MODELLING AND DESIGN OF PROCESS CONTROL SOFTWARE

Gregor Kandare

Department of Computer Automation and Control
Jozef Stefan Institute
Jamova 39, 1000 Ljubljana, Slovenia
gregor.kandare@ijs.si

Abstract: In process control software engineering, the software modelling techniques have not been commonly used. In the beginning of the nineties, the IEC 61131 standard for programming of programmable logic controllers was accepted. Introduction of the new modular languages opened the way for software engineering methods into process control software design. In the paper, a software modelling method using finite state machines will be presented. Finite state machines are a very convenient mechanism for describing the dynamic view of reactive systems. A prototype of a modelling tool that supports the modelling method and implements code and documentation generation is also shown.

Keywords: Software engineering, Process control software, Programmable logic controllers.

1. INTRODUCTION

Programmable logic controllers (PLC) are microcomputer devices used for sequential control. Originally these devices were designed to replace relay logic circuits and the basic programming language, ladder diagram, resembles relay logic schematics. PLCs are real-time controllers with cyclic behavior. Each cycle consists of three steps. The first step scans the inputs to the controller and maps a picture of the input status into the controller memory. After that a program stored in the controller memory is processed, taking into account the memory image of the inputs. As a result, an image of the outputs is produced. In the third step the image of the output variables is mapped to the actual outputs.

In the past, the PLCs were programmed using the languages such as ladder diagram and instruction list. In the beginning of the nineties, the International Engineering Consortium (IEC) 1131 standard for the languages for programmable logic controllers was accepted. The IEC standard recommends the structure for five programming languages, which includes ladder diagram, instruction list, function block diagram, sequential function charts and structured text (Lewis, 1998; Wratil, 1996). The IEC standard tries to unify the standard PLC languages with constructs similar to those in modern programming languages. The introduction of new modular PLC programming languages by the IEC 1131-3 standard opened the way for software engineering methods into process control software design. In modern computer software engineering, various analysis and design techniques have been

developed and successfully used for several years (Yourdon, 1989). The software models employed in these techniques allow the programmer to visualize the system before building it. In process control software engineering this is often not the case and software engineering methods and tools are not commonly used. One of the reasons for this is that the experts that usually develop PLC control software are not software engineers but mostly electrical and control engineers.

2. PROCEDURAL CONTROL SOFTWARE MODELLING

The development process for procedural control software is does not differ much from the process of development of other types of software. The main development stages are represented in Figure 1.

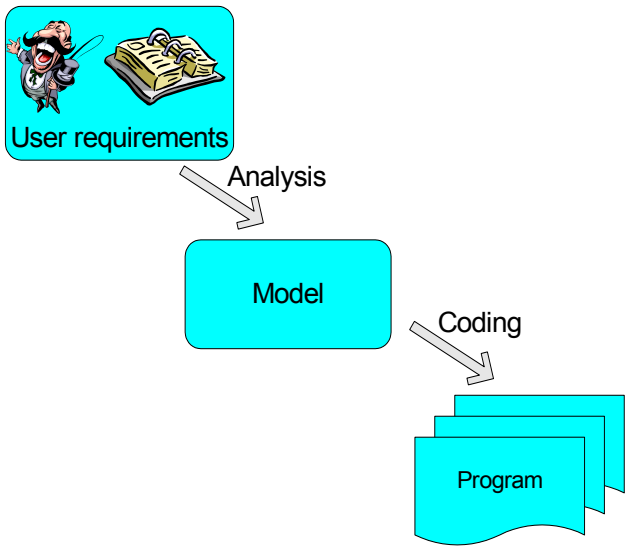

Figure 1: Software development process

In the first stage, the user defines the requirements of the system. The form in which the requirements are given depends on the user's experience. Usually, the requirements are presented in a natural language and they are often incomplete, ambiguous or even conflictive. The system analyst has to build a formal model of the system based on the user requirements. For this purpose, he has to communicate with the user in order to acquire all necessary information. The models are built for the following reasons:

- to focus on important system features while downplaying less important features.
- to discuss changes and corrections to the user's requirements at low cost and with minimal risk.
- to verify that we understand the user's environment and have documented it in such a way that systems designers and programmers can build the system.

