Proc. of the 17th IEEE Int. Conf. on Emerging Technologies and Factory Automation, pages 1–8, 2012

Automated Code Generation for Programmable Logic Controllers based on Knowledge Acquisition from Engineering Artifacts: Concept and Case Study

Michael Steinegger and Alois Zoitl

Automation and Control Institute, Vienna University of Technology,
Gußhausstr. 27-29|E376, AT-1040 Vienna, Austria.
E-mail: {steinegger,zoitl}@acin.tuwien.ac.at

Abstract

The effort for design and implementation of process automation systems increases with the complexity of modern production plants. To shorten the implementation phase we propose a conceptual approach for automated code generation for programmable logic controllers (PLCs) based on knowledge acquisition from standardized engineering artifacts and their corresponding tool database. Ontologies are defined for each engineering domain artifact, representing the concepts and relations specified in the associated domain standards. An automated data extraction and population of the domain-specific ontologies together with semi-automated mapping definitions between common engineering objects then enables a rule-based code generator to query the ontologies. The query results are translated afterward into PLC code by applying a rule transformation algorithm.

1. Introduction

The phases of industrial engineering projects are often carried out in concurrent and distributed ways and involve several disciplines like process, electrical or automation engineering. A fundamental issue in such projects is the seamless exchange of data or information, since different domain expert groups apply their own best-suited engineering tools, producing domain-specific artifacts, and have different views on engineering objects. Semantic integration of heterogeneous tool sets and design data not only improves data exchange, version control, consistency and change management in concurrent and distributed engineering processes. It also enables possibilities to automate engineering tasks in late project phases, like the development of *programmable logic controller* (PLC) applications for industrial plant control.

Process control applications are developed based on plant-specific engineering artifacts, defined by project-involved domain experts in preceding project phases. Engineering artifacts contain information about the physical plant structure, electrical wiring, recipe phases in case of batch processes, control strategies for start-up and shut-down procedures, as well as safety-related specifications like interlocks or emergency plant shut-down. The accuracy of these artifacts has a major impact on the efficiency and quality of PLC-based control applications [5].

To partially automate the development of industrial control applications, we propose a conceptual approach for ontology-supported PLC code generation based on knowledge acquisition from typical engineering artifacts. In contrast to several other existing code generation approaches discussed in Sec. 2, we focus on the integration of digital design data and process information as a basis for control code generation (cf. Fig. 1). The information incorporated in engineering artifacts is represented within domain-specific ontologies and mappings define the translation between overlapping concepts. As a consequence, the integrated knowledge can be inferred by a rule-based reasoning engine in order to generate plant-specific control or defect detection code in a consecutive step.

Outline. The paper is organized as follows. In Sec. 2, related approaches for semantic data integration and automated PLC code generation are summarized. The general concept of the ontology-based data acquisition and rule-based code generation is described in Sec. 3. In Sec. 4, the conceptual approach is illustrated by application to exemplary control tasks for a didactic process plant. Section 5 concludes the paper and future research issues are stated.

2. Related work

This section summarizes related approaches for semantic data integration and automated PLC code generation.

2.1. Distributed data integration approaches

In order to bridge the semantic gaps between different tools involved in engineering projects, Anhäuser et al. [1] introduced PlantXML for data integration across different engineering disciplines. The data model PlantXML is based on four company-specific *extensible markup language* (XML) schemes and therefore differs from the neutral meta model *computer aided engineering exchange* (CAEX) defined in IEC 62424 [13] or the interchange standard ISO 15926 [17] for life-cycle data of industrial process plants.

