AUTOMATYKA/ AUTOMATICS • 2013 • Vol. 17 • No. 2

http://dx.doi.org/10.7494/automat.2013.17.2.263

Igor Wojnicki*, Michał Rad*

Rapid Design and Development of Control Applications for Electrical Engineering

1. Introduction

Conducting a diagnostic process of electrical motors, and other electrical engineering tests, recording of variety electrical measurements and processing of collected data is needed. Presently, the most common way of recording data is to use a computer equipped with a data acquisition card. The card converts electrical signals to digital values which are processed in turn by the computer.

Data processing is often relatively complex – it is not a simple scaling, or arithmetic calculations [1]. A complex computer program which ensures data recording, processing (often real time processing) and result visualization is needed. Development of such applications is a significant part of industry oriented projects carried out in the Department of Electrical Machines AGH.

During production, fixing and even normal use of electric motors a wide range of tests are performed. The way of testing, and specific conditions during the tests are described by Polish Standards (Polska Norma) and other relevant documents (International Electrotechnical Commission – IEC standards in Europe). Procedures of the tests are composed in such a way, that they can be carried out by trained operators. Test automation is not an easy task though because it needs to take into consideration multiple measurements and states of the machine being tested.

The proposed method could be helpful in both cases which is developing a measurement application and test automation. For measurement applications – it simplifies the development process, allowing for component reuse regarding application logic, interfacing and visualization. For control applications – it is possible to validate them at the design stage and enforce other properties such as completeness.

Applicability of the proposed approach is not limited to the problems presented in this paper. The case showed below constitute a well established and real-world example.

* AGH University of Science and Technology, Faculty of Electrical Engineering, Automatics, Computer Science and Electronics, Krakow, Poland

2. Motivation and research context

Presently, to develop control or measurement applications, two programming approaches are used:

1. low-level methods, based on imperative or functional programming languages (e.g. C/C++, Ada, Erlang, ST, IL),
2. high-level methods, supported by a programming environment, generally utilizing a graphical representation of logic operations or data/control flows (e.g. LabView, Dasy-Lab, Matlab-Simulink).

Among the latter, languages aimed at creating description and implementation of control applications, backed by industrial standards, need to be identified, such as: SFC, FBD, LD. It should be noted that the SFC, FBD and LD are often supplemented by code written in lower-level languages such as ST or IL [2].

The low-level approach shows problems with verification and modification of the application. It is subject to well-known classical drawbacks of Software Engineering, which are two semantic gaps: requirement-design and design-implementation [3, 4]. This has a negative impact on the quality of the resulting software (differences between the implementation and design or requirements). It also reduces the possibility of its modification or adaptation to the new requirements.

The high-level approach generally limits the ability of clean exception handling. Furthermore, there are problems with the implementation of the control system response based on logical rules e.g. exception identification, operation mode change, reaction to particular stimuli etc.

3. Example

The No-load test of induction motor is chosen to show the advantages of the proposed solution. This test is intended to obtain both iron loss of stator and mechanical one. Measurements are taken at different voltages changing from 0.3 to 1.2 of the nominal voltage. Voltage, no-load current and no-load input are measured (U, $I0$ and $P0$ respectively).

Human readable instruction for the operator for this test is as follows. Set the voltage to the value ranging from 1.2 to 0.3 Un (nominal voltage) and at each step record the value of voltage, no-load current and power input. Writing this procedure in a high-level language using Matlab/Simulink is not straightforward. A sample of such an effort is given in Figure 1. The implementation is hard to check against errors and it is also hard to verify how it would respond to unusual, e.g. emergency, situations. This procedure could be written in a functional or procedural programming language, but it would not make it clearer.

