Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 126 (2018) 527–536

Procedia Computer Science

www.elsevier.com/locate/procedia

22nd International Conference on Knowledge-Based and Intelligent Information & Engineering Systems

$ONTO\text{-}PLC$: An ontology-driven methodology for converting PLC industrial plants to IoT

Matteo Cristani[1], Florenc Demrozi[1], Claudio Tomazzoli[1]

[a]*Dipartimento di Informatica, Università di Verona, Strada Le Grazie 15, Italy*

Abstract

We present the new methodology $ONTO\text{-}PLC$ to deliver software programs on system-on-chip or single-board computers used to control industrial plants, as substitutes for programmable logic control technologies. The methodology is ontology-driven based on the abstract description of the plant at a level in which the plant itself is viewed as a set of *instruments*, each instrument being a set of *machineries* coordinated in functional terms by a control system, formed by sensors and actuators, under the control of an abstract model of behavior delivered by means of an *extended finite state machine*.

1. Introduction

Numerous industrial plants, in particular those that have been built to govern complex machineries that make use of sensors and actuators, employ Programmable Logic Controllers (PLC). In these plants, on board of the PLC, there are integrated programs, usually written in special-purpose languages such as LADDER, AWL, 1TOOL. On the base of the development of these software tokens are consolidated methodologies of software engineering, in particular general-purpose ones as the waterfall model, or, more recently developed methods for agile programming.

There are many evident drawbacks with the PLC approach. In particular, PLCs require *signal conditioning* and *function syncronization*, two processes that can relent in a heavy way the control of the plant itself.

Signal conditioning is necessary for the signal on remote General Purpose Input Output (GPIO) and is provided in Voltage range and Amperage range that can be different in the source sensor (or target actuator) connected to the GPIO, and in the GPIO entry in the PLC.

Function synchronization is needed, conversely, to guarantee that the source signal of sensors (or the target one, for actuators connected by GPIO) is read by the protocol in use in the system (typically ModBus, Carel or TCP/IP for standard IoT solutions) with a temporal clock especially dedicated to the reading process. Time for a GPIO to deliver a specific signal can directly depend on the source, or can be settled for the purpose of the communication in an intentionally directed way.

To reduce the impact of the above mentioned processes, it is often suggested in the literature [12] to substitute the PLC with especially designed architecture formed by Signal Conditioners, Temporal Synchronizers and Field

10.1016/j.procs.2018.07.287

Programmable Gate Array. Further on, following the development of several solutions in the market and in research, recommendations are given to provide solutions based on Systems on Chip (SoC), potentially also in Single Board Computers (SBC) [10].

Although a significant effort has been spent on these matters in the above mentioned studies, the real-life solutions and many solutions that have been documented in the current literature (for instance, for irrigation systems [3]) are completely designed *ex-novo* or re-invented from the existing machines by using a specific *ad-hoc* conversion process. These methods, although can lead to special-purpose solutions that result very efficient, are not valid for the generality of the plants, and require specific efforts. These efforts might be rather cost-ineffective and time consuming.

In this paper we propose a general methodology for the conversion of an industrial plant with PLC, into a plant that is driven by the above mentioned combination of technologies (Signal Conditioners, Function Synchronizers, SBC or SoC), where the original plant behaviour is emulated by means of software components that implement *Extended Finite State Machines*. The logical foundation of the solution is to host an ontology of behaviors of the machines driven by Sensors and Actuators, and a technique to convert an actual machine into a theoretical model of its behavior by means of Extended Finite State Machines. This methodology is based upon the notion of *ontology of behaviors*, namely a classification of the ways in which a configuration of input signals and output controls connect to each other, and is therefore named $\mathcal{ONTO}$-$\mathcal{PLC}$.

The rest of the paper is organised as follows. In Section 2 we report references to the current literature, and in Section 3 we describe the methodology core, in Section 4 we provide an example of application of the methodology, and Section 5 takes some conclusions and sketches further work.