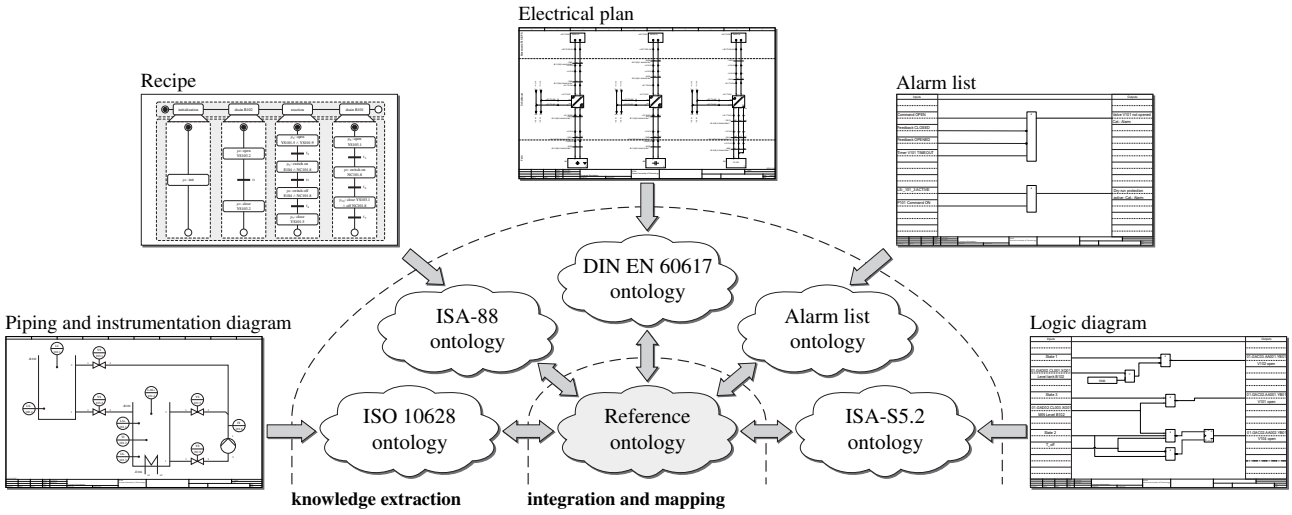

Figure 1. Ontology-based knowledge acquisition from engineering artifacts.

Szulman et al. [31] proposed an ontology-based framework for semantic data integration within collaborative engineering processes. The domain-specific concept models are represented within several ontologies and a superior bridging ontology defines the semantic transformations between overlapping data and concepts. A similar conceptual approach for definition of engineering artifact ontologies was introduced by Kitamura [22].

Integration of distributed design data motivated the development of OntoCAPE [25], a large-scale ontology for computer aided engineering of process plants. Based on PlantXML and OntoCAPE, Morbach and Marquardt [24] elaborated an ontology-based integration framework for chemical process engineering, which also enables version control and inconsistency checking for design data as described by Wiesner et al. [35].

2.2. Automated code generation approaches

To support and improve current programming methods in industrial automation projects, automated code generation for PLC controlled systems became a major research topic. Several approaches focus on the design of automation systems based on discrete event models like *petri nets* (PNs) and transformation of project-specific PNs into IEC 61131-3 [10, Part 3] compliant code like *ladder diagrams* (LDs) or *instruction lists* (ILs). Cutts and Rattigan [3] proposed a PLC code generation method with PN-based modeling techniques for a three-stage manufacturing system. The mentioned PN-based approach was extended by Jörns et al. [20] in terms of *signal interpreted petri nets* (SIPNs) to design process automation systems. Therefore, an algorithm sequentially activates the PN transitions to detect unknown switching sequences and translates the sequences to LD or IL code. In contrast, Frey [8] proposed a PLC code generation approach by arranging code fragments which directly correspond to PN elements. Mušič et al. [27] applied real-time petri nets to generate control logic implementations.

Other approaches focus on code generation from *timed automata* (TA) as introduced by Wang et al. [34]. Therein, the edges of a TA are prioritized in order to overcome the non-determinism of TA and enabling automated translation of a TA into LD. Sacha [28] proposed a formalization for transformations of timed finite state machines into PLC control code. A similar approach is described by Flordal et al. [7], where deterministic finite automata are applied to generate executable interlocking policies implemented in PLC programming languages for industrial robot cells.

Beside approaches based on discrete event models, there exist several code generation methods and concepts which take the process information from preceding engineering steps into account [30, 32]. Schäfer et al. [29] proposed an integration framework in order to generate PLC code based on state-chart transformations. Integration of digital process information was described by Bergert et al. [2] to be a promising approach for code generation for controlling process plants. The required information for PLC code implementation beside the nominal sequence were analyzed by Güttel et al. [9]. Based on structural plant information exported in the CAEX format, Drath et al. [5] proposed a rule-based code generation approach.

Vogel-Heuser et al. [33] developed a framework for translation of models specified in *unified modeling language* (UML) into PLC code according to IEC 61131-3. An ontology-based approach to generate IEC 61499 [12] compliant code was proposed by Dai et al. [4].

2.3. Discussion

Most of the available literature dealing with PLC code generation addresses only some separated aspects of PLC programs as defined by Güttel et al. [9]. Furthermore, the disadvantage of the summarized approaches based on petri nets [3, 20, 27] and the automata-based approaches [28, 34] is, that the program has to be specified in an extra model which is afterward transformed into executable code. For

industrial applications it is necessary that the generation process fits smoothly into the existing workflow. This can be achieved by integration of knowledge from engineered artifacts and developing the PLC code based on these indirect code specifications.

3. Concept

In this section the three major steps of the proposed PLC code generation concept are described. First, the ontologies as well as their automated population methodologies are defined. The populated ontologies are queried by an rule-based reasoning algorithm in the second step as described in Sec. 3.3. Finally, the transformation of the query results into PLC code is stated.

3.1. Engineering artifacts

During an industrial process engineering project several engineering artifacts are designed. One of the basic documents are *piping and instrumentation diagrams* (P&IDs), which define the physical structure and the required process control equipment of the plant and thus link the process engineering with the electrical and automation domain. In general, a P&ID can be modeled as a quadruple $\mathcal{P} = \{P, I, F, C\}$, where P represents the set of pipes connecting the process equipment elements I like vessels or valves, represented by symbols defined in ISO 10628 [15]. The set F represents *process control system* (PCS) functions and C is the set of interconnections within control loops according to IEC 62424 as depicted in Fig. 2.

