Implementation of state transition models in IEC 61499 and its use for recognition and selection of sequences of events and objects

Victor Dubinin
Penza State University,
Penza, Russian Federation
victor_n_dubinin@yahoo.com

Artem Voinov
Penza State University,
Penza, Russian Federation
voj49@yandex.ru

Ilya Senokosov
Penza State University,
Penza, Russian Federation
senokosov.i@yandex.ru

Valeriy Vyatkin
Luleå University of Technology, Sweden,
Aalto University, Helsinki, Finland
vyatkin@ieee.org

***Abstract*— Efficient application of model-based software design methodologies in industrial automation requires methods and tools for automatic code generation. Formal models can be especially useful to avoid ambiguity, to verify and evaluate performance, which ultimately will improve the quality and reliability of the project and lead to lower design costs. This paper proposes methods for implementing state-transition formal models, such as finite state and pushdown automata, as well as extended Petri nets (A-nets) by means of IEC 61499 function blocks. These implementation approaches can be used in the design of industrial cyber-physical systems for monitoring, diagnostics, conformance checking, detection and selection of specified sequences of events and parameterized objects from an input stream. One of the proposed applications is illustrated using an example of an assembly process with LEGO blocks.**

Keywords*— non-deterministic finite automata; pushdown automata; extended Petri nets; A-nets; conformance checking; selection system; LEGO; function block; IEC 61499

I. INTRODUCTION

Model-based design methodologies are becoming increasingly popular in industrial automation. For example, applications of model-driven engineering [1] and model integrated computing [2] have been reported. To make these developments practically usable, methods and tools for automatic generation of executable code from the models is required. A special class of models are formal models with mathematically rigorous semantics. Such models allow one to avoid ambiguity and uncertainty in the controller design, and to verify and evaluate its performance, which ultimately will improve the quality and reliability of the project and lead to lower design costs. In the field of industrial automation, formal models can be used in the design of control algorithms themselves, monitoring and diagnostics, supervisory control, conformance checking, detection and selection of specified sequences of events and objects from an input stream, etc. Sequences recognition is commonly used in compilers [3]. A similar task of selection can occur, for example, in assembling of some products [4]. Unlike the recognizers in language processing, a selection system can explicitly ignore input objects if they do not trigger corresponding changes in it.

In state transition models (STM) the functioning of a system or process is represented as sequences of transitions from one state to another. The choice of these models is explained not least by the requirements of reliability and safety, according to which explicit states should be determined in the system [5]. There are many kinds of STM such as finite automata (FA) [3,5], pushdown automata (PDA) [3], Petri nets (PN) [6], abstract state machines (ASM) [7], etc. In addition to the "pure" models, there exist their extensions and modifications. However, despite certain similarities, there are differences in the complexity and interpretation of transitions between states. This determines the need to develop particular methods for implementing the models of the main classes.

In our work we investigate implementation of STM of some important types by means of the IEC 61499 standard and its use for recognition and selection of sequences of events and objects. Currently, the importance of this standard is increasing due to the active shifting from centralized to distributed control systems. The IEC 61499 provides an architecture and component-oriented language for building distributed control systems in industrial automation [8].

The paper is structured as follows. Section II, Section III, and Section IV propose methods for implementing non-deterministic finite automata, deterministic pushdown automata, as well as selective A-nets (extended Petri nets) based on IEC 61499 function blocks (FBs), accordingly. In Section IV, a demo example of using A-nets for assembling LEGO constructions is considered too. Finally, conclusions and future work are attached in Section V.

II. IMPLEMENTATION OF FINITE AUTOMATA

Finite automata and their extensions are the most popular models used in industrial automation. There is a wide variety of models of this class. Despite this, the basic idea of the FB-based implementation of the FA-based models remains approximately the same.

In this section, an approach to the implementation of non-deterministic finite automata (NDFA) is considered. The peculiarities of the proposed approach are:

1) implicit determinization of NDFA in real time ("on the fly"). The essential point here is the synchronous nature of the functioning of NDFA.

2) a two-phase execution scheme in which at the first phase enabled transitions of the NDFA are fired, and at the second phase the new states of the automaton are published.

3) a token passing mechanism for the simulation of NDFA behavior.