2. Related work

Automation is a vast field of electronics, computer engineering and industrial engineering research. It covers several application issues including integration, security, and many software engineering topics that are important for the solution of everyday problems in the reference field. On one hand, it is quite common to explore methods for industrial engineering approaches, as well as this happen often also for industries that are concerned with electronics and informatics, but on the other hand radical top-level descriptions of software engineering techniques as related to actual industrial processes involving automation is rather uncommon. An exception to this limit in literature is a past study [2] where authors dealt with the practical issue of software deployment in industrial engineering.

Recently, some methodological issues have emerged in related fields, as in embedded systems [1] that has been used to inspire the approach presented here. On the other hand we have been also inspired by research problems related to sensor and actuators energy control, that we have developed in the recent past [7, 8, 6, 5, 13], but it is mainly based on the ontological methodology used for SCADA systems as described in a past study [4].

More generally, we should regard the problems related to the deployment of industrial engineering software deployment in automation as related to the abstraction level, where several different methods have been used. One possible approach is the one of SOA (Service Oriented Application), that has been widely studied also for industrial automation [11], and dealt with specifically from a methodological viewpoint more recently [9].

3. Methodology core

In this section we analyze the problem of developing a solution based on SoC or SBCs to control an industrial plant starting from the control of the same plant by PLCs. Generally speaking, we illustrate the methodology we have developed, called $\mathcal{ONTO}$-$\mathcal{PLC}$, within a framework described in five distinct analyses:

- the description of PLC systems;
- the description of the communication system;
- the analysis of medium/big size plants, as specific target of the approach we present here;
- and finally, the definition of the design procedure based on Extended Finite State Machine (EFSM).

Before to go into the above mentioned details of the structural aspects of a plant, let us introduce the core idea of the methodology itself. Basically, we refer to a plant as a set of *instruments*. An instrument is managed by a PLC in the

pre-update situation, and by an SBC (or a SoC) in the post-update situation. An instrument is, on turn, a set of devices, controlled by actuators and a set of sensors to gather data from the environment where the devices are embedded. A set of constraints to the input sensors is named an *input configuration*. Analogously, a set of constraints to the output actuators is named an *output configuration*.

We name *behavior* of an instrument the functional connection between one input configuration and one output configuration. From a general viewpoint, the input configurations, in real-life industrial applications, can be described by a taxonomy of behaviours of *values* and *gradients*. A general scheme, analogous to what has been done in Cristani et al. [4], regards the notion of *trend*. We consider, in particular, those input values that are returned by a sensor that measures data on a scale. We employ here the commonly known notions of *Time series* and *Temporal abstraction*.

On the one hand, the input configurations can be viewed as constraints to time series. The intuition tells us that there might be a peak in a certain time interval if the gradient of the signal is "high enough", in other words if the gradient value exceeds a precise threshold related to the human "peak" notion. We start from this observation to classify relevant signal behaviors on order to have a compressed representation of the signals expressed as temporal abstractions chains.

Particular attention must be payed to thresholds over the time-intervals size which is an important feature to classify Temporal Abstractions. We denote by $\tau_{\mu t}$ the maximal size for intervals that can belong to the *micro-trend* TA, by τ_{peak} the maximal size for intervals that can belong to the *peak* TA, by τ_t the minimal size for intervals that can belong to the *trend* TA, and finally by τ_{Mt} as the minimal size for intervals that can belong to the *mega-trend* TA. The thresholds describe the basic traits to be considered.

Basic TAs are event sequences while ***Complex TAs*** are episodes chains, in other words Basic TAs sequences. As the basic ones, Complex TAs can be defined with constraints over the size of time intervals r_I and angular coefficient m but they need even conditions over the concatenation rules. We employ in an obvious way the terms **Increasing**, **Decreasing**, **Stationary** and **Complex** relatively to gradient values.

