Hyper-automaton: a Model of Control Algorithms

Vladimir E. Zyubin

Task Force “Linguistic Means of Applied Programming”, Institute of Automation and Electrometry SB RAS, prosp. acad. Koptyuga 1, 630090 Novosibirsk, Russia

zyubin@iae.nsk.su

keywords: hyper-automaton, control algorithms, modeling, event-driven polymorphism.

Abstract. The *hyper-automaton* is introduced as a model of complex control algorithms. The model reflects basic aspects of control algorithms: presence of an external environment, cyclic and event-driven functioning, synchronism, hierarchical structure and logical parallelism. The paper introduces basic definitions and conceptual means of hyper-automaton model, which encourage program implementation of the algorithms. A new kind of functional polymorphism we name *event-driven polymorphism* is distinguished and discussed.

1 Introduction

Present-day digital control system is the only way for industrial automation. The heart of such a system is programmable logic controller (PLC). PLC contains a microcomputer based control device, usually with multiple inputs and outputs, and software, which implements specific functions such as I/O control, logic, timing, counting, three mode (PID) control, communication and arithmetic. Software plays an ever-increasing role in industrial automation. With this, the associated software cost increase even to the point that it becomes the highest part of the total expenses.

PLC programming assumes formal languages and design patterns based on fundamental formalisms such as a control algorithm model, conceptual means and terminology, which reflect main aspects of control algorithms.

In the last thirty years, a variety of different formalisms for specifying control and reactive systems were introduced, like Hoare's model of Communicating Sequential Processes [1], Harel's Statecharts [2], Input / Output Automata [3], Esterel [4], Calculus of Communicating Systems [5], and their timed extensions [6, 7], to name a few. In general, each of them was introduced targeting a specific area of investigation. So, for the engineer it is not always obvious which model to select for a specific application domain. As a result the industry has chosen very disputable and heterogeneous means of the IEC 61131-3 [8] standard: relay logic language, functional block diagram, and sequential function chart. Experts criticize the standard for the low expressiveness of the languages, their serious limitations and yet mention the standard as an example of a misuse of standardization process [9]. So, a homogeneous and flexible model of control algorithm could drastically reduce expenses of PLC programming.

The idea of constructing a special formalism for reasoning about control algorithms is not new. A large amount of related work has been published over the years. Unfortunately, the efforts were dissipated over different fields: reactive systems, programming, logic, parallelism, homeostatic systems, so called real-time systems, robot-technique, etc., while the solution needs integral activities.

The author strongly believes that useful formalism for control algorithms can be designed on base of precise analysis of the target task only. The formalism has to reflect basic features of the field and must encourage a program implementation. In particular, the last assumes that conceptual means and terminology of the formalism should be compliant with the modern tendencies in education, namely, the total computerization.

In this paper we discuss hyper-automaton model, which is introduced as a useful formalism for control algorithms reasoning. In section 2 we discover basic features of control algorithms. In section 3 we give hyper-automaton model of control algorithm. Finally, in section 4 we sum up the results of the previous sections.

2 Basic Features of Control Algorithms

When we speak about a modern control we expect to see a digital *control system*, which includes feedback sensors and actuators. Sensors and actuators are connected to an external environment – so called *controlled object*, which implements an industrial process. The system acts on the controlled object by means of the actuators. The actions or, to be more precise, *reactions* of the control system are formed by control algorithm according to *events*, which are happened in controlled object. Events are continuously come in the control system from sensors. For digital control systems this circumstance causes *cyclic functioning* of control algorithm according to the following pattern: a) input of information about current state of controlled object by way of sensors, b) data processing and determination of required reaction, c) and data output, which changes current state of the actuators. *Event-driven* nature of control algorithm implies an algorithmic alteration of the program and a set of processed input / output signals depending on the events, i.e. control algorithms have a behavior, which cannot be defined by the knowledge of inputs only, but depends on the history of the events.

Control algorithm implies *synchronization* of its functioning with physical processes in the external environment. So, there is necessity of a time maintenance system and active manipulations with time entities such as time interval, time delay, and timeout.