978-1-7281-2927-3/19/$31.00 ©2019 IEEE

Within the framework of the proposed approach, it is possible to do its further detailed elaboration on the basis of the following classification criteria:

1) semantics of NDFA execution;

2) whether function blocks represent transitions or states of an automaton. In this paper, the state-based approach is considered, while a transition-based one has been considered in [4];

3) the manner of processing the input signals, as well as the organization of the second phase of execution (sequential vs. parallel).

For taking into account the third classification criteria, in the case when the state-based implementationn is adopted, the corresponding structural patterns are proposed. The main elements of the patterns are FBs that model the NDFA states (also called FB-states), and a FB-dispatcher, which organizes the overall computation process.

Implementation of a NDFA from Fig. 1 is represented in Fig. 2 in the form of a FB network and has been carried out in NxtStudio [9].

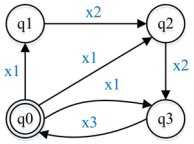

Fig. 1. Example of NDFA

Fig. 2 shows FB-states (*FB1-FB4*) and FB-dispatcher (*FB6*). Through event inputs $x_1..x_3$ one of input signals comes into the NDFA. In the parallel circuit, input signal x_i goes simultaneously to all relevant FBs. To synchronize the completion of the FB group execution, an acknowledgement mechanism with counting the number of receipts is used. At that, upon completion of processing the input signal x_i, which can include the change of a current state variable, each FB sends the corresponding receipt to the dispatcher. The dispatcher collects all receipts (their number is determined by the number of relevant FBs on the input signal x_i), and then initiates the second phase of execution.

In the second phase, all the FB-states publish in parallel their internal states to outputs (for other FB-states) and inform the dispatcher about this. When the dispatcher received acknowledgements from all the FB-states, signal *Out* is emited that indicates the end of processing the input signal.

III. Implementation of Pushdown Automata

Finite automata (FA) are fairly simple models and cannot describe complex processes, for example, represented by context-free (CF) languages. For recognition of CF-languages, pushdown automata (PDA) are used. The formal definition of PDA can be found, for example, in [3].

Below a method for implementing deterministic PDA based on IEC 61499 FBs is considered. As a baseline description, the method uses the graphic representation of PDA proposed in [3]. In this case, a transition between two states of an automaton has a label *<A, B, C>* consisting of three components: *A* is an input symbol; *B* is a symbol of the top of the stack; *C* is a symbol (or a sequence of symbols) which the symbol of the top of the stack will be replaced to. It should be noted that the stack itself does not appear explicitly in this representation.

Summary of the technique for transforming a PDA to a FB implementation is as follows:

1) each PDA transition is mapped to FB. The advantage of this approach is that almost all transitions are modelled by FBs of the same type, differing only in some parameters used during initialization. An alternative approach is the approach when states are represented as FBs. In this case, each FB will differ in the number of input/output signals and variables, and the complexity of the implementation of each FB-state will be larger.

2) tokens are used to mark transitions enabled by states. A token is a dynamic object that can be transferred from one FB-transition to another FB-transition. If a FB-transition has a token, then it can accept input symbols.

3) to represent the stack, a separate FB is used, which implements the operations of pushing and popping an element into the stack, as well as comparing the top of the stack with the specified value.

4) PDA states are not explicitly represented in the FB-implementation.

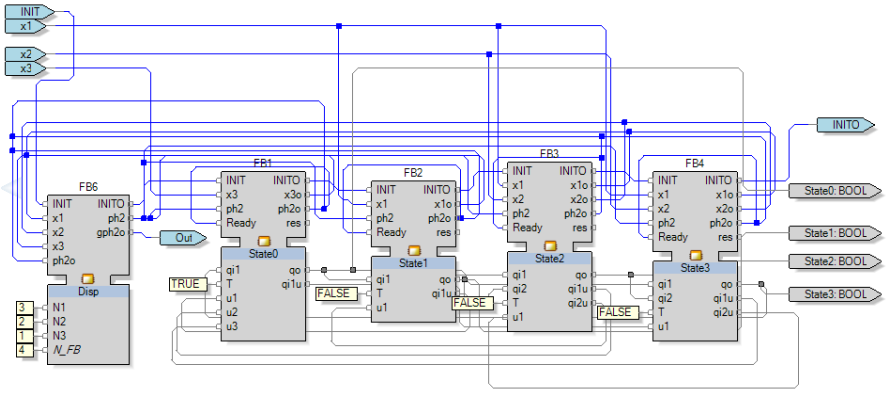

Fig. 2. FB-based implementation of NDFA from Fig. 1 in NxtStudio.