On the other hand, the output configurations can be described into three basic groups:

- *Turn on/Turn off* binary controls over a given actuator;
- *Turn up/Turn down* with a parameter of size for a digital or analog actuator to change the value of an ambient value;
- *Setpoint fix* with a parameter of the value to which an actuator should be set.

Based on the above models of configurations, we define *specific structures of instrument models*, essentially schemes of the behaviors. This is the core notion used in the steps of the methodology, as presented below.

The typical PLC based system has two categories of components: **sensors** (device interacting directly with the system under measurement, it is commonly referred only to the component which converts the input physical quantity to a signal) and actuators (device by which an agent acts upon a system: the agent might be an artificial intelligence agent or any other autonomous being such as a human being).

Figure 1 summarizes the communication flow of sensors and actuators with a PLC device, and depicts the communication with a IoT technology as well.

The number of industrial plants governed by industrial automation systems has increased systematically in the past thirty years.

Although this evolution has affected positively the world of industrial automation in numerous practical respect, such as *software development flow* and quality of control, it is also true that the languages used to control sensors and actuators by PLC are obsolete in various senses, as the languages did not take much advantage of the evolution of other control systems. SBC, in particular the open source projects such as Arduino, and commercial ones, such as O-droid have provided room for a new frontier in Industrial Automation: the transition phase between PLC-driven technologies towards sensors and actuators completely controlled by high-level programming languages. This approach has also the strong advantage of providing the opportunity of implementing solutions that adapt to the progress line of *internet of things*, or more practically, to devices using the TCP/IP protocol.

Though this is certainly an interesting perspective, in the practice of Industrial Automation there are still resistances to the transition onto systems like the one devised above. The main reason for this is that the transition is financially disadvantageous in short-terms.

The Programmable Logic Controller (PLC) based architecture offers evident advantages. However, if we focus on the long-term aspects of the development of industrial plants, we notice a few major drawbacks.

Among the disadvantages of this architecture we can also count *rigidity*: PLC can operate for decades, so that in an installation, usually, there are PLCs with technology and functions which dates back to installation time, preventing the update to any new functionality the business might need. Moreover, PLC can be quite expensive, with cost growing easily to hundreds of Euro apiece.

Therefore, the goal is to define an architecture which might overcome these disadvantages and offer better maintainability and cheaper production and upgrade costs.

The IoT device guarantees several advantageous features, including, in particular, innovative ways to sense and deliver information from the physical world to the cloud.

When the methodology we are devising is implemented, developers deal with two possible scenarios: (a) a brand new installation or (b) an existing installation in which one or more device are due to be replaced.

In the first case, as we shall analyse more in detail at the end of this section, it is out of discussion, in a company that has already settled his competences to a level that is sufficient to provide IoT solutions, to go into an IoT solution. When there is a choice between updating an existing PLC-based system, without changing architecture and instead going to a SBC (or SoC) solution, that the methodology becomes relevant.

In fact, in current decision making procedures, possible choices are:

1. To preserve a fully-functional existing PLC plant with a PLC, without changes, risking obsolescence;
2. To update a faulty PLC plant towards a corrected PLC plant without changing the structure, risking obsolescence;
3. To update a faulty or fully-functional PLC plant to a SBC plant, re-building the software from scratch;
4. To update a faulty or fully-functional PLC plant to a SBC plant, porting the software with a porting standard methodology;
5. To update a faulty or fully-functional PLC plant to a SBC plant by $\mathcal{ONTO}$-$\mathcal{PLC}$.

At the end of this section, we provide an analysis of the impact of the choices above, by arguing that choice 1, 2 and 3 are risky, whilst 4 and 5 are both reasonable, but $\mathcal{ONTO}$-$\mathcal{PLC}$ guarantees more flexibility and reuse.

