Abstract – Re-engineering and reuse of programs implemented on Programmable Logic Controllers (PLCs) became vital in the recent years. This is because the programming of PLC programs is an exhaustive and expensive activity. This paper outlines a re-engineering approach based on the formalization of PLC programs. The approach utilizes XML as an intermediate step for the transformation of the existing PLC programs into a vendor independent format. After this transformation XML is used as a basis for the visualization and formalization of the PLC code. The formalization consists of two main steps: first the modular structure of the program is transferred to an UML Class Diagram and in the second step the algorithms contained in the modules are converted to State Diagrams. This conversion uses an abstraction method to avoid large automata.

I. INTRODUCTION

Machines and plants in the industry, however, also equipment automating buildings (elevators) and traffic control systems (traffic light devices) are automated mainly with the use of Programmable Logic Controllers (PLCs). A PLC is a computer which is suitable for the application in control tasks because of its special qualities concerning robustness and response time. PLCs are mostly programmed with vendor-specific assembly-like code languages. In Germany the assembler language Instruction List (IL) predominates especially under the tradenames of Siemens STEP5 or STEP7.

The control programming is performed by especially qualified technicians who seldom own knowledge of modern software technologies. Furthermore controllers are often reprogrammed during the operation of the plant to adapt them to new requirements. These two points lead to the fact that for practically no implemented controller a formal description exists. However, the formal description forms the basis for newer methods of control technology. These are initially systematic methods to check the functionality and safety of the controller, like dynamic simulation and formal verification.

The formal description also serves as a starting point for the automatic generation of run-time code for newer hardware (re-implementation). The requirements for the re-implementation extend beyond those of the already widespread reuse. Reuse is concerned with utilization of single functional units of the software (function modules) on the same platform, whereas re-implementation is aimed at the reuse of entire applications on a new platform.

The concept of re-implementation is associated closely with re-engineering. Following the taxonomy of Chikovsky and Cross [1] (cf. Fig. 1), the formalization of already implemented PLC code is a reverse engineering step (Design Recovery) while the re-implementation is a forward engineering step. The connection of a reverse and a forward engineering step over the same abstraction level is called re-engineering (renovation).

Reverse engineering means evaluating something to understand how it works in order to duplicate or enhance it. It allows the reuse of the know-how hidden in already implemented PLC programs. Reverse engineering is supposed to receive growing importance especially if existing hardware has to be replaced by new hardware with a different programming environment. This is expected to happen more often nowadays since automation suppliers move from special purpose hardware to PC technology with its fast innovation cycles.

The purpose of the presented work is to develop a new procedure for reverse engineering of control programs realized in STEP5 by deriving a formal representation and then an automatic implementation on new systems. This paper deals only with the first point (i.e. the reverse engineering). As a notation for the formal representation UML is used due to its widespread acceptance in the industry.

The rest of this paper is structured as follows. In the next Section some more information on PLCs and STEP5 is given. Section III presents the Software-Engineering (SE) methods and Internet technologies and their application in design of logic controllers. The re-engineering approach based on SE and Internet technologies is expressed in Section IV. This presentation is followed by an example in Section V. An outlook on further work concludes the paper.
II. PLCs And Step 5

The presented approach is not based on the standard form of IL proposed by the IEC [2] but on the vendor-specific STEP5 IL (used on the last generation of Siemens PLCs). This language was a quasi-standard in Germany for several years and the corresponding hardware is now no longer in production. This means that there is a good possibility for the application of re-engineering. The software on a STEP5 PLC is implemented in a hierarchical-like form using four main types of modules as follows:

OB: Organization Module, serves for the management of the user program in form of a listing of the program modules to be worked on.

PB: Program Module, in this module the user programs are located structured in groups. It includes only binary operations.

DB: Data Module, this module contains data, on which the user program works.

FB: Function Module, this is often used to realize frequently used or very complicated functions. The PLC operations in FBs are in a hybrid form i.e. digital and binary operations.

In addition to these general modules STEP5 also holds special types of modules like timers and counters.

III. Isolation and Visualization Using Oo-Paradigm And Internet Technologies

A. UML

Unified Modeling Language (UML) defines a notation that resulted from the unification of several formerly distinct object-oriented approaches and notations [3]. UML is made up of nine diagrams that can be used to model a system at different steps in the software development process. In the presented work two UML diagrams are used:

Class diagram: The class diagram is used to refine the use case diagram and define a detailed design of the system. The relationship or association between the classes can be either an "is-a" or "has-a" relationship. Each class in the class diagram may be capable of providing certain functionalities. These functionalities provided by the class are termed "methods" of the class. Apart from this, each class may have certain "attributes" that uniquely identify the class.

State Diagram: A State Diagram, as the name suggests, represents the different states that objects in the system undergo during their life cycle. Objects in the system change states in response to events. In addition to this, a State Diagram also captures the transition of the object's state from an initial state to a final state in response to events affecting the system.