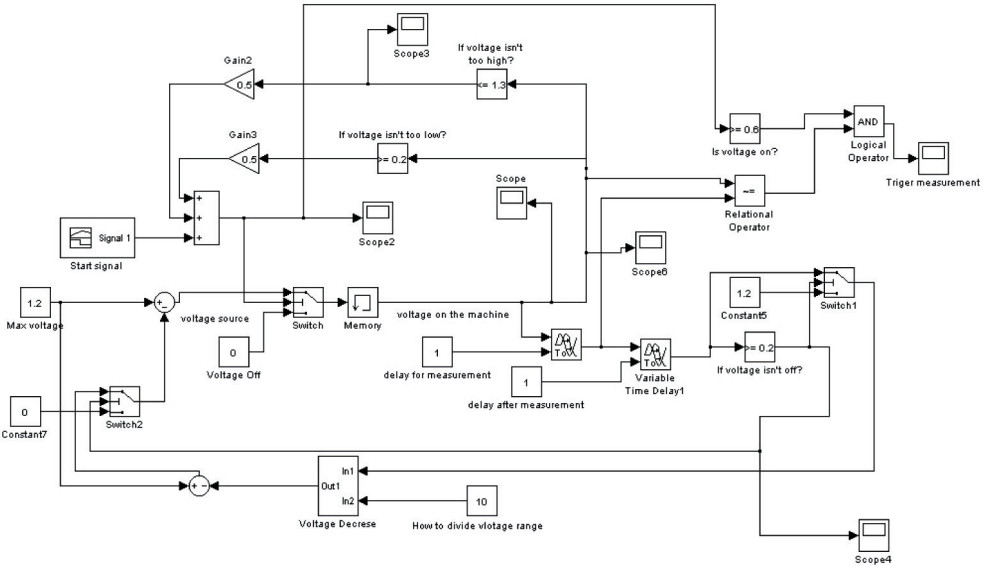

Fig. 1. No-load test, Matlab/Simulink program

4. Proposed solution of the problem

To give the possibility of making a control and measurement applications while eliminating the need for low-level programming of peripheral devices (sensors, actuators) a clear division into three layers is proposed:

- application logic,
- communication modules,
- mapping.

The proposed layers are inspired by the Model-View-Controller paradigm known from computer science. Its advantages for control purposes are presented in [5].

The application logic layer defines the behavior of a control and measurement system, describing the measurement process and interpretation of the results. The communication modules is a collection of ready-to-use software components for accessing peripheral devices (sensors, actuators) as well as the user interface (switches, buttons, graphical user interface).

The mapping layer defines which specific communication modules will be used for the particular application logic component.

Thus, while developing a control and measurement system, a potential developer--designer focuses on the design of application logic instead of fighting programing quirks. He also assigns appropriate communication modules, so the application can access and control physical devices, and interact with the operator.

Applying the proposed approach allows to focus on describing the measurement process, not on the programming itself.

A measurement process usually consists of several stages. Each stage consists of steps. Taking each step, and the transiting between the stages depends on values being measured, or operator response. Thus, the measurement process itself can be described as a series of natural language rules. A single rule consists of conditions and decisions. Conditions are expressed in terms of the values being measured, or operator responses, while decisions are setting control values, result calculations, informing the operator, or moving to another stage. Therefore formalizing rules by specifying them in a clear and unambiguous manner, is a natural way to define a control and measurement process.

```
IF u = [0] THEN uo = [1.2] * un p1m = [ ] im = [ ] n = [0] dpm = [ ] dpfe = [ ]
```

Fig. 2. Example rule

To express rule conditions and decisions attributive logic is used [6]. It is a simple and powerful method of describing world being modeled using attributes. An attribute is uniquely identified by its name. Each attribute represents information about the selected object of a modelled process, such as data from a particular sensor, information for the actuator, a switch state, a partial result of the calculation, etc. Attributes take values representing information. For each attribute there is a well-defined domain provided in terms of admissible values. This allows for subsequent verification of the system, as well as automatic detection of hardware anomalies.

A single rule is given in the following form:

```
IF (A1 o1 a1) I ... (An on an) THEN (B1 = b1) ... (Bp = bp)
```

where `A1...An` are attributes in the condition part, `o1...on` are logical operators, `a1...an` are values, `B1...Bp` are attributes in the decision part, `b1...bp` are values assigned to attributes.

An example rule in Figure 2 says that, if a value of attribute `u` is equal to zero, then the value of attribute u0 will be set as a result of 1:2*`un` (where `un` is also an attribute), attributes `p1m`, `im`, `dpm` and `dpfe` will be reset (assigned an empty value), while `n` will be set to 0. In the above rule only one logic operator is used (comparison). There are more operators provided which implement arithmetic relations (grater than, lower than, etc.), set operations (belongs to, is a subset of, etc.). There are also operators which provide expression evaluation (in the example a multiplication is used) performing arithmetical and set operations. Such operators can be used both in the condition or decision part of a rule.

To support the stages of a measurement process, rules are grouped into contexts. An active context and an active rule concepts are also introduced. Active rules are these that belong to the active context. While executing a measurement process only active rules are considered, which means that only for these rules their respective conditions are checked,

and as a result, if the conditions match – actions are performed. Since a measurement process can take several stages to complete, which match contexts, it is available to switch between them. The switching is also carried out by rules, which – in the decision part – contain information about the context to be activated or deactivated.

Contexts are divided into two categories: major contexts and minor contexts. At any time at most one major and one minor context can be active. Major and minor contexts form a hierarchy of contexts. Within the major contexts there could be rules activating other major or minor contexts. Within the minor contexts there could be rules activating only other minor contexts.

Switching among minor contexts implements performing subsequent measurement process stages. Major contexts can contain rules that handle emergency situations, such as an im mediate shutdown or unconditional jump to a specific stage (e.g. due to a malfunction or readout error). Having a hierarchy of contexts allows to decompose an entire control process into smaller functional parts. This allows to define even complex processes in a clear way.