The *mass logical parallelism* is another unique feature of control algorithms. It reflects existence of a large set of concurrent (or to be precise – independent and weakly connected) physical processes in the controlled objects. As the events appear in an arbitrary consequence, any attempt to describe the system reaction within a monolithic block leads to a combinatorial task with exponential increase of complexity, so called the combinatorial explosion of complexity [10]. Mass logical parallelism means that control algorithm consists of a large amount of independent and weakly connected parts, *logically isolated control flows*.

At last, any complex control algorithm has to have *hierarchical structure* that reflects artificial nature of the external environment, the designer plan that is implemented in form of the facilities [11]. Because of the logical parallelism, the hierarchical structure consists

of chains independently executed in parallel. It means that the *divergence* and *convergence* of control flow are a significant part of control algorithm. The algorithm structure can be arbitrary, irregular (not having strict parent-child relations between the chains) and, moreover, closed on itself.

Features the target model shell compliant with summarized in the following short list:

- presence of an external environment to interact with,
- cyclic and event-driven functioning,
- synchronism,
- mass logical parallelism,
- hierarchical structure.

3 Hyper-automaton Model of Control Algorithms

Cross-impact analysis shows that the target model can be designed on the basis of the finite state automata (FSA) theory. The FSA model has a set of attractive characteristics, which includes presence of external environment, event-driven behavior, and cyclic functioning, although they are not expressed in the model explicitly [12].

Other necessary features of the control algorithms cannot be expressed by means of FSA, namely: operations with time intervals, mass logical parallelism, hierarchical structure, divergence and convergence are not supported within the model.

Furthermore, the FSA model was designed for a hardware implementation. The cause was in the historical circumstances of the model creation [13], and negative effects become apparent even on conceptual level, e.g. in the terms “input alphabet” and “output alphabet”. The FSA model is extremely hard for studying, especially under the nowadays condition of totally computerized living environment and education. That leads to common misunderstanding and discourages the usage of this potentially very powerful concept [14].

We need modify the FSA model if we want the formalism will be understood and implemented in the software domain. To achieve the target model we introduce a concept of process.

3.1 Process

A *process* is a state machine where the states are functions. So we define process as a set of mutually exclusive functions, which are handled as a unified entity, or *polymorphic function*. Every time we call the process, it reduces to one of its function called *current function.* The current function provides the instructions to execute. One of possible instructions is 'to change the current function of specified process'. Also process has its own timer, which is zeroed when the process changes its current function. Mathematically, *i*-th process is a quintuple

$$p_i = <F_i, f_{1i}, fcur_i, T_i>, \text{ where} \quad (1)$$

- F_i is a set of mutually exclusive functions,
- f_{1i} is the *first function*, ($f_{1i} \in F_i$),
- $fcur_i$ is the *current function*, $fcur_i \in F_i$,
- T_i is the *current time*.

The F_i and f_{1i} elements characterize static features of the process. The $fcur_i$ and T_i elements give us means for reasoning about a process dynamics.

3.2 Functions, Events and Reactions

A function of a process is a set of events and reactions to the events. Mathematically, *j*-th functions of *i*-th process is a twain

$$f_{ji} = <X_{ji}, Y_{ji}>, \text{ where} \quad (2)$$

- X_{ji} is a set of events ($X_{ji} = \{x_{ji1}, x_{ji2}, \ldots, x_{jiL}\}$),
- Y_{ji} is a set of reactions ($Y_{ji} = \{y_{ji1}, y_{ji2}, \ldots, y_{jiL}\}$).

Any change (or an optional superposition of the changes) happened inside or outside of a hyper-automaton can be considered as an event: $fcur_i$ value of *i*-th process (*check-state* operator), value of a variable (regular conditional expression), T_i value (*timeout* operator).

An event is connected to a reaction it stimulates. The reaction can be an arbitrary superposition of operations (variable modification and $fcur_i$ value modification) which is determined according to *fcur* events.

Mutually disjoint sets of *passive functions* (Fp_i) and *active functions* (Fa_i) are distinguished among process functions ($f_{ji} \in F_i$).