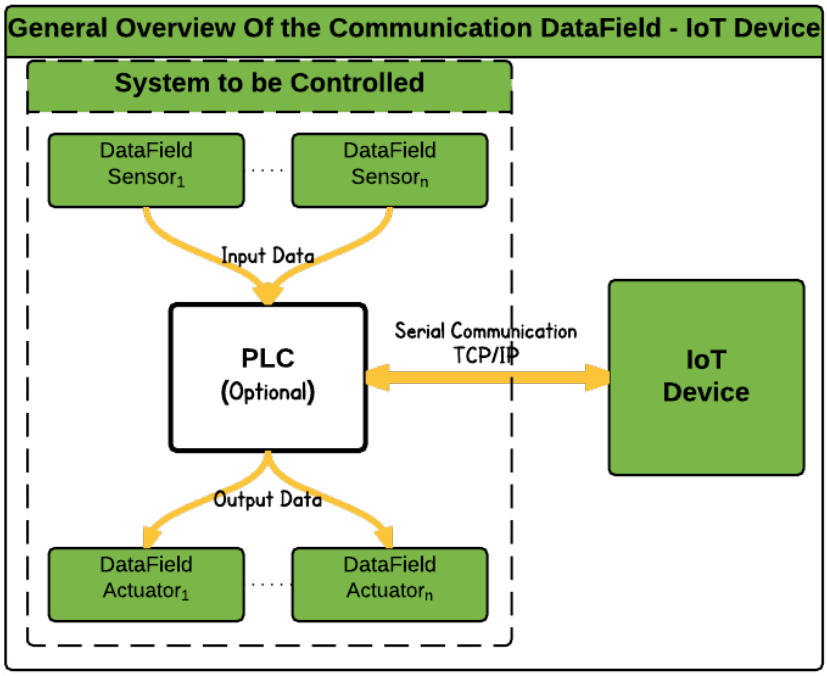

Fig. 1. General communication flow between a set of sensors and actuators and a intelligent Internet of Things (IoT) device.

We presuppose an IoT architecture that is structured into various levels, and is implemented, potentially, by means of Cloud Computing solutions. Basically, industrial plants that we presuppose underlying the transition of software, after the update are formed by one IoT master, potentially on Cloud, and a number of IoT slaves. Each slave controls an instrument, namely a set of devices connected to the ambient by sensors and controlled by actuators.

The majority of SBC and the whole set of SoC technologies do not control directly signal conditioning, and often assume synchronized digital signals on input GPIO. Therefore, in practice, PLC and IoT technology will coexist.

The IoT device acts also as a connection to the Internet

Clearly, we could imagine different configurations also depending on size of plants.

To provide an example that illustrates how $\mathcal{ONTO}$-$\mathcal{PLC}$ could act, we introduce a sample architecture, the architecture of a control and management system, in use in a winery plant.

Architectures can vary depending on the size of the installation itself, in relatively small plants we have:

- one IoT Device Master;
- no IoT Device Slave;

Growing with the size of the installation the need for more complex and reliable communication arises, so that we will find:

- one IoT Master
- one or more IoT Slave

Communication between Master and Slave IoT device is based on net protocol TCP/IP, limited to the address space of a private network so that all Slave devices are visible only to the Master. Master IoT device access the Internet so that it can transit data to the server and receive updates regarding several parameters of the private network containing all the IoT Slave Devices.

A typical IoT Slave device comprises these software modules:

- Input/Output Module
- EFSM module
- Internal DataBase

The Input/Output module takes care of the reading of all the data coming from all sensors as well as sending all output data toward the actuators.

Depending on the installation configuration, a PLC can be located between the set of sensors/actuators and IoT Device: in this case we have a communication between the Input/Output Module and the PLC, as described above, and the PLC itself takes care of the communication with the set of sensors/actuators.

The Extended Finite State Machine (EFSM) module, as the name suggests, represent the *extended finite state machine* model of the plant: it is developed on a case by case basis and strongly depends on functional requirements representing the behavior of the specific plant itself. The guide to this passage is presented at the end of this section, and is driven, in $\mathcal{ONTO}$-$\mathcal{PLC}$, by the time series ontology presented above.