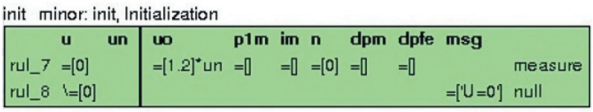

Fig. 3. Sample context with the rules

The use of a hierarchy of contexts is inspired, among others, by the Context-Based Reasoning [7] as well as Aspect-oriented Programing [8]. The proposed solution is a continuation of research in this area in the Department of Automatics AGH [9, 10, 11]. It is based on the concept of Contextual Networked Decision Tables (CoNDeT) [12].

To facilitate programing of the rules a visual notation is introduced. Rules, grouped by contexts, are represented in a way similar to the decision tables. This ensures easy identification of rules and contexts, simultaneously being visually clear.

A sample visualization of a single context is given in Figure 3. A description in the upper part consists of (from left) a unique context identifier (`init`), context category (minor), unique context name (init) and its description (`initialization`).

A table with rules is divided into two parts separated by a vertical line. The left part corresponds to the conditions of the rules, the right one corresponds to the decisions. The table's header identifies attributes being used, attribute names are in bold. Subsequent rows in the table represent rules of the given context. The first column represents unique identifiers of the rules. The last column, indicates a name of the context to activate while firing the rule, or `null` if the active context needs to be deactivated. For example the row representing the rule `rul_7` corresponds to its definition given in Figure 2. It should be noted that not all available attributes need to be used in a rule. For example, regarding rul_7 the un attribute is not used in the conditional part. Similarly, `msg` is used in the decision part of the `rul_8` rule but it is not used in `rul_7`.

If there is more than one context relationships among them are visualized in a form of a directed graph. An edge between a rule and a context indicate that the rule switches current context to the one the edge points to upon firing. An example visualization of four contexts is given in Figure 4.

The quoted example (Fig. 4) is a complete implementation of the control logic for the no-load test. Compared with the similar logic written using Matlab/Simulink (see Fig. 1) it can be noted that the proposed approach provides the following.

1. Measurement process stages are clearly indicated by contexts. Each stage corresponds to a separate minor context (`init`, `measure`, `calculate`).
2. The control logic is clearly given as rules. The rules represent conditions and actions to be performed, including context switching.

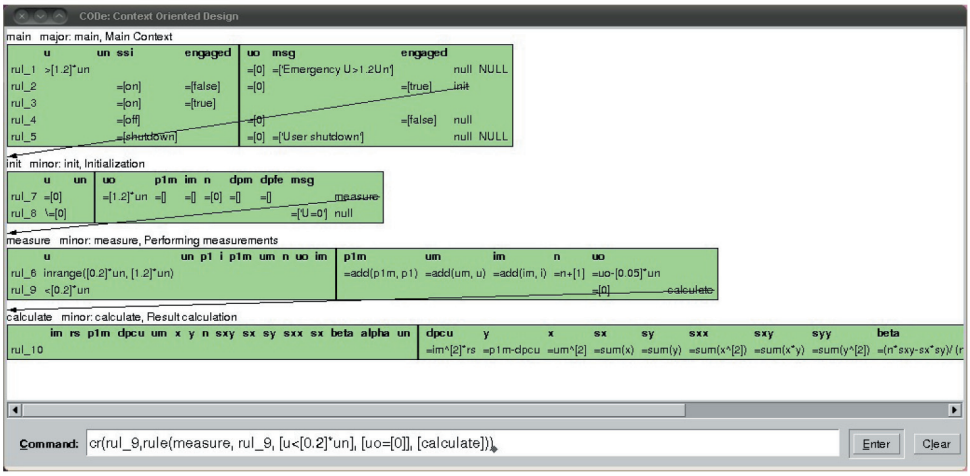

Fig. 4. Many contexts, no load test

It should also be noted that the proposed approach can easily handle emergency or unusual situations. It is achieved by utilizing the context hierarchy. Considering the example (Fig. 4), if at any stage of the measurement process, the emergency shutdown button is pressed, appropriate attributes are set to ensure proper shutdown procedure which is safe to the operator and the equipment. Pressing the button causes the attribute `ssi` to be set to `shutdown` therefore, the rule `rul_5` is fired (see Fig. 4). This sets the supply voltage to 0 (`uo = 0`) and informs the operator setting attribute `msg` to `User shutdown`. Simultaneously, it deactivates both minor and major contexts (`null NULL`)which stops the control process.

Similarly, if `u` is above the maximum allowed voltage (`rule rul_1`), the control process is safely stopped by setting the supply voltage to 0 and deactivating the contexts as well.

