jet INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2012, VOL. 58, NO. 1, PP. 27–34
Manuscript received December 31, 2011; revised March 2012. DOI: 10.2478/v10177-012-0004-8

UML Modelling in Rigorous Design Methodology for Discrete Controllers

Grzegorz Łabiak, Marian Adamski, Michał Doligalski, Jacek Tkacz, and Arkadiusz Bukowiec

***Abstract*—The paper presents an application of UML technology in a discrete system development process. In the process at the analysis stage UML diagrams are fundamental tool. The outcome of this stage is a basis for formal models exploited at the design stage, where the design is symbolically verified and treated as a rule-based system. Two formal models of good graphical appeal are proposed: Petri nets and state machine diagrams. Both are heavily using Boolean expressions what makes that design can easily be implemented in modern programmable structures.**

***Keywords*—UML modelling, binary controller, decomposition, digital synthesis, formal analysis, verification.**

I. Introduction

DISCRETE system is a system with countable number of states. One of such systems is a discrete control system, where the controller generates signals to the controlled object and the controlled object responds to the controller (Fig. 1) [1]. Moreover, the controller receives signals from an operator and generates signals to the operator. For an engineer the design of the system mainly involves the control unit design and the controlled object and its behaviour are requirements given by a client. If the signals are of binary value the controller is a binary controller and can be produced as a digital circuit (eg. FPGA or CPLD) [2], [3].

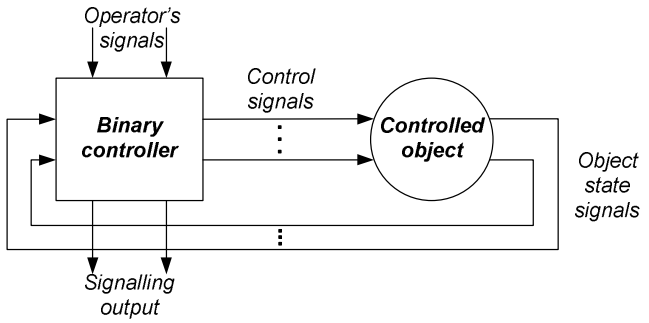


Fig. 1. Discrete control system.

The Discrete system development process (Fig. 2) consists of the four following phases: analysis, design, implementation and maintenance. The main goal of the system analysis is requirements definition and feasibility study (both functional, technological and economical). The outcome of the analysis phase is informal and makes ground for formal specification

This work was supported by the Ministry of Science and Higher Education of Poland. Research grant no. N516 513939 for years 2010-2013.

G. Łabiak, M. Adamski, M. Doligalski, J. Tkacz and A. Bukowiec are with the Institute of Computer Engineering and Electronics, University of Zielona Góra, Licealna 9, 65-417 Zielona Góra, Poland (e-mails: {g.labiak, m.adamski, m.doligalski, j.tkacz, a.bukowiec}@iie.uz.zgora.pl).

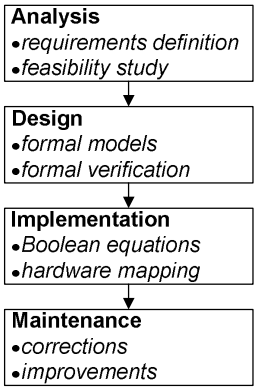


Fig. 2. Discrete system development process.

(eg. FSM, Petri nets, statechart diagrams). Formal specification is the main goal of the next stage, mainly, design phase. The main tool of the analysis phase is UML technology [4].

After the system takes formal form it can be formally and automatically verified using, for example, model checking techniques [1], [5] (applied to statechart diagrams as a state oriented system) or can be treated as a dynamic inference system based on Gentzen logic (applied to Petri nets) [1]. In comparison with testing and simulation formal verification (model checking technique) proves that given property holds for every state of the discrete system, while testing verifies only small subset of space of possible states. Having formal model transformed into rule-based decision system allows to state about validity of the system in straightforward way and any transformation (eg. minimization for logic synthesis) of the specification (eg. Petri net) preservers its meaning.

Implementation is the last technological phase. After the system works in accordance with defined requirements and is well formed (eg. free from deadlocks, livelocks and traps) it can be mapped into technological resources. At this stage the discrete system can be subject to testing. In maintenance phase correctly designed and produced system is functioning as it was desired and can be modified in accordance with the newly emerged needs. The whole development process (Fig.2) is repeated and the system is reimplemented.