With the help of the models, different perspectives of the system can be depicted. The most important views are:

- functional view,
- behavioural view,
- data view,
- communication view.

Which perspective is more relevant, depends on the application. In business software development, the data view is of most importance. On the other hand, when developing real-time software, behavioural (i.e. dynamic) view is pervasive.

The models can be graphical, textual or both. Graphical models contain higher information density, therefore they have more expressive power than textual models. Moreover, for human brain, it is easier to perceive two- or three-dimensional pictorial information than linear text.

In the third stage of the software development process, the programmer builds the program from the model. The process of mapping of model to code can be automated, if the model is formal, which means that it can be interpreted in an unambiguous way.

3. MODELING THE DYNAMIC BEHAVIOUR USING STATE TRANSITION DIAGRAMS

State transition diagram (STD) highlights the time-dependent behaviour of the system (Harel, 1987; Hatley, and Pirbhai, 1987; Booch *et al.*, 1999). Basically, the STDs consist of two types of elements: boxes, which represent states and arrows, which depict transitions between the states. Each state represents a period of time during which the system exhibits some observable behaviour. Arrows have transition conditions attached to them. If the condition is fulfilled, the transition fires and the system moves on to the following state, indicated with the transition arrow. An example of a simple state transition diagram with two states and one transition can be seen in Figure 2. If the *condition* is fulfilled, the system moves from state S1 to state S2.

Figure 2: Simple state transition diagram

The state transition diagram is the graphical part of the system model. The atomic actions performed in each state are described in the textual part of the model.

4. EXAMPLE PROCESS

The procedural control modelling technique will be shown on a concrete example of a part of an industrial grinding process shown in Figure 3.

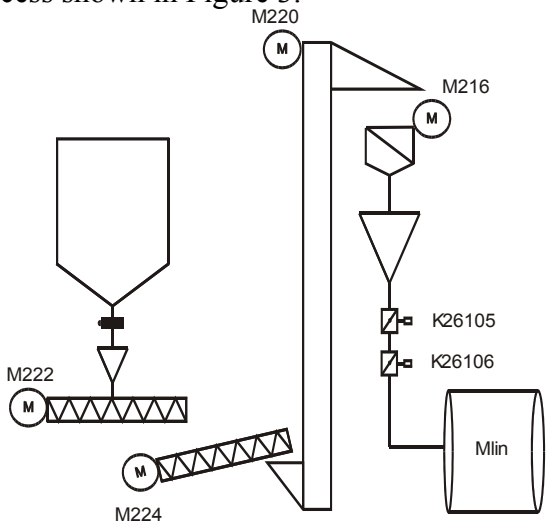

Figure 3: Dosing of the ore

From the storing silo, the ore is poured through a funnel onto a belt scale, where a frequency converter controls the mass flow of the ore. From the belt scale the ore is transported by a conveyor belt to the elevator and from there to a vibration sieve, where coarser particles and impurities are removed. The sieved ore falls through a funnel and a damper system into the

grinding mill. The damper system prevents excessive airflow entering the mill. In the rotating mill, the ore is ground by the grinding bodies. When starting the process, the damper system has to be started first, followed by the M216 motor, all other motors and finally the M222 motor. Stopping of the devices must be performed in the reverse order. Figure 4 shows the corresponding state transition diagram of the dosing control process.

