generator to produce IEC 61131-3 code. The resulting code is depicted at the bottom of Fig. 3.

Although the engineering process appears linearly in Fig. 2 for simplicity, in practice it is an iterative process, where the engineering contractor hands over successively refined requirements multiple times to the automation provider. When re-generating based on updated requirements, the code generator must consider control logic written manually in parallel. To support this, the CAYENNE approach suggests a merging approach where the newly generated code is compared with the existing code and lets the control engineer review and approve any merge conflicts.

While the rule-based engineering approach is illustrated specifically for simple interlocks in the scope of this paper, it is potentially also applicable for other parts of control logic, e.g., batch production recipes, function block configuration parameters, and shut-down procedures.

4 CAYENNE IMPLEMENTATION

To validate the concept of CAYENNE rule-based engineering, we have designed and implemented a prototype tool that interfaces with commercial CAD tools and control engineering tools. Fig. 4 shows the tool's high-level software architecture as an UML component diagram. To decouple the rule engine and code generation from

specific serialization formats for P&ID files, we created a so-called "Topology Model", a meta-model that captures the most important concepts from P&IDs for code generation [9]. The CAYENNE tooling is implemented using C#/.NET and includes importers for SmartPlant P&ID's Excel export format, as well as Microsoft Visio P&ID files. An importer for DEXPI ISO15926 is still work in progress, since the standard and serialization format are not yet finalized.

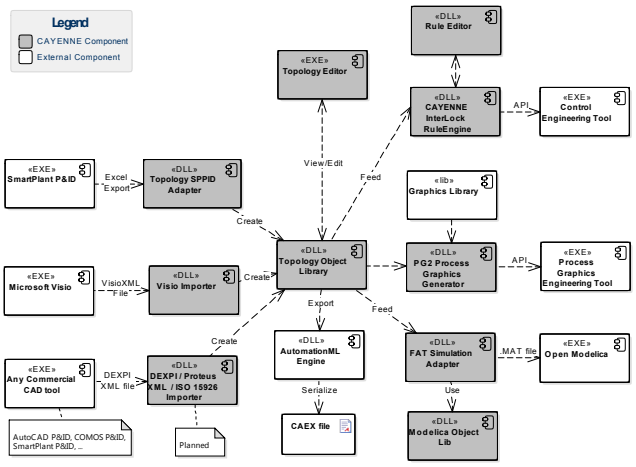

Figure 4: CAYENNE prototype software architecture: static view

Listing 2: CAYENNE Rule Specification Grammar (excerpt)

```
keywordPipe = 'Pipe', ['-', number], [mediatype];

keyword = 'Valve' | 'Conveyor' | 'Vessel' | 'Reactor' | ... ;
 (* plus all elements from Topology Model, see Fig. 6 *)

keynum = keyword, '-', number;

functions = 'Stop' | 'Close' | 'Open' | 'ESD' | 'Start' | 'Restart
      ' | 'Sleep' | 'Trip' | 'Activate' | 'Inactivate' | 'ShutDown'
       | 'InhibitPermissive';

attributes = 'Stopped' | 'Closed' | 'Opened' | '
      AlarmTemperatureHigh' | | 'AlarmTemperatureLow' | '
      AlarmPressureHigh' | 'AlarmPressureLow' | ...
 (* plus additional attributes not depicted here *)

cause = (keyword | keywordPipe | keynum), '.', (functions |
      attributes);

traversal = '->', (keyword | keywordPipe | keynum | wildCard),
      connectionList;
connectionList = [('/' | '\', (keyword | keywordPipe | keynum |
      wildCard), connectionList];

action = '=>', (keyword | keywordPipe | keynum | wildCard), '.',
      (functions | attributes);

rule = cause, traversal, action;
```

Users can edit imported topology models (Topology Object Library) in Fig. 4 using a Topology Editor or serialize them to the AutomationML/CAEX (IEC 62714) XML file format[4] for further processing. CAYENNE provides an Interlock Rule Engine to traverse topology models and apply domain-specific rules from a database to generate C&E matrices or directly IEC 61131-3 structured text in the PLCopen XML file format [5]. The rule engine accepts rules specified according to the domain-specific grammar in Listing 2. The tool uses the visitor design pattern to implement the model traversal by tracing the piping and instrumentation structure imported from P&IDs. Users can configure pre-defined rule sets as well as specify new project-specific rules using a textual Rule Editor. We implemented the rule grammar as well as semantic checks using the Irony .NET language implementation kit [6].

Fig. 5 depicts a simplified view of the CAYENNE topology model class hierarchy. It contains vessels, sensors, controllers, flow objects, termination points and actuators and thus can express most elements typically used in plant engineering. An instance of the model derived from an object-oriented P&ID specification includes connections between the elements (i.e., material flow and information flow) as well as numerous properties and references to graphical shapes and coordinates. Users of the CAYENNE rule engine can refer to any element from the topology model when defining new rules.

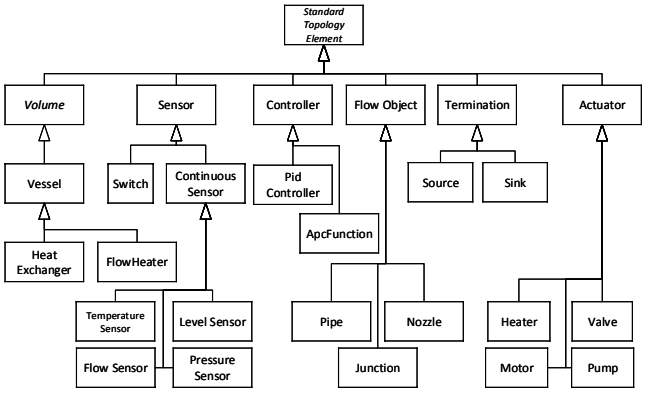

Figure 5: CAYENNE Topology Model (simplified): target for P&ID import, basis for rule language grammar and rule implementation.