II. UML Modelling

An application of UML technology in discrete system design at analysis stage is very similar to its original application in software engineering. Its main goal is to create simplified blue print of the system under design, and the process involves stages as follows:

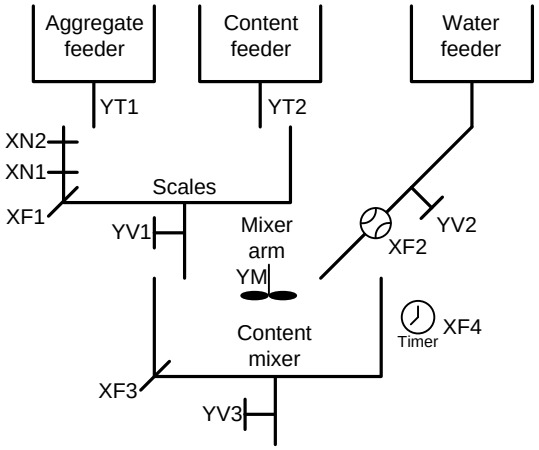


Fig. 3. Schematic diagram of process plant.

- Object-oriented analysis – the goal of this process, usually performed at the beginning, is not making an object abstraction of real world entities but identification of material object in real world (ie. controlled object), eg. controller, operator, scale, valve. This process uses mainly UML class and object diagrams.
- Functional analysis – the goal of this process is a description what the system performs and what the controlled objects identified at the previous stage do. This process uses UML use case diagrams and describes relationships between particular functionalities.
- Behaviour analysis – the goal of this stage is to describe how particular functionalities are executed and which object takes part in it. The UML diagrams useful at this stage are: activities, sequential, collaboration and state machine.

The last stage should give solid but not formal framework for further formal description in Petri nets or statechart diagrams. It is note worthy that in the discrete system design activity diagrams and state machine model play the same role like class diagram in software engineering. Both can be easily transformed into their formal counterparts, namely, Petri nets and statechart diagrams respectively.

III. The Example

A designer assignment is to design control algorithm of a technological process [6], [7]. The technological process is carried out in a chemical plant depicted in Fig. 3. The process mixes two substrates contained in two containers (called *Aggregate feeder* and *Content feeder*) in water environment. The two substrates are measured up sequentially in one scale and next are mixed together in main container (*Content mixer*) for a given period of time. The process is repeated over and over. To optimize time consumed by whole process the substrate from *Aggregate feeder* can be measured out simultaneously with mixing final product of previous technological cycle.

At first glance at chemical plant designer can identify its main objects. These objects are presented in Fig. 4. Three of them *Aggregate*, *Content* and *Water* are of *CContainer* class,

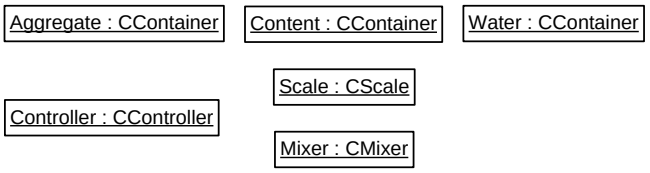


Fig. 4. Object diagram with main controlled objects.

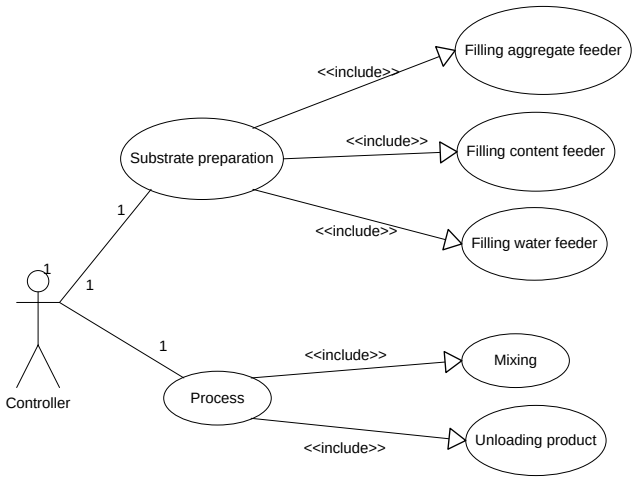


Fig. 5. Use case diagram of the control system.

as they play role of simple container, the rest are of its own class, namely, *CController*, *CScale* and *CMixer*.

After having informally acquainted and identified control system designer can formulate main functions of the control object. For the control system (in sense of Fig. 1) its two main functions are *Substrate preparation* and *Process*. Further in-depth analysis allows to determine more detailed functions which are included in main functions. Use case diagram from Fig. 5 presents these functions. Stereotype *«include»* communicates that simple use cases (eg. *Filling aggregate feeder*) make up more general use cases (eg. *Substrate preparation*). In the diagram the controller object is an actor, namely, this means that controller object starts interactions in the control system.

Object diagram and use case diagram allow to define basic UML artefacts (eg. objects nad system functions) upon which abstract model can be build. Next designer can specify how system objects interact each other to perform given function. Sequential diagrams and 7) in two dimensions describe how particular objects exchange communications in order to execute system functions. In one dimension (horizontally) are placed objects and in the second dimension (vertically) is placed axis of time. Arrows symbolize exchanged messages and labels placed above them are conditions which must be met to generate message. In case of control system the conditions are binary signals, actuators and sensors which, respectively, open/close valves or inform about liquid level in containers.