Communication modules provide both input and output for the control logic. These are ready-to-use software components, the use of which requires only a declaration and an appropriate assignment to the control logic. Hence the division of modules into two categories: output and input. Input modules provide data read from the peripherals (e.g. acquisition cards), which are connected to appropriate sensors in turn, as well as the operator interface built from physical controls (buttons, switches) or graphically visualized ones and more abstract data sources – such as files or data streams. Output modules provide data transfer to actuators as well as the operator interface: meters, graphs, data sinks (e.g. files).

Module assignments are purely declarative. A relationship between an attribute and specific module need to be stated. To make a proper module-attribute mapping all attributes need to be assigned to classes. There are the following two classes available: non-state, and state. The non-state class is subdivided into: input, output, input-output. Input attributes are used to read values from the input modules, output ones – to send data to output modules, and inputoutput attributes should be assigned to both the input modules and output modules, to be used for two-way communication. The state class' attributes have no modules assigned. They are used to express internal state of the control logic, not to communicate with the environment.

The assignment of modules is purely declarative. Moreover, it can be changed without having to modify the application logic if needed. This allows to use the same control logic to handle various physical devices or collect data from different acquisition cards. Similarly, if there is a need to modify the operator interface, it can be achieved without altering the control logic, by reassigning the modules. For example, displaying data as a graph can be easily switched to saving it to a file (for later analysis) by changing the assignment of the modules.

5. Summary

This paper presents a concept of computer system streamlining measurement processes for electrical engineering. Performing measurements in a classical way is based on either: the use of a high-level approach relying on existing, dedicated software, or implementing software in low-level programming languages. The high-level solutions often do not provide flexible mechanisms to handle unusual or emergency situations, which might be required. The low-level solutions require writing a program from scratch, which shifts the focus from the measurement process to the implementation issues.

The proposed solution provides both insight into the measurement process making it easy to define and refine, and relieves the user from having to write a low-level control software for handling sensors and actuators. Thus, the one who programs the measurement process focuses on defining it rather than the technical details related to the data acquisition.

The proposed solution is based on using rules. The rules represent the steps of the measurement process. They are grouped in contexts in order to be able to indicate

the measurement process stages. Such an approach is similar to the natural language description which the measurement process is usually defined by. The hierarchy of contexts makes it convenient to define the behavior of the system in case of an unusual or emergency situations. Communications with peripheral devices is provided by a set of communication modules.

The proposed solution is currently being tested. Most of the features regarding the rule editor, which is one of the most important components, have already been implemented. The ability to use communication modules, and a prototype of runtime environment have already been tested as well.

References

[1] Rad M., *Diagnostyka wirnika maszyn indukcyjnych z wykorzystaniem analizy falkowej i układów uczących się – induction motor cage diagnosis based on wavelet transform and self-training systems*. Przegląd Elektrotechniczny = Electrical Review, 2010, pp. 55–59.

[2] Mikulczyński T., *Automatyzacja procesów produkcyjnych. Metody modelowania procesów dyskretnych i programowania sterowników PLC*. WNT, 2006.

[3] Mellor S.J., Balcer M.J., *Executable UML: A Foundation for Model Driven Architecture*. Addison-Wesley Professional, 1st ed., 2002.

[4] Rash J.L., Hinchey M.G., Rouff Ch.A., Gracanin D., Erickson J., *A tool for requirements-based programming*. In Integrated Design and Process Technology, IDPT-2005. Society for Design and Process Science, 2005.

[5] Wojnicki I., *Separating i/o from application logic for rule-based control systems*. Decision Making in Manufacturing and Services, 2011.

[6] Ligęza A., *Logical Foundations for Rule-Based Systems*. Springer-Verlag, Berlin, Heidelberg, 2006.

[7] Gonzalez A.J., Stensrud B.S., Barrett G.C., *Formalizing contextbased reasoning: A modeling paradigm for representing tactical human behavior*. Int. J. Intell. Syst., 23(7), 2008, pp. 822–847.

[8] Kiczales G., *Aspect-oriented programming*. ACM Comput. Surv., 28(4es), 1996, pp. 154.

[9] Ligęza A., Wojnicki I., Nalepa G.J., *Tab-trees: a case tool for the design of extended tabular systems*. Proceedings of the 12th international conference, DEXA 2001, Database and expert systems applications, Munich, 2001.

[10] Ligęza A., Wojnicki I., Nalepa G.J., *A concept of cad/case tool for computer-aided design of rule-based systems*. Proceedings of the KKIO 2001, Otwock, October 2001.

[11] Ligęza A., Wojnicki I., Nalepa G.J., *Design and implementation support of rule-based systems* (in Polish). K. Zieliński (Ed.), II Polish National Conference on Software Engineering. AGH, Zakopane/Cracow, 2000, pp. 229–236.

[12] Wojnicki I., *From tabular trees to networked decision tables: an evolution of modularized knowledge-base representations*. Pomiary Automatyka Kontrola, 12, 2011.