The CAYENNE tooling also includes a prototypical process graphics generator, and a simulation generator for factory acceptance tests (FAT). The former interfaces with a commercial tool for process graphics engineering, where additional details can be added manually. The latter generates Modelica [7] files and interfaces with OpenModelica [8] to execute low fidelity plant simulations [9].

5 CASE STUDY METHODOLOGY

5.1 Goal/Question/Metric

We formulated the main goal for evaluating the CAYENNE rule-based approach using the Goal-Question-Metric (GQM) template [10]: "Determine (purpose) the practicability (issue) of a rule-based control logic generation approach (object) for a process engineer (viewpoint) in the context of designing interlocks for automation of industrial plants". A secondary goal was to create an initial rule set for testing and later refinement.

Fig. 6 shows the questions and corresponding metrics we defined in order to achieve the main goal. M1 simply provides the fraction of interlocks that could be generated with the CAYENNE rule language, compared to interlocks that would require different

Goal: Determine practicability of rule-based approach for process engineer		
Question	RQ1	How suitable are rules to generate control logic?
Metric	M1	Percentage of generatable interlocks from rules in a given case
Question	RQ2	How reusable are rules for control logic generation?
Metric	M2	Percentage of rules classified as 'generic'
Metric	M3	Percentage of rules generating multiple interlocks
Metric	M4	Percentage of rules generating interlocks in more than one case
Question	RQ3	How much engineering efforts can be saved?
Metric	M5	Percentage of estimated engineering effor savings
Question	RQ4	How do practitioners perceive the usefulness?
Metric	M6	[Qualitative Feedback from Domain Experts]

Figure 6: GQM model for CAYENNE Validation

means for rule specification. Our target value for M1 was 70 percent, because the approach may lack user acceptance in case it only addressed a low fraction of interlocks.

M2-M4 characterize the reusability of the derived rules. M2 quantifies our own categorization of rules into 'generic' and 'process-specific' rules, with the expectation that 90 percent of the rules will be reusable. This value must be high, since unique rules may not contribute to effort savings. M3 states how many rules generated multiple interlocks, even if that occurred only in a single case (target: 90 percent). Finally, M4 provides the fraction of rules that already generated interlocks in multiple of our case studies, thereby validating their reusability.

Research question RQ3 asked for the efforts savings from the code generation approach. For M5, we rely on effort estimations taking the results of M2-M4 into account, since we have not yet applied the rules in a new project to actually measure the effort savings. Finally, RQ4 closes the loop with practicing process engineers and asked for feedback regarding the perceived usefulness of the approach. To obtain values for the defined metrics, we executed four case studies following guidelines for case study research [24].

5.2 Case Sampling

We conducted a purposive case sampling augmented with convenience sampling [19]. ABB business units provided P&IDs and C&E matrices from four production plants. External engineering contractors had created these specifications manually and ABB had developed control logic and process graphics for these plants based on them. The specifications were three to six years old. In the meantime, each of the plants had been erected and was in production. Our case studies were thus post-mortem analyses to determine how much manual work could have been automated with a rule-based approach.

Due to the business sensitivity for the plant owners, the specifications are subject to numerous non-disclosure agreements (NDA) and the artifacts reside in various local storages. Therefore, we let our ABB business units select cases they considered relevant and interesting for code generation. From these, we selected cases representing different application domains (e.g., chemical production, oil refineries) to explore cross-case rule applicability.

5.3 Data collection

We divided our data collection into two phases:

Phase 1: Rule mining. For the four cases, the domain experts, who provided the original specifications for the plants' interlocks, were largely unavailable (i.e., coming from different companies, now working on different projects). Hence, we decided to perform the rule mining ourselves by analyzing the patterns underlying the specified interlocks. The originally specified interlocks in the respective C&E matrices serve as the 'ground truth' for the rule-based generation.

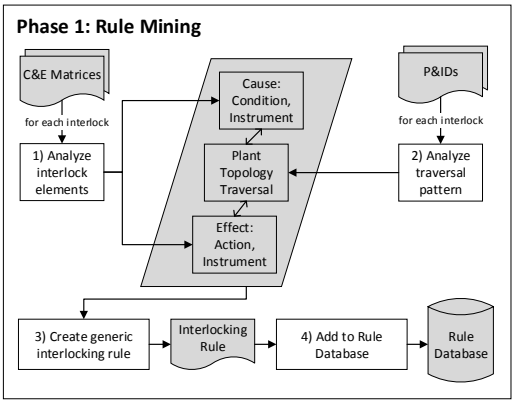

Figure 7: Phase 1: Rule Mining

Fig. 7 shows the four steps of our rule mining (Phase 1). In Step 1), we first analyzed the interlock elements specified in the C&E matrices of each case. For each interlock, we identified the *cause* including an instrument and a condition as well as the *effect* including a list of instruments and actions. For example, the low-level alarm condition of a sensor on a tank would cause the stopping of a pump on the tank outlet as an effect. In Step 2), we located both instruments in the corresponding P&IDs and determined the plant topology *traversal* path between causing instrument and effecting instrument. In the previous example, this would be the traversal path between the tank and the pump. In Step 3), we expressed the traversal path in terms of elements of the topology model abstracted from concrete instances. In Step 4), the triple of *cause, traversal, and effect* either matched with an existing interlocking rule, or we created a new rule and added it to our rule database.