Fig. 6 presents sequential diagram for the use case *Substrate preparation*. Every interaction starts from actor object, namely from *Controller*. Substrate preparation involves preparing sep-

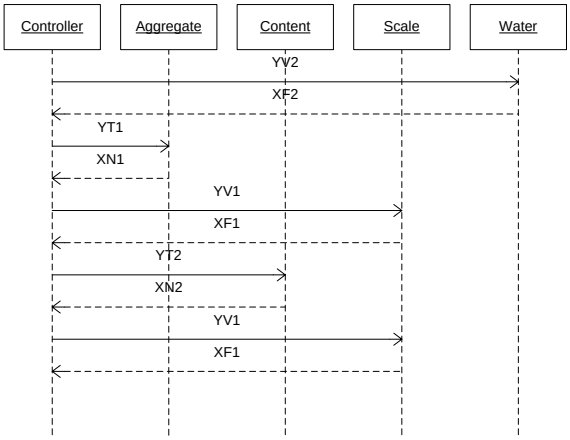


Fig. 6. Sequential diagram for the use case *Preparation substrates*.

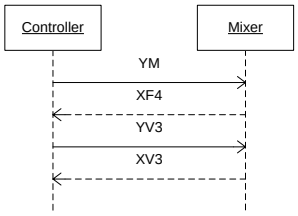


Fig. 7. Sequential diagram for the use case *Process*.

arately two components and water. Fig. 7 presents sequential diagram for *Process* use case. This relatively simple functionality is only responsible for mixing product and its unloading.

The main goal of UML modeling is to prepare abstract model of complex system. Such a model should not be as complex as modeled system. It should contain as much information as it is needed to comprehend main function of the system, its basic internal working and to visualize them graphically. UML language is very plentiful of diagrams and constructs and hence designer must make reasonable trade-off between UML potential and his/her needs. Other UML diagrams which could be useful in controller design are collaboration diagrams, deployment diagrams, activity diagrams and state machine. The two latter diagrams can be basis upon which formal model of the controller is specified. The two formal models are state-machine-based model and Petri nets. Next two chapters present in detail application of these two models in synthesis of digital controller.

IV. State Machine Based Approach

UML state machine diagrams are formal, state based method of behavioral specification. Ability to model concurrent processes, exception handling [8] and hierarchy allows the use of state machine diagram in the process of logic controllers specifications. User-friendly interface makes it easy to develop diagrams, but the lack of formal methods makes it difficult to formally verify the controller. In the proposed approach,

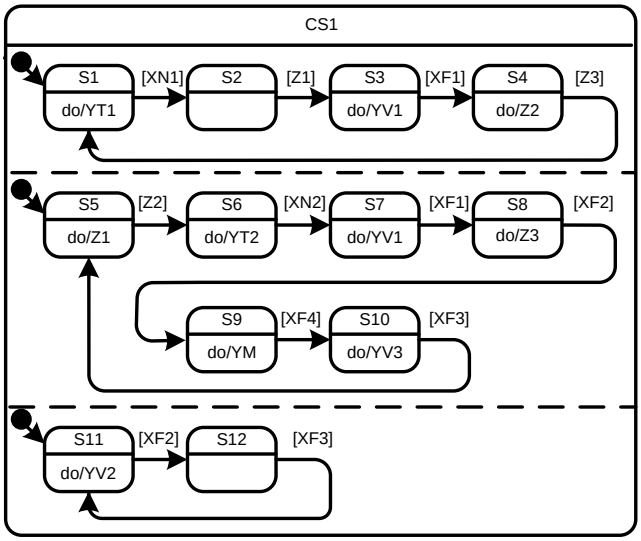


Fig. 8. UML state machine diagram.

UML state machine diagrams are front-end of the behavioral specifications, Petri nets are the back-end.

For given industrial process the UML state machine diagram was elaborated (Fig. 8). In order to accurately compare the two approaches, developed diagram implements the control algorithm described using Petri net (Fig. 11).

State machine diagram consists of two levels of abstraction. At the first level there is only one pseudostate (*initial*) and composite state CS_1. At the second level of abstraction, the composite state exists. It consists of three concurrent submachines. In order to properly implement control algorithm mapping, the synchronization through variables is necessary [9]. Three global variables Z_1, Z_2, Z_3 are broadcast in states S_5, S_4, S_8. In this particular case exceptions handling mechanism is not required.

As it was said, the UML state machine diagram is only the user interface and next step of the logic controller developing process is transformation into Petri net. Lack of exceptions handling mechanism in the logic controller, allowed to transform state machine model into the interpreted Petri net (Fig. 9). The top level of abstraction of the state machine corresponds to the Net_0 subnet, it consists only of two places: P_1 and MP_1. Second level of abstraction consists of submachine described in composite state CS_1, which corresponds to Net_1 subnet.

Net_1 subnet is started when macroplace MP_1 is marked. The transition $T^{init}2$ has guard condition specified as $T^{init}1$. It is a global variable, specified in Net_1 subnet, where it is the macroplace MP_1 initial transition. Subnet Net_2 implements three concurrent area from composite state CS_1.