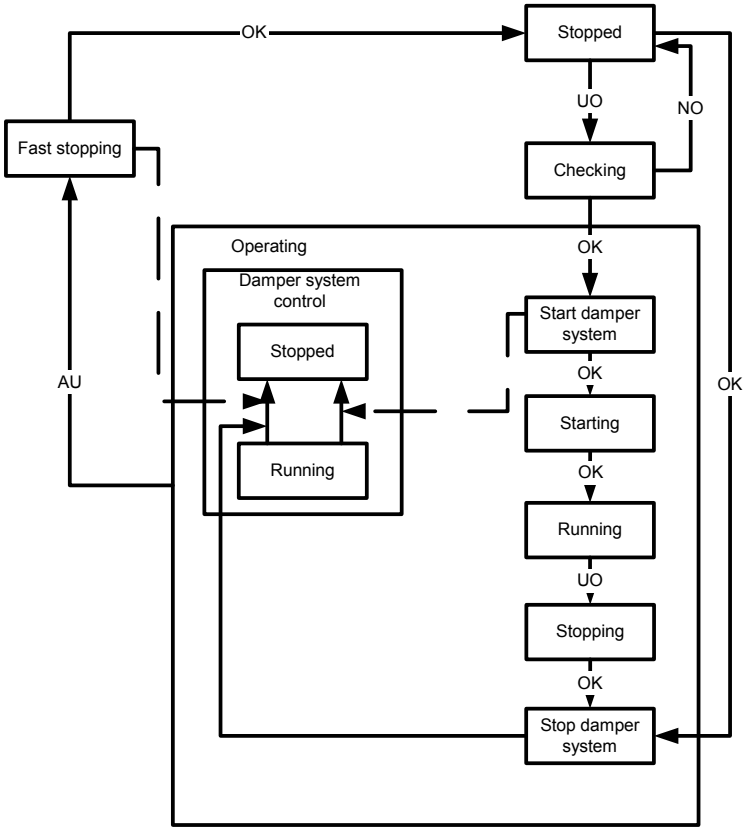

Figure 4: State transition diagram of the dosing control process

Textual model is complementary to the graphical model and describes the actions the control system performs in specific states. So, for example, the textual part of the model describing the actions in the Starting state would be as follows:

Starting:
Start M216,
Start M220,
Start M224,
Start M222.

5. GRAPHICAL MODELLING TOOL

In order to support the software modelling process using state transition diagrams, a modelling tool called Gecko is being developed. The modelling tool has a graphical user interface that enables the user to draw state transition diagrams using the appropriate graphical elements. The diagrams can then be automatically converted to a skeleton of function block language source code in the programming environment for Mitsubishi Melsec controllers. Each state in the diagram is mapped to a function block in the source code as shown in Figure 5.

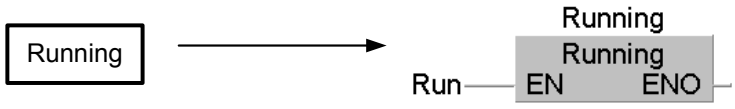

Figure 5: Mapping of the states to function blocks

The superstates, which enclose substates are mapped to function blocks that contain other function blocks. Furthermore, the modelling tool can convert the diagram to a documentation skeleton in a form of a Microsoft Word document. This is done by inserting (into the document) first the image of the state transition diagram followed by a heading hierarchy that corresponds to the hierarchy of the states in the state transition diagram.
The user interface of the modelling tool is shown in Figure 6.