Based on the P&ID, the engineering artifacts defining the automation aspects of the plant are elaborated. The electrical plan specifies the process control instrumentation $I_p \subseteq I$, the signal routing from the physical device to the assigned PLC input or output, and contains additional information like vendor, equipment type, signal adaption or measurement range. The symbols used in electrical plans are defined in the standard EN 60617 [11].

Furthermore, interlocks and safety critical shutdowns of the plant are defined in logic diagrams according to [18]. The logic diagram can be seen as a detailed solution of cause-and-effect analysis like safety analysis function evaluation according to ISO 10418 [16]. It specifies the logical equipment connections with respect to safety aspects. Furthermore, recipes according to ISA-88 [19] define required process steps and equipment control during batch processes and conditions for triggering alarms or alerts during production are specified in alarm lists.

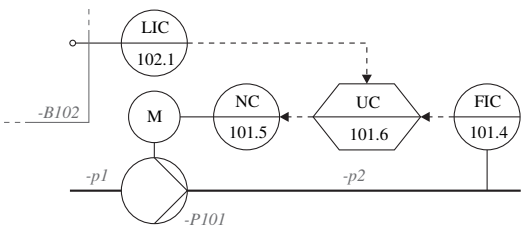

Figure 2. Process control loop according to IEC 62424.

3.2. Ontology-based data acquisition

In order to automatically access and process the design data and knowledge incorporated in engineering artifacts, a two-layered ontology structure similar to the *engineering knowledge base* (EKB) framework described by Moser et al. [26] is applied. Referring to Fig. 1, the first layer consists of domain-specific ontologies which model the standardized information and concepts of the specific engineering domain. For example, the ontology for representing P&IDs contains classes for equipment and control structures as well as properties like equipment type or measurement range. Since structural information and equipment properties can be exported in hierarchical and machine-readable formats like CAEX or other XML-based files from engineering tools, the ontologies can be automatically populated in terms of file parsing and instantiating classes, relations and properties based on the information contained in the exported data. The partial PandIX[1] specification for the pump and flow sensor as part of the P&ID shown in Fig. 2 can be defined as

```
<InternalElement Name="P101">
  <RoleRequirements RefBaseroleClassPath="PPE@PumpRequest">
</InternalElement>
<InternalElement Name="p2">
  <Attribute Name="DN"><Value>25</Value></Attribute>
  <RoleRequirements RefBaseRoleClassPath="PPE@PipeRequest">
</InternalElement>
<InternalElement Name="FIC101.4">
  <Attribute Name="FunctionCode"><Value>F</Value></Attribute>
  <Attribute Name="SignalCode"><Value>IC</Value></Attribute>
  <RoleRequirements RefBaseRoleClassPath="PCE@SensorRequest"/>
</InternalElement>
<InternalLink Name="p2_101.4" RefPartnerSideA="p2:101.4"
      RefPartnerSideB="101.4:P1"/>
```

Information contained in the PandIX file, representing P&ID information in structured formats, are then mapped to the instantiated classes within the specific P&ID ontology. The advantage of this approach is its independence of the plant structure which should be represented by the ontologies, since they model the concepts and relations defined in the domain-specific standards. Thus, the defined ontologies are applicable for a wide range of engineering projects, especially for process engineering. However, the mappings between the exported information and the concrete ontology classes or attributes have to be once defined manually.

Inter-ontology mapping. So far, the ontology modeling for representing information contained in the domain-specific engineering artifacts as well as their semi-automated population has been described. In order to integrate the data distributed over the engineering artifacts, their underlying tool data base and, as a consequence, over the several ontologies, common concepts are defined. Since signals represent the lowest conceptualization level, a mapping between the reference designation scheme according to IEC 81346 [14], which is used for labeling in electrical plans, and IEC 62424 applied to P&IDs is defined. The mapping transforms IEC 81346 designations,

[1] PandIX is a meta-model for modeling P&ID information [6].