At the RTL level, the logic controller is decomposed into two blocks, each for one subnet (Fig. 10). In the proposed approach, the hierarchy is maintained at every stage of the design process.

This may cause a slight increase in the use of hardware resources, which is acceptable because this way of decomposition makes it easier to carry out the process of the logic controller module based partial reconfiguration [3].

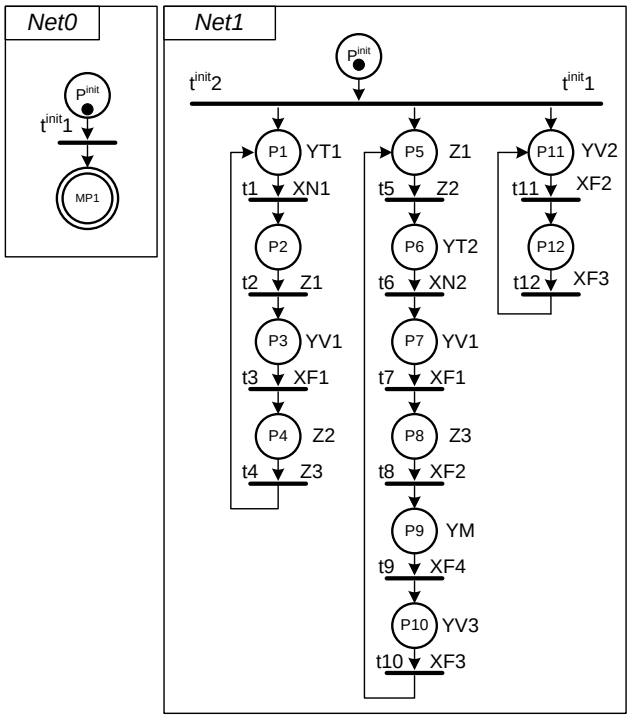


Fig. 9. State machine based Petri net.

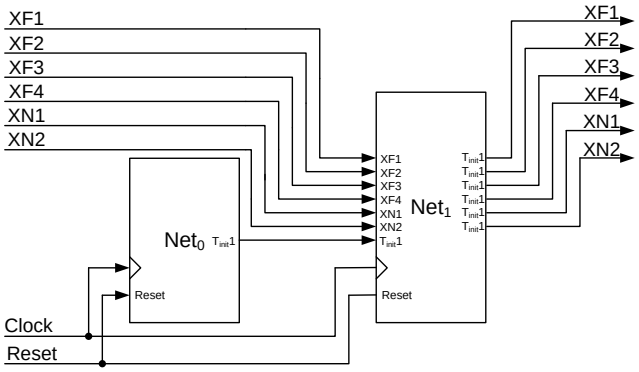


Fig. 10. Top-level module of state machine Petri net based.

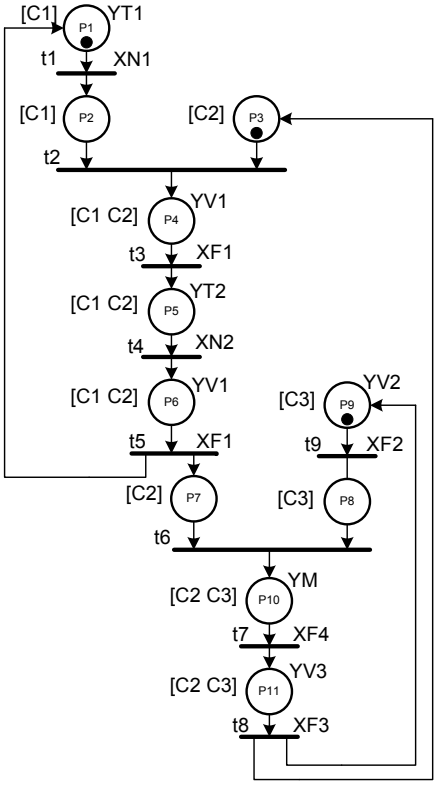


Fig. 11. Interpreted colored Petri net.

V. Petri Net Based Approach

This approach shows how to design logic controller using only Petri net specification. The usage of Petri net allows to preform analysis process using well known and effective algorithms [10], [11]. Additionally, by application of coloring and decomposition into linked state machines (LSMs) logic controller can be implemented as a distributed system without use of any transitional specifications [12]. The whole design process is illustrated by analysis and synthesis of example Petri net (Fig. 11). This Petri net describes control process of industrial mixer of aggregate with content and water presented in chapter III.

A. Analysis of Petri Net by Using Sequent Calculus System

The proposed method of analyzing of Petri net checks liveness and looks for any possible defects [13]. When Petri net is correct the coloring is preformed. Colored Petri net is required for synthesis purpose. The analyzing and coloring are done with use of sequent deduction system. To simplify the calculation process Petri net is replaced by equivalent macronet [11].