In each IoT Slave device there will be an internal data storage with the responsibility of the local storage of input data coming from the field as well as the actual state of each actuator controlled by that IoT Slave device.

Moreover, within each IoT Slave device there will be two separate database, one responsible for all input/output data, the other to be used by the EFSM module which will take care of all the settings of the actuators in the installation. Parameters for these settings can be remotely changed by any user with the correct permission level.

All data contains in both database are stored in a central remote server at scheduled times and therefore the memory space, usually an issue in such devices, is cleared of these data.

The IoT Master Device comprise only one software module which has two responsibilities: receiving update parameters for IoT Slave devices and saving into the central server all daily measures coming from IoT Slave devices. Communication between Master and Slave is limited to a private network including only these devices, whereas Master and Central Server are connected through the Internet either directly or using a Virtual Private Network.

Step name	Step purpose
Phase 1: Devise of behaviors	Analyze the behaviors of the sensors and the actuators in the PLC plant, and classify the sets of input and output functional connections as temporal abstractions both on time series values of the sensor data or actuator control signals, and describe them in terms of temporal abstractions.
Phase 2: Similarity analysis	Similar behaviors are found within the PLC behavior abstraction performed in Step 1 and re-used, when present in the EFSM Knowledge Base.
Phase 3: Conversions to EFSM	Behaviors are converted into EFSM technologies by means of the scheme of behaviors devised in Step 1 and Step 2. It is also possible that in this phase, due to unacceptable complexity of the resulting EFSM, some of the behaviors provided in Phase 1 are actually dismissed and captured by means of distinct substructures of the EFSM per se.
Phase 4: EFSM implementation	The EFSM is implemented in high-level programming language, re-using features that have been abstracted in Phase 2 and compared to existing behaviors. The resulting model of behavior is simplified and the software re-use patterns are employed specifically.

Table 1. $\mathcal{ONTO}$-$\mathcal{PLC}$ methodology steps.

The behavior of the system is modeled and implemented in the EFSM software part of the IoT device.

The procedure begins when a set of *functional requirements* regarding the plant under control, usually defined by a skilled professional like an Electronic Engineer, has been correctly specified. This procedure maps the behaviors into the ontological scheme proposed in this paper.

Automatic plants are nowadays entirely governed by PLC, however significant benefits may be achieved with plants controlled by System on Chip devices (SoC) in which PLC, if present, are used only to condition signals and to synchronize GPIO; nevertheless a transition between the former and the latter type of plant is an expensive one. There are three ways to decrease the costs so far:

- Remain conservative, and this increases the risks of obsolescence;
- Convert to SBC, throwing the past solutions away, but this is not sustainable from an industrial point of view;
- Prudentially change both the plants and the internal competences towards a novel model of plant design.

$\mathcal{ONTO}$-$\mathcal{PLC}$ gives room to the third choice. The steps of the methodology are illustrated in Table 1
Once that the EFSM has been generated, depending on the programming language that is employed on the SBC or SoC that will control the machinery, we shall have a further step that brings to the code. This step is controlled by means of one specific Software Engineering methodology. The final result is a software program that controls the functions of the machinery. Notice that the signal is still conditioned by means of an external tool with respect to the SBC or SoC, that usually is a PLC, for the simple reason that it is cheaper than a specific devise especially designed for the purpose of conditioning the signal.

On the other hand, it is rather natural in every implementation of a system in which the machineries to control are more than one, that we aim at reducing the re-developing effort. Thus, general software engineering principles recommend to implement the core structure of an abstract EFSM, and specify the behavior of the EFSM itself in a separate source, for instance an initialization file, or a database.