Phase 2: Rule validation. In this phase (not depicted here), we executed an additional rule validation. In Step 5), we first manually re-drew the original PDF-based P&IDs using Microsoft Visio to have smart P&IDs as input. Note that the redrawing step is for research purposes only and once XML-based export formats for P&IDs are available, this step can be omitted, and the exported XML files can be directly imported into a topology model. In Step 6), the Visio P&IDs were mapped into a topology model based on CAEX using a self-implemented importer tool. In Step 7), the CAYENNE rule engine applied the rules from the developed rule database in phase 1 on the topology model. This results in an automatically generated C&E matrices, which get compared against the original, manually-specified C&E matrices in Step 8).

6 CASE ANALYSIS

6.1 Overview

Table 1 characterizes the four analyzed cases.

The specifications of each plant included input/output (IO) lists with 400 - 7000 IO points, i.e., digital/analog input and output

signals. According to Forbes and Clayton [13], systems with 900-2300 IO points are considered 'medium-sized' (45 percent of the overall market), while systems with more than 2300 IO points are considered 'large-sized' (16 percent of the overall market).

Property	Plant 1	Plant 2	Plant 3	Plant 4
I/O points	1000	4000	400	7000
P&ID notation	ISO	ISA	ISA	ISA
P&ID native file format	.dwg	.dwg	?	?
Number of P&IDs available	50	116	8	134
Number of P&IDs analyzed	10	23	8	12
Number of vessels in analyzed P&IDs	18	23	8	4
Number of pumps in analyzed P&IDs	9	27	13	12
Number of C&E matrices available	12	9	1	54
Number of C&E matrices analyzed	2	7	1	~30
Number of analyzed interlocks	125	58	100	89
Number of rules to generate interlocks	28	14	19	21
Number of rules for regular equipment	19	14	8	21
Number of rules for special equipment	9	0	11	0

Table 1: Properties of the analyzed cases.

P&IDs were available for the cases either in the ISO 10268 or ANSI/ISA 5.1 notations. Their native file format was AutoCAD drawing (.dwg), but all files were available as PDF exports only. The number of available P&IDs per case ranged corresponding to their size (e.g., in the small-sized Plant 3, we had 8 diagrams, while we had 134 diagrams for the large Plant 4). We scoped our analysis to those diagrams that represented the main material flow of the underlying process and suggested the biggest learning effects for rule mining. The selected P&IDs per plant contained 4-23 vessels (i.e., tanks and heat exchangers) and 9-27 pumps.

Each case had hundreds to thousands of interlocks specified in C&E matrices. The C&E matrices also contained versioning information as well as free text annotations, which however did not affect the interlock behavior. We found several typing errors, where signal references were misspelled, as well as inconsistencies between the P&IDs and the C&E, where tags specified in the C&E were missing in the referenced P&ID. These specifications were meant for manual interpretation, where humans can compensate for errors to some extent. For a fully automated interlocking tool chain these documents would need to be specified according to standards (e.g., ISO 15926) and validated for consistency.

We did not analyze all of the thousands of interlocks in detail, but selected between 50-100 interlocks per case corresponding to the main material flow to achieve a representative coverage. We classified the equipment in the cases into regular equipment (e.g., pumps, valves, pipes, heat exchangers, controllers) and special equipment (e.g., conveyors, power units, special vessel features). Rules involving special equipment (also see Table 2, column 4) may have lower reusability across plants. Thus, this classification provides a rough measure for the uniqueness of the plant under analysis.

In the following subsections, for space reasons we discuss findings for two selected cases (Plant 2 and 3), before summarizing the results of the overall rule mining in Section 7.

6.2 Plant 2

Plant 2 is chemical plant from South America The plant automation includes more than 4000 IO points. The case included 116 P&IDs, which however were not all relevant for interlocking logic. The P&IDs included detailed specifications for the involved vessels (e.g., diameters, volume, operating conditions) as well as alarm limits and interlock references. They also included complex piping structures. Over the course of 2.5 years, the P&IDs had gone through multiple revisions. The process featured various replicated equipment, e.g., duplicated vessels or duplicated pumps.

The interlocking logic specification consisted of nine C&E matrices. These included a high number of permissives and inhibits, which specify the starting conditions for the whole process. Instead of only linking signal references using boolean logic, the matrices contained 'actions' as cell entries, e.g. stopping a pump or closing a valve. From the C&E matrices we selected only entries related to measurement instruments, which are considered interlocks during operation, while the permissives and inhibits rather pertain start-up and shut-down procedures.

For deriving the interlocking rules of the 58 selected interlocks, we had to analyze 23 of the 116 P&IDs. Despite several complex piping structures most of the interlocks could be mapped to simple interlocking rules. We created a new interlocking rule for anti-surge control of compressors in this plant. Fig. 8 shows the (simplified) P&ID structure this rule was based on. When valve V1 opens the corresponding check valve V2 needs to close. However, a generic rule should not match any two valves on connected pipes in the plant, but only match those valves on an outlet of a compressor. So the rule created for this case first starts a topology traversal from the cause V1 and checks whether a controller and compressor is connected. Then, it traverses in the opposite direction of the material flow back to a junction point, where the material flow is inverted. Afterwards, it matches the first valve on this outflow.

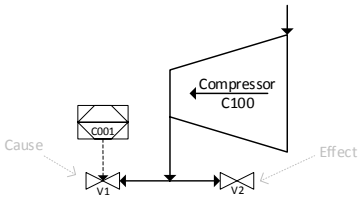

Figure 8: Example for a non-trivial traversal pattern from plant 2 between cause (valve V1) and effect (valve V2).