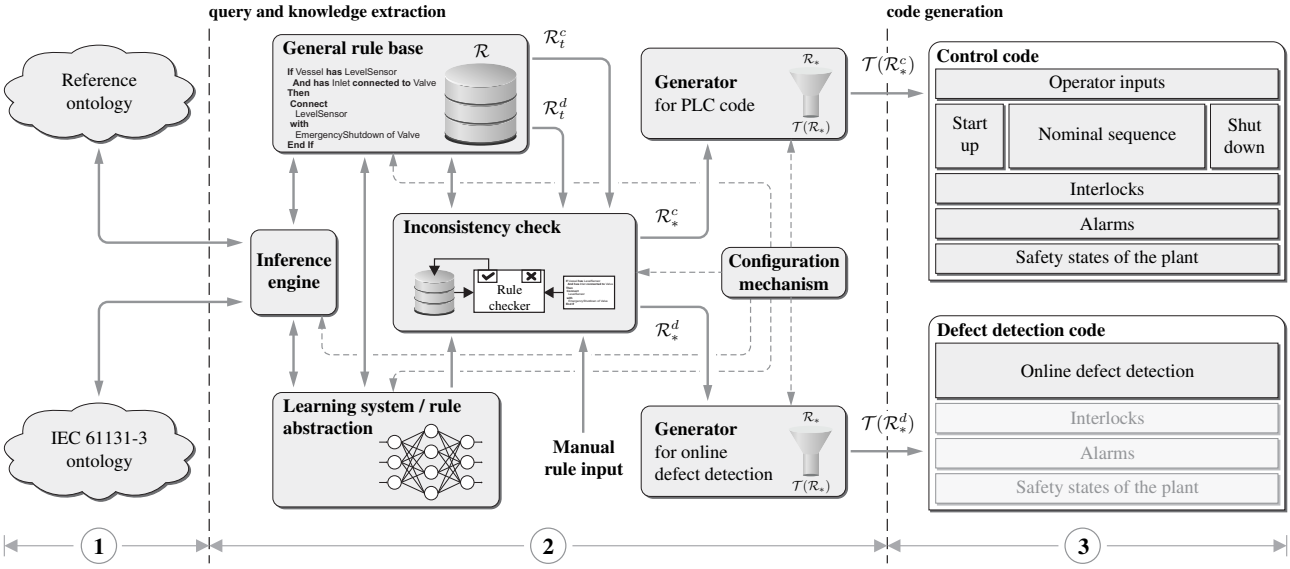

Figure 3. Scheme of the code generation framework: A rule-based reasoning algorithm queries the common ontology and provides the results to the code generator, which transforms query results into executable PLC code. The predefined rule base can be extended by rule abstraction from existing PLC code represented within an IEC 61131-3 ontology.

which, for example, define the locational and functional aspects of an object, and the *process control engineering* (PCE) identification label defined in IEC 62424. A complete PCE label defines the PCE category or function and also contains an unambiguous identifier of the object.

For example, consider a flow sensor which is connected to an analog PLC input. The IEC 81346 compliant designator of the hardware device specified in the electrical plan could be defined as `+A1+T2=B3`, where `+A1` and `+T2` are locational aspects and `=B3` is the functional aspect of the flow sensor. If the labeling within P&IDs is defined according to IEC 62424, the IEC 81346 aspect numbers could be directly mapped onto the unambiguous PCE identifier of the object. If further the mapping between the physical address, also contained in electrical plans, and the symbolic address of the PLC signal related to the flow sensor is assumed to be known via manually defined hardware configuration tables, one could define complete common concept mappings between all engineering artifacts and the domain-specific ontologies. The integrated view on the design data for the flow sensor, as it can be accessed via the common ontology, can be represented as the following CAEX specification

```
<InternalElement Name="FIC101.4">
  <Attribute Name="FunctionCode"><Value>F</Value></Attribute>
  <Attribute Name="SignalCode"><Value>IC</Value></Attribute>
      <Attribute Name="PhysicalAdd">
            <Value>IW301</Value></Attribute>
      <Attribute Name="SymbolicAdd">
            <Value>FlowAfterPump</Value></Attribute>
  <RoleRequirements RefBaseRoleClassPath="PCE@SensorRequest"/>
</InternalElement>
```

Data integration and knowledge inference via the reference ontology as a single-access point also enables possibilities for consistency analysis at design time or code verification methods. A simple code verification example with respect to the integrated design data could be to check the correct usage of sensor and actuator physical or symbolic addresses within software control loops.

3.3. Rule base population and maintenance

In the previous sections the data extraction from several engineering artifacts for representation in ontologies has been described. To query the ontologies and to generate the code for controlling a process plant it is necessary to define rules for the plant. Generally, these rules are stored within a rule base $\mathcal{R} = \{r_1, r_2, \ldots, r_n\}$ containing n rules, where each rule $r_i : \phi_i(x_i) \rightarrow \psi_i(y_i)$, $i \in \{1, \ldots, n\}$ consists of a condition function $\phi_i(\cdot)$ and an action function $\psi_i(\cdot)$, both defined in propositional or first order logic to represent the logical combination. The condition function takes a subset $x_i \subseteq I_p$ as its input and the action function acts on a subset $y_i \subseteq I_p$ of outputs. Further, we define the antecedent of r_i as $Ant(r_i) = \phi_i(x_i)$ and the consequent $Con(r_i) = \psi_i(y_i)$.

Referring to Fig. 3, the rule base $\mathcal{R}$ is applied for querying the ontologies and generate PLC code and thus $\mathcal{R}$ represents a composition of two different rule sub-bases. The sub-base $\mathcal{R}_q$ contains the rules r_k, $k \in \{1, \ldots, l\}$, $l \leq n$ for querying the ontologies and sub-base $\mathcal{R}_c$ contains the rules r_m for code generation, where $m \in \{1, \ldots, z\}$, $z \leq n$, $l + z = n$. The rules $r_k \in \mathcal{R}_q$ are applied by a rule-based inference algorithm in order to detect patterns in the ontologies. If a pattern is found which matches the rule r_k, an associated rule $r_r \in \mathcal{R}_c$ is selected and stored in a rule set $\mathcal{R}_t$ which is translated into executable control code for a PLC in the next step. As a consequence, we can define the consequent of a rule r_k as $Con(r_k) = r_r$.