1) Gentzen deduction system: The sequent is a formalized statement used for deduction and calculi [14]. In the sequent calculus, sequents are used for specification of judgment that are characteristic to deduction system. The sequent is defined as a ordered pair (Γ, Δ), where Γ and Δ are finite sets of formulas, and $\Gamma = \{A_1, A_2, \ldots, A_m\}$, $\Delta = \{B_1, B_2, \ldots, B_n\}$. Instead of (Γ, Δ) it is used notation with use of turnstile symbol $\Gamma \vdash \Delta$. Γ is called the antecedent and Δ is a succedent of the sequent. The sequent $\Gamma \vdash \Delta$ is satisfiable for the valuation v iff for the same valuation v the formula $\bigwedge_{i=1}^{m} A_i \rightarrow \bigvee_{j=1}^{n} B_j$ is satisfied. In proposed implementation of Gentzen system there are defined ten rules of elimination of logic operators. For each operator (negation, disjunction, conjunction, implication and equivalency), there are defined two rules of its elimination. First rule is used when the operator is located in antecedent and the second one when it is located in succedent. The elimination process is repeated while only normalized sequents are received. The normalized sequent is a sequent without any logical operators. Sequent is a tautology iff it has the same formula in a antecedent and a succedent. The located tautology sequents could be removed from further normalization. Iff all received sequents are tautology the analyzed sequent is also tautology. When one of received sequents is not a tautology it means that it is a counter example for analyzed sequent.

TABLE I
MACRONET DEFINITION

Macroplaces	Places
MP_1	$\{P_1, P_2\}$
MP_2	$\{P_3\}$
MP_3	$\{P_4, P_5, P_6\}$
MP_4	$\{P_7\}$
MP_5	$\{P_8, P_9\}$
MP_6	$\{P_{10}, P_{11}\}$

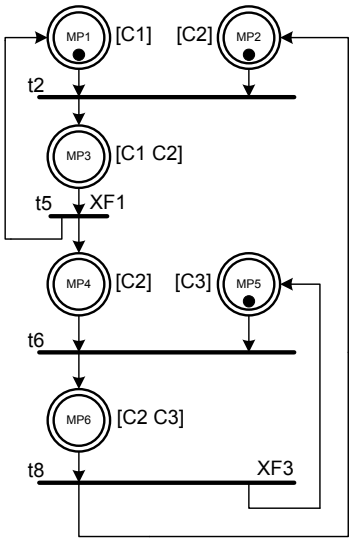


Fig. 12. Interpreted colored Petri macronet.

2) Analysis and coloring: Original Petri net is replaced by equivalent macronet [11] to simplify analysis and coloring processes. To receive equivalent macronet each sequential chain of places have to be replaced by one macroplace. For our example all sequential chain of places with corresponding macroplaces are shown in Table I and received macronet is shown in Figure 12.

First step of analysis is generation of sequent formulae for conducted Petri macronet. Such sequent is generated twice: once for siphons and once for traps. For macronet presented in Fig. 12 there are two sequents: $\mathbf{S_S}$ describes siphons and $\mathbf{S_T}$ describes traps:

$$\begin{aligned}\mathbf{S_S}:&((MP_1+MP_2)\rightarrow MP_3)\cdot\\&(MP_3\rightarrow(MP_4+MP_1))\cdot\\&((MP_4+MP_5)\rightarrow MP_6)\cdot\\&(MP_6\rightarrow(MP_2+MP_5)))\vdash;\\\mathbf{S_T}:&(MP_3\rightarrow(MP_1+MP_3))\cdot\\&((MP_4+MP_1)\rightarrow MP_3)\cdot\\&(MP_6\rightarrow(MP_4+MP_5))\cdot\\&((MP_2+MP_5)\rightarrow MP_6))\vdash;\end{aligned}$$

Then there are generated proof trees for both sequents. Proposed Gentzen system is effective and original realization of algorithm of sequent normalization [14], [15]. This tool is suitable for solving combinational problems in the digital system design. It is able to transform very complicated behavioral descriptions into simple expressions. During deduction process system generates tree proof that is formal proof for this process. For considered example following results were received (left side of the sequents S_0, S_5 and S_7 are minimal siphons and left side of the sequents T_3, T_4 and T_3 are minimal traps):