While such a traversal initially appears to be a rather special occurrence, the rule is applicable 14 times in plant 2. It is applied for multiple times for each outlet of compressor and there are two such compressors in the plant segment. Furthermore, the rule is triggered by other causes (e.g., level and temperature sensors), which open valve V1. The underlying traversal pattern is typical for anti-surge control in industrial compressors and therefore not plant-specific [21].

All analyzed interlocks in plant 2 could be generated with rules. This plant did not include any special equipment, but for one complex piping structure we needed a special rule that we classified as process-specific.

6.3 Plant 3

Plant 3 is an oil refinery plant from the Middle East. The process consists of two main steps, a dehydration and a subsequent desalting of crude oil. The products of this separation process are refined oil, natural gas, and water. Special types of vessels are used in the process that are supplied by electrical power units to drive the electrostatic separation process. In addition, different chemicals are used in the process to avoid clogging of pipes and reduce the amount of dissolved oxygen from the crude oil. The automation of the process includes about 400 IO points. The main refinery process is captured in 8 P&IDs of which 6 were analyzed in detail.

The interlocks for the case were captured in a single C&E matrix with roughly 40 causes and 30 effects and a total of 180 C&E entries. 75 of these interlocks were related to the handling of severe failures, such as a plant power failure or an emergency shutdown of the plant. Furthermore, 4 interlocks were triggering an acoustic alarm in case one of the dosing tanks of the additionally supplied chemicals raises a high-level alarm. We decided to exclude the C&Es for these two categories and focused on the process-relevant interlocks. In sum, these were 100 remaining interlocks, with causes being alarms or failure events of process equipment as well as effects being either tripping events for pumps and power units, closing of controlled valves, or inhibits for the startup of a device.

After manually studying the P&IDs and understanding the flow of the different materials involved, the definition of rules showed to be rather straightforward. The piping at first seemed more complicated than it actually was, e.g., because of the large number of redundant connections between the separation vessels and the water drain system. Due to the fact that a separator vessels is a combination of a vessel and an electrically powered actuator, multiple C&Es had an emergency shutdown procedure of the separator as an effect. To capture this special type of cause, new rules had to be added. Besides these special cases, a number of generic pump/vessel-related rules of the previous plant cases could be reused. A few new pump/vessel-related rules were added which are specific to the refinery process, although, they only involve regular equipment. Fig. 9 shows a case that occurred eight times in the plant and could be expressed by a single, yet non-trivial, rule. Here, a high-pressure alarm of a sensor causes the stop of the water pumps feeding into the material flow before a vessel.

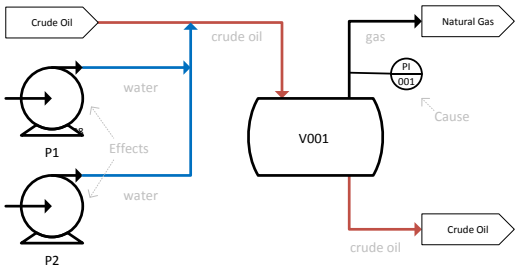

Figure 9: Example for a non-trivial traversal pattern from plant 3 between cause (sensor PI1) and effect (pumps P1/P2).

While this traversal seems complicated first, the key to the rule definition was to limit the matching of pipes with specific media types.

Out of the 100 interlocks of plant 3, we found 6 interlocks which we were not able to formulate adequate rules. We would have immediately caused false positives, meaning that the rules would have resulted in new interlocks that were not part of the original C&E matrix of the plant. The remaining 94 interlocks could be generated from only 19 rules. 14 of these rules were specific to plant 3, with 10 of them being specific to the power units supplying the separator vessels. While this could be considered a special equipment, compared to the other plants, it is a common type of equipment for oil refineries in general.

7 CROSS-CASE FINDINGS

7.1 Mined Rule Set

Table 2 provides an overview of the 92 CAYENNE rules mined in the four case studies from which we generated 336 interlocks in total. The actual rules are not shown for brevity and confidentiality reasons in case of process-specific interlocks. The table sorts the rules according the instrument that represented the cause for an interlock. There are rules for different sensors (e.g., for flow, level, position, pressure, temperature), while other interlocks are caused by actuators (e.g., valves, pumps, switches).

We classified the rules into 55 'generic' and 37 'process-specific' rules, where the latter referred to rules with traversal clauses containing more than three elements. We expect that the generic rules are applicable for different cases, even if that did not occur in the four heterogeneous cases we analyzed, as they represented different processes. However, their traversal patterns follow plausible physical principles and could therefore occur similarly even for different kinds of plants if similar equipment is involved.

The table shows the number of times a rule applied for each case, rules that matched many times are highlighted in green for easier identification.

7.2 Effectiveness

Table 3 depicts the metrics M1-M4 measuring the effectiveness of the rules-based approach to answer RQ1 and RQ2. M1 indicates that, on average, 97% of the analyzed interlocks could be generated with rules. The evaluated cases contained only few interlocks that are beyond the current expressiveness of the CAYENNE rule language. This result was above our target level of 80% and thus exceeded our expectations.

M2 shows that on average we classified 73% of the rules in each as 'generic' while 27% were 'process-specific'. This means that most of the rules have simple and straightforward causes, effects, and traversal patterns, which increases the probability of being reusable across different plants. However, 73% of generic rules lies below our target level of 90%. A process or knowledge engineer would need to invest some time in defining new rules or manually specifying interlocks to address the remaining 27% given a new plant specification to account for process specifics. M2's value of 73% may however motivate the need for more case studies or pilot projects to extend the rule base.