Rule 1 close inlet of a vessel

```
1: if vessel has maximum level sensor AND vessel has
   inlet connected to valve then
2:   if maximum level sensor is active then
3:     close valve
4:   end if
5: end if
```

Rule 2 shut down pump

```
1: if vessel has minimum level sensor AND vessel has
   outlet connected to pump then
2:   if minimum level sensor is active then
3:     shut down pump
4:   end if
5: end if
```

Rule 3 shut down pump and close valve

```
1: if vessel has inlet connected to valve AND valve connected to pump AND vessel has maximum level sensor
   then
2:   if maximum level sensor is active then
3:     shut down pump
4:     wait(2 sec.){close valve}
5:   end if
6: end if
```

Figure 4. Exemplary rules combining inference rules and code generation rules (gray shaded).

Rule base population. The rules can either be defined manually or generated by a rule abstraction system. However, rule abstraction from existing PLC applications is not focus of this paper and we focus on manually defined rules. Three exemplary rule sets are depicted in Fig. 4, where each set contains a rule for ontology querying and a consecutive rule (gray shaded). If such rule sets are defined manually, they are split and stored in $\mathcal{R}_q$ and $\mathcal{R}_c$ due to several maintenance reasons. Since two rule sets can contain the same consecutive rule, it would increase the amount of storage space for the rule bases if they are not split. A further reason is the inconsistency checking within the rule bases.

Maintenance. To formalize possible inconsistencies of a rule base we consider two rules r_j and r_s defined as

$$r_j : \phi_j(x_j) \rightarrow \psi_j(y_j) \tag{1}$$

$$r_s : \phi_s(x_s) \rightarrow \psi_s(y_s)\ . \tag{2}$$

According to Khardon and Arias [21] and Ligeza [23], the rule r_j generally subsumes r_s, denoted by $r_j \preceq r_s$, if there exists a substitution ς such that $Ant(r_j)\varsigma \subseteq Ant(r_s)$ and $Con(r_s) \subseteq Con(r_j)\varsigma$. In this case the set of inputs x_j is a subset of x_s, and $y_s \subseteq y_j$. The strictly subsumption $r_j \prec r_s$ can be defined as $r_j \preceq r_s$, $r_s \npreceq r_j$, which excludes the equivalence of rules $r_j \equiv r_s$ if $\varsigma = \emptyset$. The rule equivalence or redundancy is defined as $r_j \preceq r_s$, $r_s \preceq r_j$. Conflicting rules is another possible inconsistency of a rule base. Formally, the rule r_j is contradictory to r_s if $Ant(r_j) = Ant(r_s)$ and

$$(\exists \gamma_j \in Con(r_j) \wedge \exists \gamma_s \in Con(r_s)) \rightarrow \gamma_j = \neg\gamma_s\ . \tag{3}$$

Furthermore, it is possible that rules with circular dependencies exist.

3.4. Code generation

The code generation is a transformation $\mathcal{T}(\mathcal{R}_t)$, transforming a rule set $\mathcal{R}_t \subseteq \mathcal{R}_c$ selected during the reasoning process to executable PLC code. One main aspect during the code generation process is the ordering and merging of the rules affecting the control code. Furthermore, it is necessary to check the selected set of rules $\mathcal{R}_t$ for inconsistencies like redundant, conflicting, subsumed or circular rules in order to keep the code correct and as small as possible.

The PLC code generation process is described in the following section by application to an exemplary control task.

4. Case study

The proposed conceptual approach is illustrated with a subsystem of a didactic plant in the Odo Struger Laboratory, Automation and Control Institute, Vienna University of Technology. Figure 5 depicts the P&ID, representing the physical structure of the subsystem. We consider a process scenario for mixing the two ingredients α_1 from vessel B102 and α_2 contained in B101, heating up the mixture and pumping it back to vessel B102. The related ISA-88 recipe is depicted in Fig. 6, representing the basis of the code generation for the desired control task.

Process description. Focusing on the reaction phase of the recipe, where α_1 and α_2 are already contained in the vessel B101, the first step consists of opening the valves YS101.5 and YS101.9. Since there is no feedback on the valve position, the step transition condition is defined by a timer, firing after two seconds. In the next step, the heater E104 and the pump control NC101.8 (full speed) are switched on. The heating and mixing phase is stopped if sensor TI101.2 indicates a temperature above 70°C. The heater and the pump are switched off and after two seconds the valve YS101.5 should be closed.

Code generation. The proposed PLC code generation approach transforms specifications from different

domain-specific engineering artifacts as well as manually defined rules into IEC 61131-3 compliant code and affects the control code functionalities as defined in [9] and depicted in Fig. 3, part ③. The result of the transformation process is a XML code specification according to the PLCopen schema, which can be imported to any supporting PLC programming tool.