IV. Implementation of Petri Nets

Compared to automata-based models, models based on Petri nets are more powerful. When one considers the use of Petri nets for the recognition of situations or patterns, the works [10-12] could be mentioned. In [10], Petri nets are used to construct a query to the surveillance video (in terms of events) and further to recognize this specified pattern. The work [11] provided coloured Petri nets to model the recognition of chronicles expressed with logical and temporal operators, as well as minimum and maximum time delays. In [12], coloured Petri nets are used to determine whether an operator (for example, a pilot) correctly implements the corresponding guide. In [13], for the implementation of conformance checking, the authors proposed Petri nets with Data to model data variables, guards, and read/write actions.

In this work we apply extended Petri nets for recognition and selection of sequences of parameterized objects in the form $<a_1, a_2, ..., a_n>$ from an input stream. The A-nets, originally used in [14] for asynchronous modelling of net condition/event systems, were chosen as a basis. They are an extension of Petri nets in the direction of increasing the modelling and expressive capabilities due to the labelling of arcs and the complexity of enabling and firing rules that makes it possible to effectively process the integer variables. In addition, A-nets can be easily extended with control symbols and even actions attached to transitions and/or places that moves them to a class of transducers.

A selective A-net is defined as follows:

$(P, T, X, Y, Z, U, WX, WY, WZ, Q, A, G, m_0, TF)$,

where P is a set of places; T is a set of transitions; $X \subseteq P \times T$ is a set of input arcs of transitions with a minimum threshold (arcs with a check on "greater"); $Y \subseteq T \times P$ is a set of output transitions arcs; $Z \subseteq P \times T$ is a set of input arcs of transitions with a maximum threshold (arcs with a check on "less"); $U \subseteq X \times Y$ is a conjugacy relation of input and output arcs; $W_X: X \to N_0 \times \{N_0 \cup all\}$ is a function of X arcs weights; $W_Y: Y \to \{N^+ \cup *\} \times \{add, reset\}$ is a function of Y arcs weights; $W_Z: Z \to N^+ \times \{N_0 \cup all\}$ is a function of Z arcs weights; Q: $T \to N_0$ is a transition priority function. The following restriction must hold: $\forall t_i, t_j \in T\,[Q(t_i) \neq Q(t_j)]$. It makes the functioning of A-nets deterministic; A is a set of parameters (attributes) of an object; G is a set of transition guard functions; m_0 is an initial marking; $T_F \subseteq T$ is a set of final transitions. The firing of any final transition indicates the end of an objects sequence selection.

A transition is enabled if the guard condition is true and there is a token concession. It should be noted that combinations of arc weights form certain arc patterns that have their own semantics — a numerical arc, an unconditional and threshold reset arc, a testing arc with a "greater than or equal" checking, an inhibitory arc (arc with a "less than" checking), conjugate arcs for unconditional copying/adding/moving tokens.

For the integration and embedding selective A-nets to a control system based on the IEC 61499 standard, a method for their transformation to FB systems is proposed below. The main idea is to implement elements of A-nets (places and transitions) in the form of separate FBs. At that, a FB-place stores the marking of the corresponding place, and also performs on it the elementary operations of addition / subtraction and reset to zero. FB-transition determines the enabling of the corresponding transition of an A-net and sets the change of the marking of neighboring FB-places when it fires. FB-dispatcher manages the firing of transitions in the A-net. When interpreting an A-net, the principle of locality of changing the marking of places and statuses of transitions in the vicinity of the fired transition is used. This optimizes the interpretation algorithm and reduces the system response time, since the statuses of a few (but not all) transitions are recalculated.

The rule for transforming a transition of selective A-nets to a FB is shown in Fig. 3.