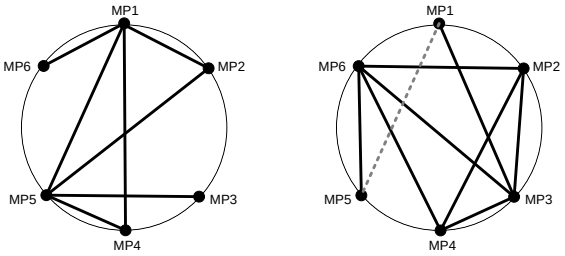
$$\begin{aligned}\mathrm{S}_0:&MP_2,MP_3,MP_4,MP_6\vdash;\\\mathrm{S}_1:&MP_3,MP_4,MP_5,MP_6\vdash;\\\mathrm{S}_2:&MP_1,MP_2,MP_3,MP_6\vdash;\\\mathrm{S}_3:&MP_1,MP_3,MP_5,MP_6\vdash;\\\mathrm{S}_4:&MP_1,MP_2,MP_3\vdash MP_4,MP_5;\\\mathrm{S}_5:&MP_1,MP_3\vdash MP_4,MP_5,MP_6;\\\mathrm{S}_6:&MP_4,MP_5,MP_6\vdash MP_1,MP_2;\\\mathrm{S}_7:&MP_5,MP_6\vdash MP_1,MP_2,MP_3;\\\mathrm{S}_8:&\vdash MP_1,MP_2,MP_3,MP_4,MP_5,MP_6;\\\mathrm{T}_0:&MP_1,MP_3,MP_4,MP_6\vdash;\\\mathrm{T}_1:&MP_1,MP_3,MP_4\vdash MP_2,MP_5;\\\mathrm{T}_2:&MP_1,MP_3,MP_5,MP_6\vdash;\\\mathrm{T}_3:&MP_1,MP_3\vdash MP_2,MP_5,MP_6;\\\mathrm{T}_4:&MP_2,MP_3,MP_4,MP_6\vdash;\\\mathrm{T}_5:&MP_2,MP_3,MP_5,MP_6\vdash;\\\mathrm{T}_6:&MP_2,MP_5,MP_6\vdash MP_1,MP_4;\\\mathrm{T}_7:&MP_5,MP_6\vdash;MP_1,MP_3,MP_4;\\\mathrm{T}_8:&\vdash MP_1,MP_2,MP_3,MP_4,MP_5,MP_6;\end{aligned}$$

TABLE II
SIPHONS AND TRAPS

Minimal siphons	Minimal traps
$MP_1, MP_3 \vdash;$	$MP_1, MP_3 \vdash;$
$MP_2, MP_3, MP_4, MP_6 \vdash;$	$MP_2, MP_3, MP_4, MP_6 \vdash;$
$MP_5, MP_6 \vdash;$	$MP_5, MP_6 \vdash;$

Siphons S_1 and S_3 and S_6 are reduced respectively by smaller siphons S_7. Siphons S_2 and S_4 are reduced by smaller siphons S_5. Traps T_0 and T_5 and T_6 are reduced respectively by smaller traps T_7 by analogy. No siphons and traps are described by S_8 and T_8 sequents (they have not left side of the sequent where places have positive value). The minimal sets of siphons and traps for our example Petri macronet are shown in Table II. Basing on algorithm for finding all siphons and traps in Petri net and checking dependencies between sets of siphons and traps, it is possible to answer the question if Petri net is live. Analyzed Petri net is live because all minimal siphons contain marked traps in initial marking.

The adjacency graph of concurrency (Fig. 13a) is created based on the typical reachability graph. The generation of reachability graph has exponential complexity. The lists of concurrent places is presented in Table III. Products of logical expressions define concurrency in analyzed Petri macronet. The monotone characteristic sequent ($\mathbf{S_C}$) of Petri net discrete



(a) Adjacency graph of concurrency (b) Graph of invariants

Fig. 13. Adjacency graph of concurrency and graph of invariants.

TABLE III
CONCURRENT PLACES

List of concurrent places	Logical expressions
MP_1, MP_2, MP_5	$(MP_1 \cdot MP_2 \cdot MP_5)$
MP_3, MP_5	$(MP_3 \cdot MP_5)$
MP_1, MP_4, MP_5	$(MP_1 \cdot MP_4 \cdot MP_5)$
MP_1, MP_6	$(MP_1 \cdot MP_6)$

TABLE IV
INVARIANTS OF THE PETRI MACRONET

Invariant	Color	Places
	C_i	P_i
$MP_1, MP_3 \vdash;$	C_1	$\{P_1, P_2, P_4, P_5, P_6\}$
$MP_2, MP_3, MP_4, MP_6 \vdash;$	C_2	$\{P_3, P_4, P_5, P_6, P_7, P_{10}, P_{11}\}$
$MP_1, MP_5 \vdash;$	–	–
$MP_5, MP_6 \vdash;$	C_3	$\{P_9, P_8, P_{10}, P_{11}\}$

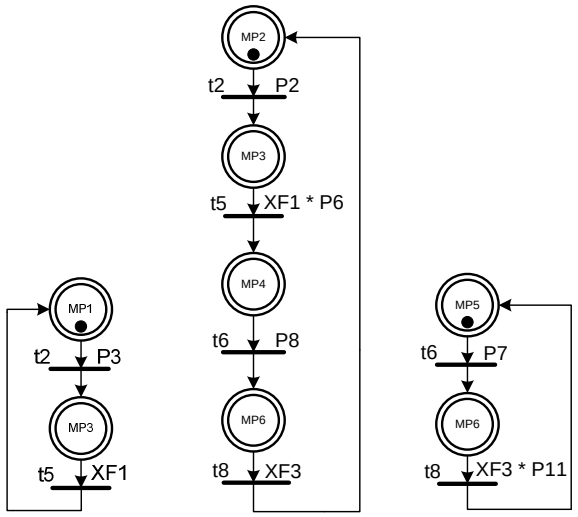
space is as follows:

$$\mathbf{S_C} : \vdash (MP_1 \cdot MP_2 \cdot MP_5), (MP_3 \cdot MP_5), (MP_1 \cdot MP_4 \cdot MP_5), (MP_1 \cdot MP_6);$$