The first generation step consists of transforming the steps and equipment controls defined in the recipe (cf. Fig. 6) into an IEC 61131-3 *sequential function chart* (SFC). This is achieved by inferring the recipe ontology, extracting and mapping the recipe structure to SFC steps and transitions. Within the recipe the reference designation according to IEC 62424 is applied. Since the mapping between the IEC 81346 and IEC 62424 reference designation schemes is defined as described in Sec. 3.1 (Part *Inter-ontology mapping*), the corresponding symbolic address for the signals specified in the recipe can be found via the signal routing within the electrical plan from the hardware device (IEC 81346 designation) to the PLC input or output device with corresponding physical address. Afterward, the physical address is translated to the symbolic address via the hardware configuration table, which is defined manually. Thus, the complete recipe sequence for controlling the reaction phase can be transformed into an SFC. The distinction between an set and reset output of the SFC is achieved by semantic mappings e.g. of the key words *switch on* or *close*. The resulting IEC 61131-3 SFC is depicted on the left side of Fig. 7.

After the transformation of the recipe control specifications, the safety related functionalities defined in the logic diagram are translated to PLC code. Therefore, the ontology containing the information of the logic diagram artifact is queried and the contained structure is directly transformed to control code, since the control specifications in the logic diagram are specified in terms of logical interconnections. The mapping between the reference designation

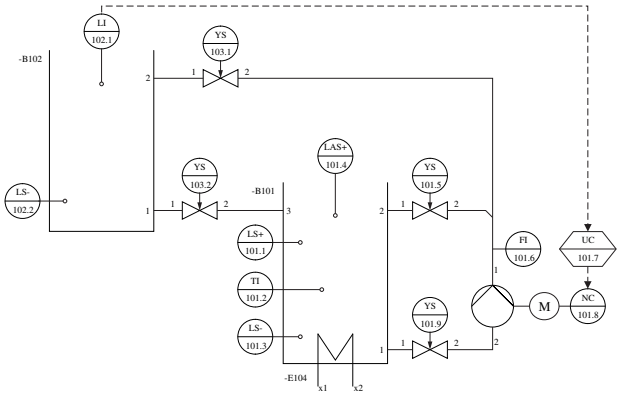

Figure 5. Piping and instrumentation diagram of the didactic process plant subsystem with additional control structure according to IEC 62424 (dashed lines).

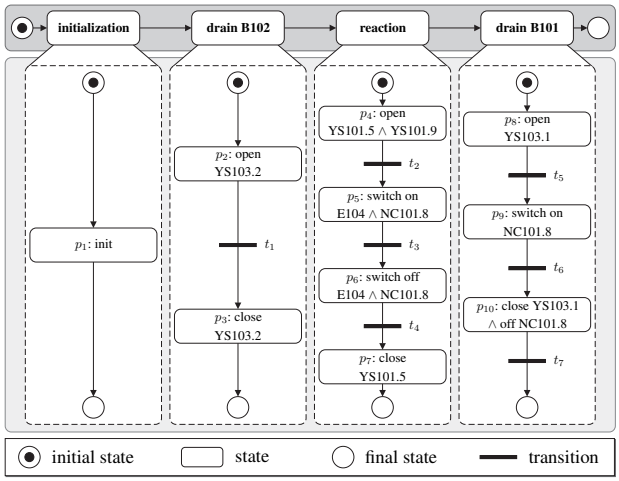

Figure 6. Process recipe for the exemplary control tasks, where the upper states represent the process operations (dark gray shaded).

to the symbolic address is achieved in the same manner as described for the recipe transformation. Furthermore, the manually defined rules have to be defined. For the didactic workstation depicted in Fig. 5 there exist three general rules as stated in Fig. 4. It can be seen, that Rule 3 subsumes Rule 1, which is detected by the consistency checking algorithm, and thus neglected for knowledge inference and the code generation process. First, the inference engine applies Rule 2 in order to search for matching patterns within the P&ID ontology. Since the outlet of vessel B101 is connected to a pump, the consequent of Rule 2 is activated and the contained code generation rule is selected and stored in $\mathcal{R}_t$. Furthermore, Rule 3 is also applied for querying the ontology and the consequent part is also stored in $\mathcal{R}_t$, since the condition part is also fulfilled. The general placeholders within the selected rules are instantiated with the concrete symbolic addresses before storing them in the rule base.

5. Conclusions & future work

In this paper, we described a conceptual approach for PLC code generation based on data extraction from engineering artifacts, which can be seen as informal software specifications. The information incorporated into several engineering artifacts is accessed via XML-based tool export files which are parsed and the information is represented within domain-specific ontologies associated with each artifact type. Furthermore, a common ontology was defined which realizes inter-ontology mappings, linking common engineering objects and makes the data accessible for a rule-based reasoning algorithm. The rule base as a base for the inference engine is composed out of reasoning rules. These reasoning rules further reference rules, which can be transformed by the code generator into PLC code.