This FB-transition has the following interface. *Event inputs*: *INIT* is "FB initialization"; *fire* is "fire the enabled transition"; *obj* is "arrival of a new object"; chp_i is "change marking of place p_i". *Event outputs*: *INITO* is "FB initialization completed"; *en* is "the transition has become enabled"; *dis* is "the transition is no longer enabled"; *fired* is "the transition has fired". *Information inputs*: a_k is "the k-th parameter of the input object"; m_i is "a new marking of place p_i"; b_i (c_i) is "the first (second) component of the weight of the input arc from place p_i"; r_i (s_i) is "the first (second) component of the weight of the output arc to place q_i". *Note*: inputs of types *b, c, r, s* are optional, since they can be rigidly embedded inside the FB in

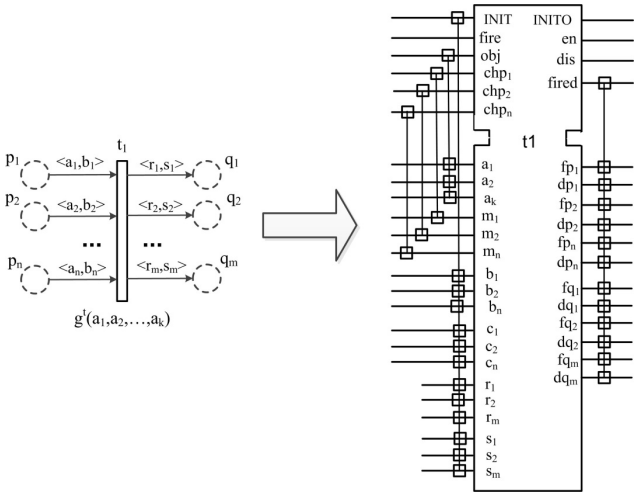

Fig. 3. Transformation of a transition of a selective A-net to a FB

processing algorithms. However, putting them out allows us to build self-modifying systems (by modifying these parameters). *Information outputs*: fp_i (fq_i) is "the modifier of the action on the marking of place p_i (q_i)"; dp_i is "the number of removed tokens from place p_i"; dq_i is "the number of added or assigned tokens to place q_i".

The dispatcher stores the current list of enabled transitions. When a signal arrives from input en_i, this list is replenished with transition t_i. When a signal arrives from input dis_i, the transition t_i is removed from this list. When a signal arrives at

the *exe* input, the dispatcher selects the highest priority enabled transition for activation, and issues signal to fire it.

Below is a demonstration example of the use of selective A-nets to build a product line of LEGO constructions (in the form of bars with a square cross-section, ending by a triangular cover). An instance of such a product line in the form of a "house" is shown in Fig. 4.

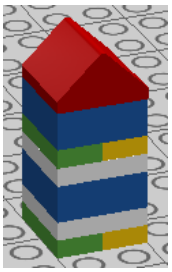

Fig. 4. Instance of product line of LEGO structures

When building LEGO constructions, objects (i.e. LEGO elements) with the following parameters are used: 1) *t* is a form (the possible values are: *R* is "a rectangular parallelepiped", *T* is "a triangular prism"); 2) *c* is a color (the possible values are: "*green* (*g*)", "*yellow* (*y*)", "*white* (*w*)", "*blue* (*b*)", "*red* (*r*)"); 3) *s* is the ratio of the rectangle sides lengths (the possible values are: "2:1", and "2:2"). It is assumed that objects move along one or more conveyors, and an assembly robot equipped with appropriate sensors selects the necessary objects from the flow.

The order of selection of objects is specified by the A-net represented in Fig. 5.

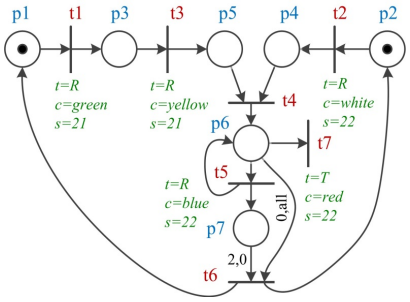

Fig. 5. Selective A-net for building LEGO constructions production line

This A-net determines all possible selectable objects' sequences. Firing a transition means that the corresponding object is selected. It is considered that the priority of a transition is inversely proportional to its number. It should be noted that the mechanism of priorities does not restrict the possibility of parallelism, but only regulates the functioning of A-net while two or more objects arrive simultaneously. The final transition is transition t_7. An unweighted arc of the A-net implies the labelling <1,1>. In principle, certain actions can be associated with transitions and places of the A-net, for example, technological operations. In the demo above, this is not explicitly done. Nevertheless, it can be assumed that there is some abstract assembly robot that builds LEGO assemblies.