After sequent calculation following sets of them were received (Tab. IV). Each color corresponds to one invariant. The invariant MP_1, MP_5 was rejected because it contained more than one initially marked place (Fig. 13b).

B. From Petri Net to LSMs

The transformation form Petri net to LSMs is based on decomposition of Petri net using symbolic deduction method. The decomposition algorithm will be described generally on an example (in detail it was described in [15], [16]). During the process a Petri net is divided into a set of subnets. These subnets have to satisfy some restriction, e.g. a subnet must include only places which are sequential to each other or cannot contain multi-input or multi-output transitions [16]. Decomposition of Petri net can be based on coloring of Petri macronet. For decomposition invariants are used. Following algorithm is based on the known method of coloring not oriented graphs [10], [15]. The coloring of sample Petri net is presented in Table IV. If it is possible to color Petri net, it means that



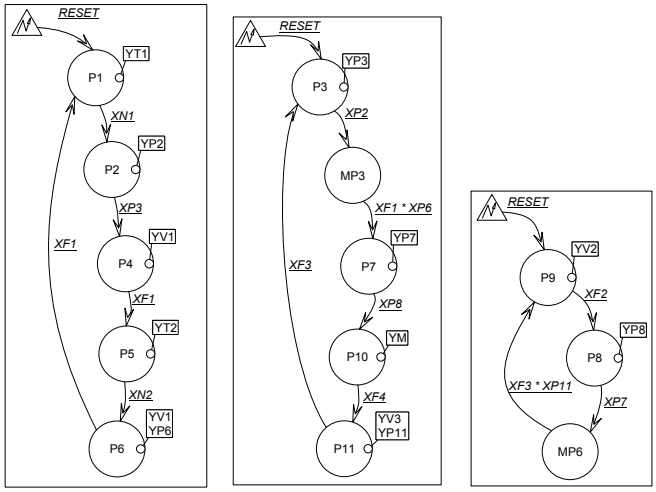
(a) subnet C_1 (b) subnet C_2 (c) subnet C_3

Fig. 14. Decomposed Petri macronet into macrosubnets.

this Petri macronet could be decomposed into macrosubnets. Each color corresponds to one macrosubnet. Because multi-input and multi-output transition occurs in more than one macrosubnet there have to be added or extended approbate firing condition. This extension of condition consists of logic conjunction of all its input places from other macronets and it is made in conjunction with already existing firing condition. There are four such transitions in our example Petri net, two multi-input: t_2 and t_6 and two multi-output: t_5 and t_8. In case of multi-input transition firing condition in all macrosubnet have to be modified. In our example, the transition t_2 occurs in macrosubnets C_1 and C_2 and there have to be added P_3 as firing condition of this transition in macrosubnet C_1 and P_2 in macrosubnet C_2. P_8 and P_7 have to be added to the transition t_6 in macrosubnet C_2 and in macrosubnet C_3 by analogy. In case of multi-output transition firing condition do not have to be modified in first macrosubnet where it occurs. The transition t_5 occurs in macrosubnets C_1 and C_2 and there have to be added P_6 as firing condition of this transition in macrosubnet C_2. There is no need to add any condition in macrosubnet C_1 because there is only one input place P_6 into this transition t_5. P_{11} have to be added to the transition t_8 in macrosubnet C_3 by analogy. Macrosubnets for our example Petri macronet are presented in Figure 14.

Now, macrosubnets can be transformed into LSMs. Each macrosubnet represents one FSM. Because such macrosubnets satisfy all restrictions mentioned in the beginning of this paragraph they can be transformed to FSMs. The macrosubnet represented by first color C_1 is transformed in this way (Fig. 15a):

- Each macroplace is replaced by sequence of states corresponding to places from original Petri Net;
- Each transition is converted into FSM transition;
- If firing condition consist a place, the additional input signal xp_m have to be created. Also the additional output signal yp_m have to be added to adequate FSM, that



(a) FSM C_1 (b) FSM C_2 (c) FSM C_3

Fig. 15. Decomposed Petri net into LSMs.

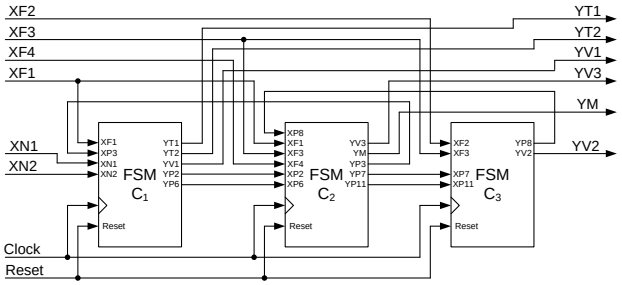


Fig. 16. Top-level module of LSMs implemented Petri net.