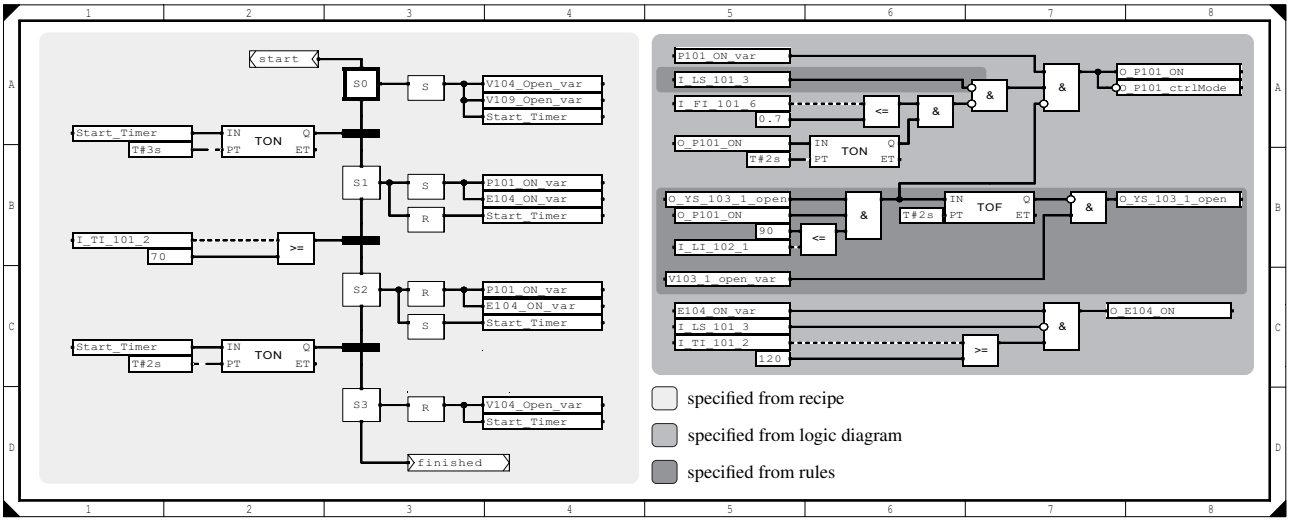

Figure 7. PLC Code resulting from specifications within recipe, logic diagram, and manually defined rule sets. The recipe phases are directly transformed to an IEC 61131-3-compliant sequential function chart (left side).

We further discussed the advantages of the proposed approach in terms of applicability to a wide range of control tasks. However, the approach is limited due to the manual input of rule sets, required for the code generation process. Furthermore, the specifications for the transformation into PLC code has to be adjusted which is a future research topic.

Acknowledgment

This work has been partially supported by the Christian Doppler Forschungsgesellschaft and the BMWFJ Austria.

References

[1] F. Anhäuser, H. Richert, and H. Temmen. Degussa PlantXML - integrierter Planungsprozess mit flexiblen Bausteinen [german]. *atp*, 46(10):62–72, 2004.

[2] M. Bergert, C. Diedrich, J. Kiefer, and T. Bär. Automated PLC software generation based on standardized digital process information. In *Proc. of the 12th IEEE Int. Conf. on Emerging Technologies and Factory Automation*, pages 352–359, 2007.

[3] G. Cutts and S. Rattigan. Using Petri Nets to develop programs for PLC systems. In *Application and Theory of Petri Nets 1992*, volume 616 of *Lecture Notes in Computer Science*, pages 368–372. Springer Berlin / Heidelberg, 1992.

[4] W. Dai, V. Dubinin, and V. Vyatkin. IEC 61499 ontology model for semantic analysis and code generation. In *Proc. of the 9th IEEE Int. Conf. on Industrial Informatics*, pages 597–602, 2011.

[5] R. Drath, A. Fay, and T. Schmidberger. Computer-aided design and implementation of interlock control code. In *Proc. of the IEEE Int. Conf. on Computer-Aided Control Systems Design*, pages 2653–2658, 2006.

[6] U. Epple, M. Remmel, and O. Drumm. Modellbasiertes Format für RI-Informationen [german]. *atp edition*, 53(1–2):14–26, 2011.

[7] H. Flordal, M. Fabian, K. Åkesson, and D. Spensieri. Automatic model generation and PLC-code implementation for interlocking policies in industrial robot cells. *Control Engineering Practice*, 15(11):1416–1426, 2007.

[8] G. Frey. Automatic implementation of Petri Net based control algorithms on PLC. In *Proc. of the American Control Conference*, volume 4, pages 2819–2823, 2000.

[9] K. Güttel, P. Weber, and A. Fay. Automatic generation of PLC code beyond the nominal sequence. In *Proc. of the 13th IEEE Int. Conf. on Emerging Technologies and Factory Automation*, pages 1277–1284, 2008.