An exhaustive summary of approaches using UML for the modeling of logic controllers is given in [4]. A survey dedicated to the use of UML for modeling automation domain applications is [5]. UML was used as the basis for the design of PLC programs in [5-9]. These works were dedicated to forward engineering, but their modular concepts additionally allow for easier reuse of the PLC algorithms.

UML is used in the scope of this work since it is an industry-standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems.

A UML Class Diagram is used to model the structure of the PLC program. The Class Diagram allows the analysis of the SW product through the exposition of the interrelations of the different components. It also allows the code generation for the described SW system since this aspect is provided by all UML tools.

UML State Diagrams are used to model the dynamic behavior of the modules. State Diagrams can be simulated by generating a code using available tools before implementing them on a run-time system. Some tools e.g. Artisan [10] allow also a check for consistency to give a report about correctness of State Diagrams.

B. Internet Technologies

Among the most popular Internet technologies used in the industry is XML (eXtensible Markup Language) which is a simple and flexible meta-language, i.e., a language for describing other languages. Other technologies are mainly related to XML like Extensible Stylesheet Language (XSL) is used for transforming XML into another format. An XML-Parser can transfer well-shaped XML documents in an abstract representation called Document Object Model (DOM) without using a grammar. XML Metadata Interchange (XMI) [11] is an international industry-standard defined by the Object Management Group (OMG). XMI 1.0 was developed in response to the OMG’s Request for Publication (RFP) for a stream format for the interchange of metadata including UML models. It is useful for transferring models from one step to the next in the design and coding progress or for transferring them from one design tool to another. Because XMI streams models into XML datasets, it also serves as a mapping from UML to XML. Software tools are available that allow to export and import UML models using XMI [12].

Internet technologies to model the design of PLCs were used in [13] where a formal model has been described for modeling a distributed Industrial Process Measurement and Control System (IPMCS).

XMI is used in this work as a basis for the generation of UML Class and State Diagrams.

IV. Re-Engineering Approach

The presented approach is an extension to previous work of the authors described in [14]. Fig. 2 illustrates the individual steps for the re-engineering of PLC programs. The approach is implemented in Java and it consists of the following four main steps (blocks in Fig. 2):

Step A: The PLC text is converted into a raw XML mapping its tabular form [14].

Step B: The modular structure of the PLC program is retrieved and exported as Class Diagram using XMI.

Step C: The raw XML is converted to an XML with instruction ID. Based on the added information about the instructions the assembler-like program is abstracted to a series of IF-THEN-ELSE statements.

Step D: Finite State Machines (FSMs) are derived from the IF-THEN-ELSE statements and written in an XML
format (this format is compatible with the format taken from the CodeProject [15]. Starting from the FSM in XML format XMI is used to generate UML State Diagrams.

The four steps are further explained in the following subsections.

**Fig. 2. Steps of the Re-Engineering of PLC Programs**

**A. Conversion of the PLC text into XML**

Given a PLC program in ASCII format and in a tabular structure with separate columns for addresses, labels, instructions, operands and descriptions delimited by white-spaces, XSLT can convert it into a well-formed XML document. The XML document obtained through this transformation is a hierarchically structured document.

The XML obtained as a result of the previous processing can be validated using a validating parser that confirms that the XML document in addition to being well-formed conforms to the set of syntactic rules defined in context of the PLC programming language (cf. Fig. 3).

**B. Separation of the modules with XMI**

Because of the advantages of using a standard and the availability of tools, it was proposed to extend the work presented in [14]. Based on the XML generated from the PLC code, the modules and the calling structure are extracted and translated to XMI. The XMI when read by any UML SW-tool generates UML classes of the Modules implemented in the PLC. In case the generated XMI corresponds to an XML of an OB module, an UML class will be generated for this module associated to the UML classes of the other modules called by it. This can be visualized in a Class Diagram.

**C. Abstraction**

XSLT is used to transform the well-formed and valid XML to another XML which as a result of identification on instructions has additional attributes appended to the instruction tags. This XML contains the Address, Label, Instruction, and Operand, together with the attributes of the instruction like the InstructionId, Type, Condition, and the Denotation.

The InstructionId attribute notifies whether the instruction is a valid instruction of the concerned instruction set. The type attribute classifies the operations into different types. Additional information is added to each original IL command during this step indicating for example whether the operation influences the accumulator content (“beeinfl. VKE” in German, cf. Fig. 9) or if its execution depends on the accumulator content.

The main idea in the abstraction step is to merge several instructions. As an illustration take the case of binary operations. In an assembler-like language like IL there are instructions that only influence the current state of the accumulator (not any other variables or output ports). However, the state of the accumulator is not of interest for the correct description of the algorithm. Therefore, operations that only influence the accumulator may be grouped until an operation arrives in the sequence that depends on the result of this calculation. The result of this abstraction is a sequence of IF-THEN-ELSE statements. As a very simple example take the IL code in Fig. 4: The operation U resembles a logical AND (and a load if it is the first operation in a sequence) . The operation S sets the operand to 1 if the accumulator content is one else it is not executed.