M3 indicates that 68% of the 91 rules generated multiple interlocks. This includes cases where a rule applied multiple times for a single plant specification (e.g., if there are multiple similar pumps in the same plant). Using rules instead of manual specification is

Category	Ref	Classes	Equipment Type	Plant 1	Plant 2	Plant 3	Plant 4	SUM
Flow	FFHH1	Generic	Special Equipment	2	0	0	0	2
Flow	FFLL1	Generic	Special Equipment	2	0	0	0	2
Flow	FSLL1	Generic	Regular Equipment	2	0	0	0	2
Level	LSHH1	Generic	Regular Equipment	3	4	0	0	7
Level	LSHH2	Process-specific	Regular Equipment	1	0	0	0	1
Level	LSHH3	Generic	Regular Equipment	0	2	18	4	24
Level	LSHH4	Generic	Special Equipment	0	0	4	0	4
Level	LSHH5	Process-specific	Special Equipment	0	0	2	0	2
Level	LSHH6	Generic	Regular Equipment	0	8	0	0	8
Level	LSHH7	Generic	Special Equipment	1	0	0	0	1
Level	LSLL1	Generic	Regular Equipment	1	4	14	8	27
Level	LSLL2	Generic	Regular Equipment	2	1	1	0	4
Level	LSLL3	Generic	Regular Equipment	3	0	0	0	3
Level	LSLL4	Process-specific	Special Equipment	1	0	0	0	1
Level	LSLL5	Process-specific	Regular Equipment	1	0	0	0	1
Level	LSLL6	Generic	Regular Equipment	0	2	0	0	2
Level	LSLL7	Generic	Special Equipment	0	0	4	0	4
Level	LSLL8	Generic	Special Equipment	0	0	4	0	4
Level	LSLL9	Generic	Special Equipment	0	0	4	0	4
Level	LSLL10	Process-specific	Regular Equipment	0	0	2	0	2
Level	LSLL11	Generic	Regular Equipment	7	0	0	0	7
Level	LSLL12	Generic	Regular Equipment	4	0	0	0	4
Level	LSLL13	Generic	Regular Equipment	2	0	0	0	2
Position	ZSL1	Generic	Special Equipment	4	0	0	0	4
Position	ZSR1	Generic	Special Equipment	4	0	0	0	4
Pressure	PSHH1	Generic	Regular Equipment	1	0	0	0	1
Pressure	PSHH2	Generic	Regular Equipment	0	8	0	0	8
Pressure	PSHH3	Process-specific	Regular Equipment	0	1	0	0	1
Pressure	PSHH4	Process-specific	Regular Equipment	0	2	0	0	2
Pressure	PSHH5	Process-specific	Regular Equipment	0	0	8	0	8
Pressure	PSHH6	Generic	Regular Equipment	0	0	4	5	9
Pressure	PSHH7	Generic	Special Equipment	0	0	2	0	2
Pressure	PSHH8	Process-specific	Regular Equipment	0	0	0	6	6
Pressure	PSHH9	Process-specific	Regular Equipment	0	0	0	6	6
Pressure	PSHH10	Process-specific	Regular Equipment	0	0	0	2	2
Pressure	PSHH11	Process-specific	Regular Equipment	0	0	0	4	4
Pressure	PSLL1	Generic	Regular Equipment	0	1	2	4	7
Pressure	PSLL2	Generic	Regular Equipment	0	5	0	1	6
Pressure	PSLL3	Generic	Regular Equipment	0	2	0	0	2
Pressure	PSLL4	Process-specific	Regular Equipment	0	0	0	6	6
Pressure	PSLL5	Process-specific	Regular Equipment	0	0	0	6	6
Pressure	PSLL6	Process-specific	Regular Equipment	0	0	0	2	2
Pressure	PSLL7	Process-specific	Regular Equipment	0	0	0	4	4
Pressure	PSLL8	Process-specific	Regular Equipment	0	0	0	4	4
Pump	JI1	Generic	Regular Equipment	1	0	0	0	1
Pump	JI3	Process-specific	Special Equipment	1	0	0	0	1
Pump	JI4	Process-specific	Special Equipment	1	0	0	0	1
Pump	JI5	Generic	Special Equipment	6	0	0	0	6
Pump	JI6	Generic	Regular Equipment	0	0	6	0	6
Pump	JI7	Process-specific	Regular Equipment	0	0	0	1	1
Pump	JI9	Generic	Special Equipment	2	0	0	0	2
Pump	JI10	Process-specific	Special Equipment	1	0	0	0	1
Pump	JI11	Process-specific	Special Equipment	2	0	0	0	2
Pump	JI12	Process-specific	Special Equipment	1	0	0	0	1
Speed	SSL1	Generic	Special Equipment	6	0	0	0	6
Switch	HS1	Generic	Special Equipment	4	0	0	0	4
Switch	HS2	Generic	Special Equipment	0	0	4	0	4
Switch	HS3	Generic	Special Equipment	0	0	4	0	4
Switch	HS4	Generic	Special Equipment	0	0	2	0	2
Switch	HS5	Generic	Special Equipment	1	0	0	0	1
Switch	HS6	Process-specific	Special Equipment	2	0	0	0	2
Switch	HS7	Process-specific	Special Equipment	1	0	0	0	1
Switch	HS8	Process-specific	Special Equipment	1	0	0	0	1
Switch	HS9	Generic	Special Equipment	1	0	0	0	1
Switch	HS10	Generic	Special Equipment	2	0	0	0	2
Switch	HS11	Generic	Special Equipment	4	0	0	0	4
Switch	HS12	Generic	Special Equipment	2	0	0	0	2
Switch	HS13	Generic	Special Equipment	2	0	0	0	2
Switch	HS14	Generic	Special Equipment	4	0	0	0	4
Switch	HS15	Generic	Special Equipment	2	0	0	0	2
Switch	HS16	Generic	Special Equipment	1	0	0	0	1
Temperature	TSHH1	Generic	Regular Equipment	0	0	0	0	0
Temperature	TSHH2	Process-specific	Regular Equipment	1	0	0	0	1
Temperature	TSHH3	Process-specific	Regular Equipment	1	0	0	0	1
Temperature	TSHH4	Generic	Special Equipment	0	0	4	0	4
Temperature	TSHH5	Generic	Regular Equipment	0	4	0	0	4
Temperature	TSHH6	Process-specific	Regular Equipment	0	0	0	1	1
Temperature	TSHH7	Process-specific	Regular Equipment	0	0	0	1	1
Temperature	TSHH8	Process-specific	Regular Equipment	0	0	0	1	1
Temperature	TSLL1	Process-specific	Regular Equipment	1	0	0	0	1
Temperature	TSLL2	Generic	Regular Equipment	1	0	0	0	1
Transformer	TR1	Generic	Special Equipment	0	0	4	0	4
Valve	YS1	Generic	Regular Equipment	8	0	0	0	8
Valve	YS3	Generic	Regular Equipment	1	0	0	0	1
Valve	YS5	Process-specific	Regular Equipment	1	0	0	0	1
Valve	YS6	Process-specific	Regular Equipment	7	0	0	0	7
Valve	YS7	Process-specific	Regular Equipment	5	0	0	0	5
Valve	YS8	Generic	Regular Equipment	0	14	0	0	14
Valve	YS9	Process-specific	Regular Equipment	0	0	0	1	1
Valve	YS10	Process-specific	Regular Equipment	0	0	0	1	1
Valve	YS11	Generic	Regular Equipment	0	0	0	1	1
Valve	YS12	Generic	Regular Equipment	0	0	0	1	1
			Sum of interlocks generated with rules:	115	58	93	70	336
			Number of rules applied:	48	14	19	22	103
			Interlocks without rules defined (unclear/complicated):	10	0	7	19	36
			Percentage of intelocks addressed by rules	92%	100%	93%	79%	91%
			Sum of interlocks generated with "generic" rules:	86	55	81	24	246
			Percentage of intelocks addressed by "generic" rules	**75%**	**95%**	**87%**	**34%**	73%