[10] Int. Electrotechnical Commision. *IEC 61131: Programmable Logic Controllers*, 1993.

[11] Int. Electrotechnical Commision. *IEC 60617: Graphical Symbols for Diagrams*, 1996.

[12] Int. Electrotechnical Commision. *IEC 61499: Function Blocks*, 2005.

[13] Int. Electrotechnical Commision. *IEC 62424: Representation of process control engineering – Requests in P&I diagrams and data exchange between P&ID tools and PCE-CAE tools*, 2008.

[14] Int. Electrotechnical Commision. *IEC 81346: Industrial systems, installations and equipment and industrial products – Structuring principles and reference designation*, 2008.

[15] Int. Org. for Standardization. *ISO 10628: Flow diagrams for process plants - General rules*, 1997.

[16] Int. Org. for Standardization. *ISO 10418: Petroleum and natural gas industries - Offshore production installations - Basic surface process safety systems*, 2003.

[17] Int. Org. for Standardization. *ISO 15926: Industrial automation systems and integration – Integration of lifecycle data for process plants including oil and gas production facilities*, 2007.

[18] Int. Soc. of Automation. *ISA-S5.2: Binary Logic Diagrams for Process Operations*, 1992.

[19] Int. Soc. of Automation. *ISA-88: Batch Control*, 1995.

[20] C. Jörns, L. Litz, and S. Bergold. Automatische Erzeugung von SPS-Programmen auf der Basis von Petri-Netzen [german]. *atp*, 37(3):10–14, 1995.

[21] R. Khardon and M. Arias. The subsumption lattice and query learning. *Journal of Computer and System Sciences*, 72(1):72–94, 2006.

[22] Y. Kitamura. Roles of ontologies of engineering artifacts for design knowledge modeling. In *Proc. of the 5th Int. Seminar and Workshop Engineering Design in Integrated Product Development*, pages 59–69, 2006.

[23] A. Ligeza. *Logical Foundations for Rule-Based Systems*, volume 11 of *Studies in Computational Intelligence*. Springer, 2006.

[24] J. Morbach and W. Marquardt. Ontology-based integration and management of distributed design data. In *Collaborative and Distributed Chemical Engineering. From Understanding to Substantial Design Process Support*, volume 4970 of *Lecture Notes in Computer Science*, pages 647–655. Springer Berlin / Heidelberg, 2008.

[25] J. Morbach, A. Yang, and W. Marquardt. OntoCAPE - A large-scale ontology for chemical process engineering. *Engineering Applications of Artificial Intelligence*, 20(2):147–161, 2007.

[26] T. Moser, S. Biffl, W. Sunindyo, and D. Winkler. Integrating production automation expert knowledge across engineering stakeholder domains. In *Int. Conf. on Complex, Intelligent and Software Intensive Systems*, pages 352–359, 2010.

[27] G. Mušič, D. Gradišar, and D. Matko. IEC 61131-3 compliant control code generation from discrete event models. In *Proc. of the 13th Mediterranean Conf. on Control and Automation*, pages 346–351, 2005.

[28] K. Sacha. Automatic code generation for PLC controllers. In *Computer Safety, Reliability, and Security*, volume 3688 of *Lecture Notes in Computer Science*, pages 303–316. Springer Berlin / Heidelberg, 2005.

[29] W. Schäfer, R. Wagner, J. Gausemeier, and R. Eckes. An engineer's workstation to support integrated development of flexible production control systems. In *Integration of Software Specification Techniques for Applications in Engineering*, volume 3147 of *Lecture Notes in Computer Science*, pages 48–68. Springer Berlin / Heidelberg, 2004.

[30] D. Spath and U. Osmers. Virtual reality - An approach to improve the generation of fault free software for programmable logic controllers (PLC). In *Proc. of the 2nd IEEE Int. Conf. on Engineering of Complex Computer Systems*, pages 43–46, 1996.

[31] P. Szulman, M. Hefke, A. Trifu, M. Soto, D. Assmann, J. Doerr, and M. Eisenbarth. Using ontology-based reference models in digital production engineering integration. In *Proc. of the 16th IFAC World Congress*, pages 206–211, 2005.

[32] J. Thieme and H.-M. Hanisch. Model-based generation of modular PLC code using IEC61131 function blocks. In *Proc. of the IEEE Int. Symp. on Industrial Electronics*, volume 1, pages 199–204, 2002.

[33] B. Vogel-Heuser, D. Witsch, and U. Katzke. Automatic code generation from a UML model to IEC 61131-3 and system configuration tools. In *Int. Conf. on Control and Automation*, volume 2, pages 1034–1039, 2005.

[34] R. Wang, M. Gu, X. Song, and H. Wan. Formal specification and code generation of programable logic controllers. In *14th IEEE Int. Conf. on Engineering of Complex Computer Systems*, pages 102–109, 2009.

[35] A. Wiesner, J. Morbach, and W. Marquardt. Information integration in chemical process engineering based on semantic technologies. *Computers and Chemical Engineering*, 35(4):692–708, 2011.