A function f_{ji} is a passive function ($f_{ji} \in Fp_i$) if and only if $X_{ji} = \varnothing$.

The "normal halt" passive function (f^{NH}) and "error halt" passive function (f^{EH}) are *marked* passive functions. They are the same for all processes.

3.3 Hyper-automaton

A set of communicating processes forms a hyper-automaton, which reflects openness, event-driven nature, cyclicity, synchronism, and mass logical parallelism of a control algorithm.

A hyper-automaton is an ordered set of processes, which are cyclically stirred to activity with a period of activation T_H. A hyper-automaton is a triplet:

$$H = <T_H, P, p_1>, \text{ where} \qquad (3)$$

- T_H is a period of activation,
- P is a finite nonempty and ordered set of *processes* ($P = \{p_1, p_2, \dots, p_M\}$, where M is the number of processes,
- p_1 is the *first marked process*, $p_1 \in P$, which is the only non-passive when hyper-automaton starts.

Within the model we can reduce boundary conditions of the *perfect synchrony hypothesis* [7]. Instead of the original hypothesis, which states "neither computation nor communication takes time", we can use the following statement: the latency period for calculation overhead is less or equal to the period of activation T_H. The last is a less idealistic and quite reasonable condition for software implementation.

4 Conclusion

The aspects of the control algorithms are formulated. Namely:

- presence of an external environment to communicate with,
- cyclic and event-driven functioning,
- synchronism,
- mass logical parallelism,
- hierarchical structure.

As a new contribution, we propose the process concept in the form of modified FSA and the hyper-automaton model. Natural means of process concept allow create a complex hierarchical structure, which is composed of logically parallel and intercommunicated processes. The resulting model:

- reflects the main features of control algorithm,
- provides easy software implementation and gives conceptual means for control algorithms,
- considers the modern tendencies in education.

Also, we introduce a new kind of functional polymorphism, which has been called event-driven polymorphism.

References

1. Hoare, C. A. R.: Communicating Sequential Processes. Prentice-Hall Int. (1985)
2. Harel, D.: Statecharts: a Visual Formalism for Complex Systems. In: Science of Computer Programming 8. Elsevier Science Publishers B.V., North-Holland (1987) 231–274
3. Lynch, N., Tuttle, M.: An Introduction to Input/Output Automata. CWI Quarterly, 2 (1989) 219-246
4. Berry, G.: The Foundations of Esterel,.In: Plotkin, G., Stirling, C., and Tofte, M. (eds.) Proof, Language and Interaction: Essays in Honour of Robin Milner, MIT Press, Foundations of Computing Series (2000) 425–454
5. Milner, R.: Communication and Concurrency. Series in Computer Science. Prentice Hall (1989)
6. Kaynar, D. K., Lynch, N., Segala, R., Vaandrager, F.: Timed I/O Automata: A Mathematical Framework for Modeling and Analyzing Real-Time Systems. 24th IEEE International Real-Time Systems Symposium (RTSS'03), IEEE Computer Society Cancun, Mexico (2003), 166-177
7. Kof, L., Schätz, B.: Combining Aspects of Reactive Systems. In: Proc. of Andrei Ershov Fifth Int. Conf. Perspectives of System Informatics. Novosibirsk (2003) 239-243
8. IEC 61131-3. Programmable Controllers. Part 3: Programming Languages. 2nd edn. International Electrotechnic Commission (1998)
9. Crater, K. C.: When Technology Standards Become Counterproductive. Control Technology Corporation (1992)
10. Zyubin, V.E.: Multicore Processors and Programming. Open Systems J., №7-8 (2005) 12-19

11. Zyubin, V.E.: Text and Graphics: What Language Does Programmer Need? Open Systems J., №1 (2004) 54–58
12. Kuznetsov, B.P.: Psychology of the automaton-based programming. BYTE/Russia, №11 (2000) 22–29
13. Glushkov, V.M.: The Synthesis of Digital Automata. PhisMathGis, Moscow (1962)
14. Wagner, F., Wolstenholme, P.: Misunderstandings about State Machines. IEE J. Computing and Control Engineering, Vol. 16 (4), Aug (2004) 40–45