Table 2: Results from Rule Mining: Categories of rules, short reference, classification, equipment types, number times rules applicable per plant.

Metric	Description	Target	Actual	Plant 1	Plant 2	Plant 3	Plant 4
M1	% of interlocks that can be generated with rules	80%	**91%**	92%	100%	93%	79%
M2	% of rules classified as "generic"	90%	**73%**	75%	95%	87%	34%
M3	% of rules generating multiple interlocks	90%	**68%**				
M4	% of rules matching in multiple cases	50%	**7%**				

Table 3: Metrics M1-M4 collected from the Case Studies

only justified if a rule is used multiple times, otherwise the effort to construct the rule is higher than simply defining the interlock directly. Most of the rules matching only once were also classified as 'process-specific'. However, it is likely that M3 would increase in case more specifications were analyzed, which would provide a higher chance of their reuse.

For M4, we found that only 7% of our 91 rules were applicable for more than one of our analyzed cases, which was way below our target level of 50%. However, while there were only few such rules, they actually matched a high number of times. These rules refer to 'traversal patterns' that occur often, thus they reduce the manual effort significantly. We conjecture that the low value for M4 is also caused by the heterogeneity of our four cases, which both lie in different industrial automation domains and contain different kinds of equipment. Analyzing more similar cases (e.g., plants for similar production processes) could lead to a higher number of rules matching in multiple cases. Furthermore, domain experts for specific processes may specify more reusable rules from their experience.

7.3 Estimated Effort Reduction

For estimating the effort reduction (RQ3) of the CAYENNE rule-based approach compared to a conventional state-of-the-art approach, we relate back to the workflow depicted in Fig. 10. Each of the artifacts in the figure requires efforts from the engineering contractor or automation provider. We show the estimated distributions of these efforts on a percentage scale in Fig. 10, left column. For example, the engineering contractor's specification of an IO list may take around 8% effort of the overall activities, while the P&ID specification would require 17%. We do not express the efforts in person hours or costs, since there are vastly different project sizes as well as local salaries.

The column on the right hand side of Fig. 10 shows the estimated effort distribution taking the CAYENNE rule-based approach into account. The new process does not alter the creation of I/O lists, P&IDs, logic diagrams, or control narratives, therefore these efforts remain constant. Based on our case study results (M1-M4), the efforts for the creation of C&E matrices could get reduced by 80% (i.e., from 8% of the total effort to 2%). A full elimination of the manual work for this step is unlikely, since there will still be several special interlocks to be specified manually in each case, there is the need for manually customizing the rule base before applying the

rules, and the generation results need to be reviewed and approved manually as before for safety reasons.

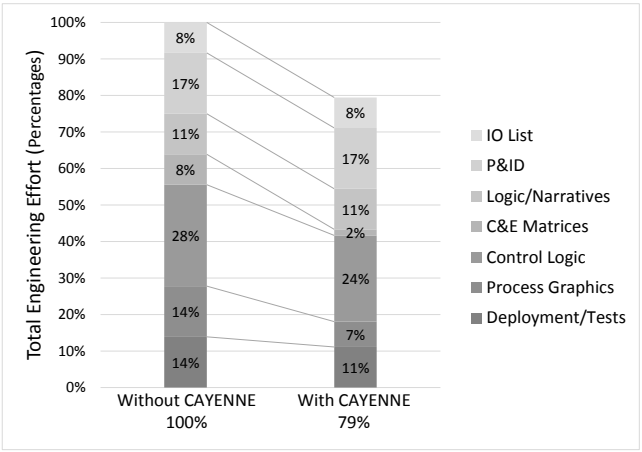

Figure 10: Estimated Effort Reduction for Engineering

Creating the control logic could be reduced by around 15% (i.e., from 28% to 24%). Interlocks comprise up to 30% of the overall control logic, which for example also include PID loops, sequences, and monitoring functions. For these 30% we estimate an effort reduction of up to 50% based on the case studies. As the rule-based code generation covers a large portion of the interlocking control logic, there is a reduced need for manual implementation. The generation based on tool-validated smart P&IDs also removes the potential for inconsistencies in the specifications, which often lead to time-consuming feedback cycles between 'engineering, procurement, and construction' contractors (EPC)s and automation providers.