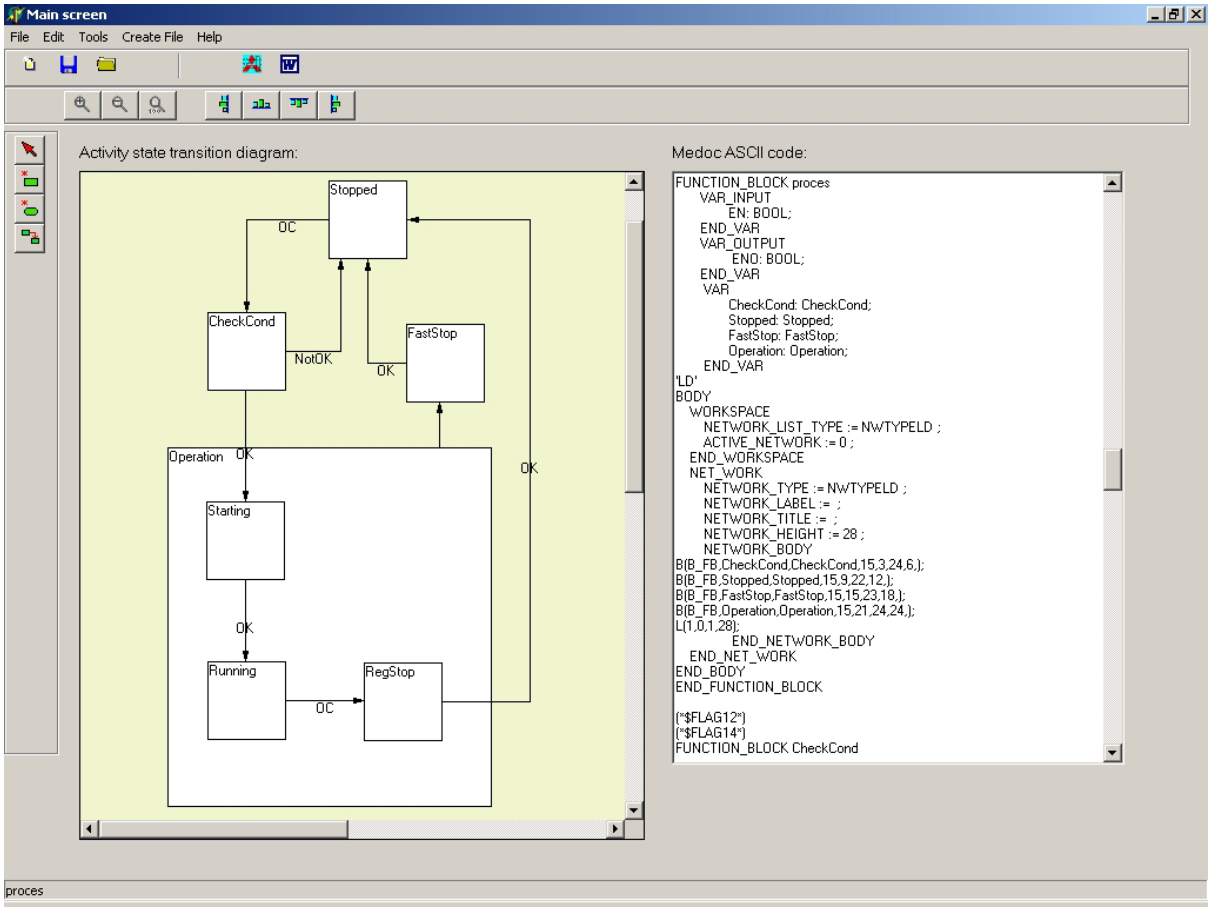

Figure 6: Gecko modelling tool

6. CONCLUSIONS

In the article, a systematic approach to process control software is presented. From the user's requirements, first a model is made in order to present them in a formal and unambiguous form. In the case of reactive software, like process control software, dynamics is the most important aspect. For this reason we use state transition diagram for modelling since this type diagram is very convenient for representation of reactive systems.
A prototype software package, which supports state transition diagram modelling, was made. The purpose of the tool is to create and edit graphical models and generate IEC 1131-3 code and documentation skeleton from the models.

REFERENCES

Booch, G., J. Rumbaugh, I. Jacobson (1999), *The Unified Modeling Language User Guide*, Addison Wesley, Boston.

Harel, D. (1987), *Statecharts: A Visual Formalism for Complex Charts*, Science of Computer Programming, North-Holland.

Hatley, D.I. and D. Pirbhai (1987), *Strategies for Real-Time System Specification*, Dorset House Publishing Co., New York.

Lewis, R.W. (1998), *Programming industrial control systems using IEC 1131-3*, The Institute of Electrical Engineers, London, UK.

Wratil, P. (1996), *Moderne Programmiertechnik für Automatisierungs-systeme – IEC 1131 verstehen und anwenden*, Vogel Verlag, Würzburg.

Yourdon, E. (1989), *Modern Structured Analysis*, Prentice-Hall, Englewood Cliffs.