In accordance with the method presented above, the A-net shown in Fig. 5 was implemented as a FB application.

V. CONCLUSION

This paper presents approaches to the IEC 61499 FB–based implementation of the most popular in industrial automation state transition models. When defining the scope of application, the recognition and selection tasks were prioritised, although it can be extended without essential problems to control tasks. The directions of further research are: 1) the investigation of distributed implementation of STM especially with regard to their validation and verification; 2) the implementation of more complex state transition models, for example, colored timed Petri nets and abstract state machines, as well as expanding the scope of their use in various fields of industrial automation.

REFERENCES

[1] T. Strasser, G. Ebenhofer, M. Rooker, I. Hegny, "Multi-Domain Model-Driven Design of Industrial Automation and Control Systems," Proceedings of 13th IEEE International Conference on Emerging Technologies and Factory Automation, Hamburg, Germany, September 15–18, 2008.

[2] K. Thramboulidis, "Model-Integrated Mechatronics – Toward a New Paradigm in the Development of Manufacturing Systems," IEEE Transactions on Industrial Informatics. 2005, Vol. 1, Issue 1, pp. 54–61.

[3] J.E. Hopcroft, R. Motwani, J.D. Ullman, Introduction to Automata Theory, Languages, and Computation (3rd Edition), Pearson; 3 edition, 2006, 560 p.

[4] V. Dubinin, I. Senokosov, V. Vyatkin, "Auto-Generation of Distributed Automation Software Based on Formal Product Line Specification," In: Mařík V., Wahlster W., Strasser T., Kadera P. (eds) Industrial Applications of Holonic and Multi-Agent Systems. HoloMAS 2017. Lecture Notes in Artificial Intelligence, vol. 10444. Springer, Cham, pp.80–91.

[5] A.A. Shalyto, N.I. Tukkel, "SWITCH Technology: An Automated Approach to Developing Software for Reactive Systems," Programming and Computer Software, 2001, Volume 27, Issue 5, pp. 260–276.

[6] T. Murata. "Petri nets: Properties, analysis and applications," Proceedings of the IEEE, Volume: 77 , Issue: 4 , 1989, pp. 541–580.

[7] Y. Gurevich, "Evolving Algebras 1993: Lipari Guide, Specification and Validation Methods," Oxford : University Press, 1995, pp. 9–36.

[8] Vyatkin V. Function Blocks for Embedded and Distributed Control Systems Design, Third Edition. – Instrumentation Society of America (ISA), 2016, 261 p.

[9] nxtControl Web site. – URL:http://www.nxtcontrol.com/

[10] N. Ghanem, D. DeMenthon, D. Doermann, L. Davis, "Representation and Recognition of Events in Surveillance Video Using Petri Nets," Conference on Computer Vision and Pattern Recognition (CVPRW '04), 2004.

[11] C. Choppy, O. Bertrand, P. Carle, "Coloured Petri Nets for Chronicle Recognition," Int. Conf. on Reliable Software Technologies - Ada-Europe 2009. Lecture Notes in Computer Science, vol 5570, Springer, pp. 266–281.

[12] V.R. Fernández, A.G. Pardo, D. Camacho, "Automatic Procedure Following Evaluation using Petri net-based Workflows," IEEE Transactions on Industrial Informatics, Volume: 14, Issue: 6, 2018, pp.2748-2759.

[13] M. de Leoni, J. Munoz-Gama, J. Carmona, W.M.P. van der Aalst, "Decomposing Alignment-Based Conformance Checking of Data-Aware Process Models," Lecture Notes in Computer Science, vol 8841, 2014, pp. 3–20.

[14] V.N. Dubinin, "Asynchronous modeling of NCES-nets," News of higher educational institutions. Volga region. Technical science. 2009, № 2, pp. 3–14. (in Russian)