The efforts for process graphics creation could be reduced by around 50% (i.e., from 14% to 7%) due to generation from smart P&IDs instead of manual specification. Finally, the efforts for deployment and testing get reduced by 20% (i.e., from 14% to 11%), since the automatic code generation reduces the potential for manual errors and shifts testing to systematic errors in the generation. Overall, the efforts reduction sums up to around 21%, which means a project where these activities cost 500 KUSD could potentially save 105 KUSD.

7.4 Domain Expert Feedback

We discussed our case study results with five domain experts, who are regularly involved with requirements specification and control logic engineering for the targeted production processes. All domain experts deemed the CAYENNE rule-based approach as potentially useful and supportive for their work, because it automates specific manual steps. The domain experts emphasized the safety criticality of interlocks, which requires careful manual review and formal approval by safety experts. The generation process must respect this context and provide the involved engineers detailed feedback on the generation and allow humans to override the rule engine if needed. An engineering contractor wants to retain full control of the interlock specification due to the involved safety accountability.

The domain experts also pointed out that some rules may need more refinement. For example, a rule checking for an overflow tank may require taking the severity level of the tank into account. A simple non-critical water tank would need a different treatment than a tank dealing with acids, which may harm equipment and humans, and thus potentially not require the same interlocks. They also raised concerns that a rule-based generation could end up in "too many" interlocks if the rule base was not carefully configured, which a human engineer would have not specified due to experience. Furthermore, they mentioned that there might be additional interlocks that may be hard to capture with the current rule specification language. Nevertheless, the domain experts encouraged extensions, refinements, and additional case studies.

7.5 Result Validity

We discuss the construct, internal, and external validity of our case studies. The construct validity refers to the appropriateness of the artifacts and procedures in the case studies to resemble realistic settings. In our case, the construct validity is supported by the use of real plant specifications for non-trivial production plants. We did not use customer-specified smart P&IDs so far due to missing tool support, but created according ones based on the original P&IDs in MS Visio. We had researchers carrying out the rule specification and execution of the rule engine, which are not the actual target users. However, we used iterative domain expert feedback to improve the design of the rule language and tooling.

To assure internal validity, we compared the interlocks generated with the rule-based approach with formerly manually specified interlocks in C&E matrices. This provides an initial validation of our rule engine. However, when the approach and the rule set is extended with more features and additional rules, a more systematic testing for false positives, contradicting rules, and possibly redundant rules is required.

The external validity refers to the transferability of the results to other situations. To improve the external validity, we analyzed four different cases and found that a number of generated interlocks apply across different plants (Metric M3). Still, more cases in additional application domains need to be analyzed to improve the external validity further. It would also be helpful to focus a future study on multiple more homogeneous plant specifications (e.g., 10 processes specifically for carbonic acid production), which could provide a refined view of 'generic' and 'process-specific' rules.

8 RELATED WORK

Automated code generation is one element of model-driven software development, which has been researched in general software engineering [25], as well as specifically for industrial automation software applications [20, 29, 30].

Researchers proposed several methods for *rule-based control logic generation*. Drath and Fay [12] generate C&E matrices from AutomationML files, which can include P&ID-related information. Steinegger et al. [26] follow a similar approach, but take more kinds of requirements specifications into account and generate control logic for different purposes. Grüner and Epple [15] use the Neo4J graph database to represent a plant topology and generate control logic using graph queries. None of these approaches was applied in

larger case studies nor featured a custom-designed rule language as the CAYENNE method.

Researchers have also worked on methods to *generate IEC 61131-3 control logic from UML diagrams* instead of smart P&IDs [20]. This requires formulating EPC requirements using UML, which goes against current practices, as process engineers are not familiar with the UML notation. Vogel-Heuser et al. [28] implemented code generators from UML class and activity diagrams for a commercial CASE tool. Thramboulidis and Frey [27] sketched a model-driven development process using P&IDs and SysML to generate IEC 61131-3 code. Hästbacka et al. designed an UML Automation profile to express EPC requirements in UML and generated IEC 61131-3 code from the UML models. These approaches require an intermediate UML representation, while the CAYENNE approach directly extracts control logic code out of EPC requirements.

Automated model or code generation from *natural language requirements* is another area of software engineering research. As the requirements for software applications are mostly formulated in natural language instead of semi-formal notations, such as C&E matrices, it is much harder to use them in a code generation tool chain. Gelhausen and Tichy [14] generate UML models from constrained natural language representations. Deeptimahanti et al. [11] implemented a tool for the generation of UML models by identifying classes and stereotypes in natural language specifications. These approaches are mostly constrained by the capabilities of information retrieval and text mining techniques.

9 CONCLUSIONS

This paper has reported on findings from four case studies, where the rule-based CAYENNE approach automatically generated IEC 61131-3 control logic by analyzing formal requirements documents (P&IDs). We have mined more than 90 code generation rules from the cases to assess the practicability of rule-based generation approaches in general. The results indicate that potentially more than 70 percent of the interlocking logic could be generated from rules that are applicable across plants, although our study has demonstrated this only to a lesser extent due to the heterogeneity of the analyzed cases. The cost savings potential was estimated to be approximately 21 percent.