**D. Formalization**

The sequence of IF-THEN-ELSE statements generated in the abstraction step is converted to a FSM described again in
an XML format. Based on this XML an XMI document is built describing an according UML State Diagram. Basically, for each IF-THEN-ELSE statement a state of a Mealy automaton is generated the THEN and ELSE parts form the arcs in the automaton.

Details about the IF-THEN-ELSE abstraction algorithms as well as about the generation of the corresponding FSMs can be found for binary algorithms in [16], timers and counters in [17], and non-binary algorithms in [18].

V. EXAMPLE

The proposed approach is illustrated in the following example: A mixing machine for two colors (cf. Fig. 5) controlled by a Siemens Simatic S5 PLC programmed in STEP 5 Instruction List.

The control program consists of two modules: the Organization Module OB1 and a Program Module PB2. While OB1 only contains two lines of code (SPA PB2 to call PB2 and BE, the end statement of STEP5).

The Code (ASCII) of the OB1 is transformed to XML which is once more converted to XMI (cf. Fig. 6). This XMI generates the object structure of the PLC program. This structure can be imported by any UML tool (cf. Fig. 7).

The actual control algorithm is programmed in PB2 as shown in Fig. 8 containing 12 groups of instructions. The realized functionality is as follows: After actuation of the start button S0 the LED H0 glows, the valve Y1 opens and the pump M2 runs. If the liquid reaches the sensor B1, the valve Y1 closes and valve Y2 opens. If the liquid reaches the sensor B2, the valve Y2 closes, the pump M2 as well as the LED H0 is switched off and the stirrer M1 switches on. After six seconds the stirrer M1 is switched off. After the described process the system is in its initial state again. Through emergency stop S6, the thermal triggers F1 or F2 or by reaching limit switch B4, the plant can be switched off at any time.
The steps for the visualization of PB2 start from the XML with instruction identification (cf. Segment given in Fig. 9). From this XML the abstracted code in the form of IF-THEN-ELSE statements is generated (cf. Fig. 10). The transformation of the program code into a FSM results in an XML description as shown in Fig. 11. Finally, from the XML describing the FSM, an XML is generated describing the dynamics of the module as State Diagram in the UML syntax (cf. Fig. 12). An important point to be noted at the State Diagram is that the number of states equals the number of groups in the original IL code (Fig. 8) plus one initial state. Further analysis shows that actually each group was abstracted into an IF-THEN-ELSE statement and finally to a state with associated transitions in the State Diagram. Therefore, the State Diagram allows an easy understanding of the original algorithm. Of course this is only the case for well structured source programs.

Fig. 9. Segment of XML with Instruction ID

---

Fig. 10: Segment of the IF-THEN-ELSE Abstraction of PB2

---

Fig. 11. Segment of the FSM in XML format (States S0 and S1)

---

Fig. 12. State Diagram of the PB2
The approach presented above was also successfully applied to a didactic Modular Production System (MPS) from FESTO. The system handles 41 binary input and 32 binary output signals. It was originally controlled by a Siemens PLC programmed in STEP 5 IL. The control software to be formalized contains 21 modules adding up to a total of 1558 lines of IL code. These modules contain binary and non-binary instructions as well as timers and counters. This conversion is briefly described in [19].

VI. CONCLUSION AND OUTLOOK

Research on PLC programs is directed into three main areas: Verification and Validation, Design (Synthesis), and reuse and re-engineering. This paper depicts an approach in the third area. The modules of a PLC program are extracted with the help of XMI based on an XML document derived from the PLC. Furthermore, a formal description for the algorithms inside the PLC modules is presented. Here, the code is abstracted to IF-THEN-ELSE statements and once more altered to Finite State Machines (FSMs). These FSMs are converted to UML State Diagrams in an XMI format. Therefore, at the end the complete information on the original PLC program is contained in UML models and can then be used for further re-engineering steps. The example used in the presentation further shows that the generated models expose the original calling structure of the PLC program (Class Diagram) as well as the internal structure of the algorithms (State Diagrams). The models are therefore not only useful for further automatic treatment but also for understanding and documenting the original code.

Although the paper describes the re-engineering approach for the vendor-specific IL of STEP 5, the methodology is generic and could be easily adapted to other IL-dialects. The presented formalization using UML State Diagrams is limited to binary operations in the PLC program where the inputs are taken as external events for changing the state of the module (object in UML). In the intermediate step using FSMs also non-binary operations can be handled. Therefore, the next step in this work is an extension to transform the non-binary operations into activity diagrams through XMI.

VII. REFERENCES