4. A sample execution of the methodology

In this section we illustrate a use case, related to the management of a tank where wine is produced throughout a process of *controlled fermentation* of grapes and other viticultural products. The system comprise the following components:

- Wine Tank: contains viticultural product;
- Buffer Tank: contains cooling or heating liquid;
- Pump: use to pump cooling liquid in and out of the tank;
- Temperature Sensors:
  - one senses the temperature inside the main tank;
  - the other senses the temperature inside the buffer tank.

A simplified graphical version of the EFSM corresponding to the model of the management process of a tank where the wine is produced can be seen in figure 4, where only state transaction are shown for the sake of simplicity and readability.

A list of all possible states and related brief description can be found in figure 3 whereas the flowchart for the implementation of the EFSM is given in figure 2.

This section is dedicated to the representation of the functionality of the system under management and how it is implemented in an SBC architecture when approaching the porting by means of $\mathcal{ONTO}$-$\mathcal{PLC}$. The behavior of the system is modelled and implemented in the EFSM software part of the IoT device. The procedure used to obtain the extended finite state machine is explained in figure 2. The procedure begins when a set of *functional requirements* regarding the plant under control, usually defined by a skilled professional like an Electronic Engineer, has been correctly specified. This procedure maps the behaviors into the ontological scheme proposed in this paper.

A tank where the wine is produced throughout a process of controlled fermentation needs

- Tank with cooling liquid;
- Tank with viticultural product;
- Pump which send liquid cooling into the main tank;
- Temperature Sensor to monitor the temperature inside the tank with cooling liquid;
- Temperature Sensor to monitor the temperature inside the tank with viticultural product;
- Valve position sensor (0 moving, 1 open or closed).

This tank can be in one of these operation modes:

- Automatic warm;
- Automatic cool;
- Manual Open;
- Manual Close.

In *Automatic warm* or *Automatic cool* mode the EFSM takes the measures coming from the sensors and acts upon the valve, whereas in manual mode the valve is operated by a human: these modes are bypass modes to be used in emergency or when a special need arises. Consider the case in which the system is in *Automatic cool* and it starts in "off" position. We will have the following behavior:

- Wait a certain delay time T to allow initialization of the whole system;
- Check temperature sensors are operational *(values are within a certain range)* if so go to state **State_2** otherwise go to state **Error**;
- Check wether the Valve probe is *closed*, if so go to state **State_3** otherwise go to state **Error**;
- Check wether the temperature remains under **Desired Temperature**: if so then go back to state **State_3** otherwise go to state **State_4** ;

- Valve must go form *closed* to *open*: if so then remain in state **State_4** until **Temperature** ≥ **Desired Temperature**+ ; **hysteresis value**; if valve can not be *open* go to state **Error**, when **Temperature** < **Desired Temperature** go to state **State_5**;
- Valve positioning must go form *open* to *closed*: when done then go to state **State_2**.

5. Conclusions and further work

In this paper we discussed $\mathcal{ONTO}$-$\mathcal{PLC}$, a novel methodology for converting an automation plant managed by PLCs onto an updated one that is driven by single board computers or SoC. The core approach of the methodology is to abstract away from the implementation details by means of an ontology of the behaviours of the plant itself. Single parts of the plants, called instruments, are managed by the implementation of abstract models called extended finite state machines.

$\mathcal{ONTO}$-$\mathcal{PLC}$ has three specific aims:

- To facilitate the transition process;
- To prevent dispersion of knowledge;
- To determine a priori the transition time with reasonable accuracy.