Practitioners can utilize the results to implement their own rule-based generation approaches taking into account the expected cost savings. Furthermore, due to the promising results, they could motivate their customers to utilize standard notations for requirements specification, such as the DEXPI ISO 15926 format for P&IDs. Researchers can extend existing methods or devise new methods for rule-based control logic generation. Interesting areas of research are for example code generation rules for other parts of control logic, beyond interlocks. Topological models derived from smart P&IDs can also be exploited for simulation, HMI generation, or training operator assistents.

As future work, we will enhance and extend the CAYENNE approach to increase its technology readiness level. An importer for the DEXPI ISO 15926 will enable processing P&IDs from all major CAD-tools. Additional case studies target extending and refining the mined rule set. Investigating more similar plants could yield tailored rule sets for specific domains. Rules need to be established for other kinds of control logic to save more engineering costs.

REFERENCES

[1] [n.d.]. https://de.codesys.com/.
[2] [n.d.]. https://www.beckhoff.de/twincat3/.
[3] [n.d.]. https://dexpi.org.
[4] [n.d.]. https://www.automationml.org.
[5] [n.d.]. https://plcopen.org/.
[6] [n.d.]. https://github.com/IronyProject/.
[7] [n.d.]. https://www.modelica.org/.
[8] [n.d.]. https://openmodelica.org/.
[9] Esteban Arroyo, Mario Hoernicke, Pablo Rodríguez, and Alexander Fay. 2016. Automatic derivation of qualitative plant simulation models from legacy piping and instrumentation diagrams. *Computers & Chemical Engineering* 92 (2016), 112–132.
[10] Victor R Basili1 Gianluigi Caldiera and H Dieter Rombach. 1994. The goal question metric approach. *Encyclopedia of software engineering* (1994), 528–532.
[11] Deva Kumar Deeptimahanti and Muhammad Ali Babar. 2009. An Automated Tool for Generating UML Models from Natural Language Requirements. In *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering (ASE '09)*. IEEE Computer Society, Washington, DC, USA, 680–682. https://doi.org/10.1109/ASE.2009.48
[12] Rainer Drath, Alexander Fay, and Till Schmidberger. 2006. Computer-aided design and implementation of interlock control code. In *IEEE Conference on Computer Aided Control System Design*. IEEE, 2653–2658.
[13] Clayton D. (ARC Advisory Group) Forbes, H. 2018. Distributed Control Systems Global Market 2017-2022. ARC Market Analysis. https://www.arcweb.com/market-studies/distributed-control-systems
[14] Tom Gelhausen and Walter F Tichy. 2007. Thematic role based generation of UML models from real world requirements. In *International Conference on Semantic Computing (ICSC 2007)*. IEEE, 282–289.
[15] Sten Grüner, Peter Weber, and Ulrich Epple. 2014. Rule-based engineering using declarative graph database queries. In *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*. IEEE, 274–279.
[16] Georg Gutermuth. 2010. *Collaborative Process Automation Systems*. ISA, Chapter Engineering, 156–182.
[17] David Hästbacka, Timo Vepsäläinen, and Seppo Kuikka. 2011. Model-driven development of industrial process control applications. *Journal of Systems and Software* 84, 7 (2011), 1100–1113.
[18] Martin Hollender. 2010. *Collaborative process automation systems*. ISA.
[19] Barbara Kitchenham and Shari Lawrence Pfleeger. 2002. Principles of survey research: part 5: populations and samples. *ACM SIGSOFT Software Engineering Notes* 27, 5 (2002), 17–20.
[20] Heiko Koziolek, Andreas Burger, Marie Platenius-Mohr, and Raoul Jetley. 2020. A Classification Framework for Automated Control Code Generation in Industrial Automation. *Submitted to Elsevier Journal of Systems and Software* (2020).
[21] Terje Kvangardsnes. 2009. *Anti-surge control: Control theoretic analysis of existing anti-surge control strategies*. Master's thesis. Institutt for teknisk kybernetikk.
[22] Bela G Liptak. 2018. *Instrument Engineers' Handbook, Volume Two: Process Control and Optimization*. CRC press.
[23] Martin Naedele. 2012. ABB's software is everywhere. *ABB Review* 3 (2012).
[24] Per Runeson and Martin Höst. 2009. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering* 14, 2 (2009), 131.
[25] Thomas Stahl, Markus Voelter, and Krzysztof Czarnecki. 2006. *Model-driven software development: technology, engineering, management*. John Wiley & Sons, Inc.
[26] Michael Steinegger and Alois Zoitl. 2012. Automated code generation for programmable logic controllers based on knowledge acquisition from engineering artifacts: Concept and case study. In *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA 2012)*. IEEE, 1–8.
[27] Kleanthis Thramboulidis and Georg Frey. 2011. An MDD process for IEC 61131-based industrial automation systems. In *ETFA2011*. IEEE, 1–8.
[28] Birgit Vogel-Heuser, Daniel Witsch, and Uwe Katzke. 2005. Automatic code generation from a UML model to IEC 61131-3 and system configuration tools. In *2005 International Conference on Control and Automation*, Vol. 2. IEEE, 1034–1039.
[29] Valeriy Vyatkin. 2013. Software engineering in industrial automation: State-of-the-art review. *IEEE Transactions on Industrial Informatics* 9, 3 (2013), 1234–1249.
[30] Chia-Han Yang, Valeriy Vyatkin, and Cheng Pang. 2014. Model-driven development of control software for distributed automation: a survey and an approach. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 44, 3 (2014), 292–305.