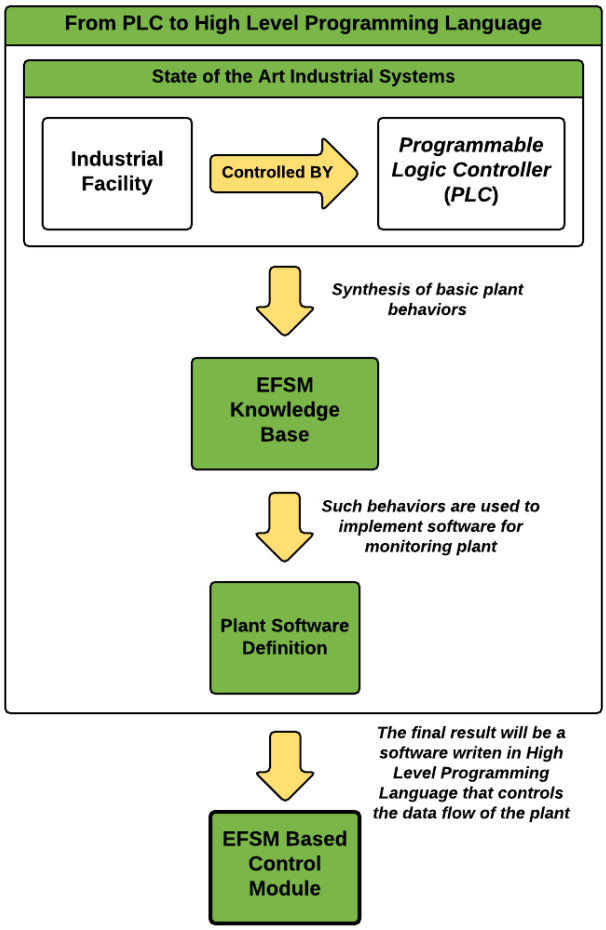

Fig. 2. Design procedure fro the EFSM module of the IoT device

State	Description
Start	System Initialization
State_1	Verify temperature sensor
State_2	Verify probe status
State_3	Verify temperature stability
State_4	Open Valve Probe - Enable Alarms
State_5	Close Valve Probe - Enable Alarms
Error	Probe malfunction

Fig. 3. EFSM states for a tank monitoring system.

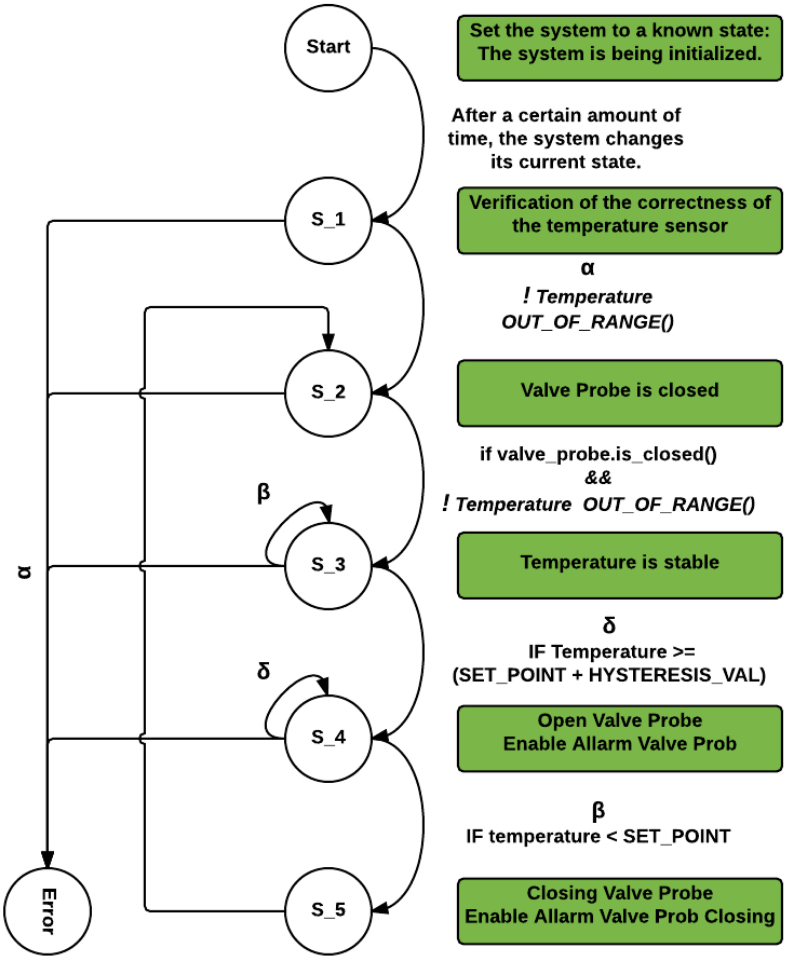

Fig. 4. EFSM and its state transactions

The idea of ontology-driven methodologies applied to industrial automation is not completely novel, as it has been applied to SCADA technologies [4], and to service-oriented architectures [11]. However, to date, there is no proposal for a methodology to be used for the deployment of the update to SBC of PLC managed plants.

There are several ways in which this research can be taken further. First of all, we plan to extend the methodology to those cases in which the plant is *similar* to an existing one, that however does not exist yet. This is a frequent case in industrial automation, as the behaviour of sensors and actuators are rather standard, and the real-life combinations of these behaviors are limited. The second aspect that is relevant is to introduce software engineering measures, in

particular estimates of development efforts (in terms of both person-months and absolute delivery time estimates). Finally we are providing a software tool for assisting developers in the tasks of the methodology.

Acknowledgments

Authors gratefully thank Sordato s.r.l. for financial support and for the provision of the experimental material employed in this investigation.

References

[1] N. Bombieri, F. Fummi, and G. Pravadelli. Automatic abstraction of rtl ips into equivalent tlm descriptions. *IEEE Transactions on Computers*, 60(12):1730–1743, 2011.
[2] I. Calvo, M. Marcos, D. Orive, and I. Sarachaga. A methodology based on distributed object-oriented technologies for providing remote access to industrial plants. *Control Engineering Practice*, 14(8):975–990, 2006.
[3] J.L. Chavez, F.J. Pierce, T.V. Elliott, and R.G. Evans. A remote irrigation monitoring and control system for continuous move systems. part a: Description and development. *Precision Agriculture*, 11(1):1–10, 2010.
[4] M. Cristani, E. Burato, and N. Gabrielli. Ontology-driven compression of temporal series: A case study in scada technologies. pages 734–738, 2008.
[5] M. Cristani, E. Karafili, and C. Tomazzoli. Energy saving by ambient intelligence techniques. pages 157–164, 2014.
[6] M. Cristani, E. Karafili, and C. Tomazzoli. Improving energy saving techniques by ambient intelligence scheduling. volume 2015-April, pages 324–331, 2015.
[7] M. Cristani, F. Olivieri, and C. Tomazzoli. Automatic synthesis of best practices for energy consumptions. pages 154–161, 2016.
[8] M. Cristani, C. Tomazzoli, E. Karafili, and F. Olivieri. Defeasible reasoning about electric consumptions. volume 2016-May, pages 885–892, 2016.
[9] T. Cucinotta, A. Mancina, G.F. Anastasi, G. Lipari, L. Mangeruca, R. Checcozzo, and F. Rusin. A real-time service-oriented architecture for industrial automation. *IEEE Transactions on Industrial Informatics*, 5(3):267–277, 2009.
[10] R. Giorgi. Scalable embedded systems: towards the convergence of high-performance and embedded computing. pages 148–153, 2015.
[11] F. Jammes and H. Smit. Service-oriented paradigms in industrial automation. *IEEE Transactions on Industrial Informatics*, 1(1):62–70, 2005.
[12] E. Monmasson and M.N. Cirstea. Fpga design methodology for industrial control systems - a review. *IEEE Transactions on Industrial Electronics*, 54(4):1824–1842, 2007.
[13] C. Tomazzoli, M. Cristani, E. Karafili, and F. Olivieri. Non-monotonic reasoning rules for energy efficiency. *Journal of Ambient Intelligence and Smart Environments*, 9(3):345–360, 2017.