TABLE V
SYNTHESIS RESULTS OF THE PETRI NET

Logic Utilization	Structure		
	PN	SMs	LSMs
Number of Slices	7	12	8
Number of Slice Flip Flops	11	14	13
Number of 4 input LUTs	12	16	15

consists state P_m, as Moore type output, that is generated in this state P_m.

Following macrosubnet represented by following colors C_i are transformed in this way (Figs. 15b-15c):

- Each macroplace, that do not occurred in previous macrosubnets represented by colors from C_1 to C_{i-1}, is replaced by sequence of states corresponding to places from original Petri Net;
- Each macroplace, that occurred in any previous macrosubnets represented by colors from C_1 to C_{i-1}, is replaced by one state corresponding to this macroplace. This state does not generate any output signals;
- Transitions and its conditions are converted in the same way like for macrosubnet represented by first color C_1.

Received LSMs for our example Petri net are presented in Figure 15. There have to be added transition form state to itself (called itself transition) in all received FSMs if corresponding transition in Petri net or macrosubnet has any condition. The itself transition go from and into the same state like considered transition. The condition of this itself transition is the negation of condition of firing transition. This step is required to hold the state. Sometimes, these transitions are not presented in graphical representation to make it more readable and they can be also omitted in some templates of behavioral description in HDLs because variable that store the state also hold its value. In our example these transition are not added because there is used one of such VHDL templates to describe FSMs.

To implement whole system it is required to create top-level module that connects LSMs together. In general, the top-level module should satisfy these conditions:

- All FSMs should have the same clock and reset signals;
- All FSMs should have input signals belonging to the subset of the same set of input signals;
- Additional output signals yp_m should be connected with corresponding additional input signals xp_m in others FSMs;
- In case of an output signal could be generated under control from more than one component, corresponding output signals from all FSMs should be connected together in final output signal via OR-gate.

For our example the top-level module is presented in Figure 16.

VI. SUMMARY

Presented examples were synthesized into device from Xilinix Virtex family. The obtained results of device utilization are shown in Table V in the column SMs are shown results for the state machine based approach and in the column LSMs are shown results for the decomposed Petri net into LSMs. All models were described as behavioral description in VHDL [17]. For comparison purpose in the column PN are presented results after synthesis of behavioral description oriented on places in VHDL [18] of initial Petri net. Increased use of hardware resources is the price of friendlier forms of modeling. The slight increase in resources is negligible in view of currently produced, very roomy reprogrammable digital circuits. The advantage of LSMs is possibility of distributed implementation [12]. In this case each FSM can be implemented into different integrated circuit. It gives possibility to place each part of controller near to adequate control object.

UML diagrams make that designer better understand working of the system under design. Doing so the model specifies, documents and visualizes different aspects of the system making that further changes of the system or changes in the client requirements can efficiently be applied. UML language is very rich in different semantic structures and the designer should make balanced trade-off. UML model should be simple enough to communicate main functionalities and internal working of the system under design. Doing so UML model is informal, but state machine and activity diagrams can be solid base for formal models, like Petri nets or FSMs. The formal models give advantage of application of verification (well formed model) and synthesis algorithms.

REFERENCES

[1] M. Adamski, M. Węgrzyn, and A. Karatkevich, *Design of embedded control systems*. New York: Springer, 2005.

[2] G. Łabiak and G. Borowik, "Statechart-based controllers synthesis in fpga structures with embedded array blocks," *International Journal of Electronics and Telecommunications*, vol. Vol. 56, no. no 1, pp. 13–24, 2010.

[3] M. Doligalski and M. Węgrzyn, "Partial reconfiguration-oriented design of logic controllers," *Proceedings of SPIE : Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2007*, vol. Vol. 6937, p. [10], 2007.

[4] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language. User Guide*. New York: Addison Wesley Longman, Inc., 1999.

[5] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. Cambridge, Massachusetts: The MIT Press, 1999.

[6] P. Misurewicz, "Lectures on real-time microprocessor control systems," in *Lecture Notes*. Twin Cities: University of Minnesota, 1976.

[7] L. Gniewek and J. Kluska, "Hardware implementation of fuzzy Petri net as a controller," *IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics*, vol. Vol. 34, no. No. 3, pp. 1315–1324, 2004.

[8] M. Doligalski and M. Adamski, "Exceptions and deep history state handling using dual specification," *Electrical Review*, no. No. 9, pp. 123–125, 2010.

[9] G. čabiak and M. Adamski, "Concurrent processes synchronisation in statecharts for FPGA implementation," in *Proceedings of IEEE East-West Design & Test Symposium EWDTS'08*, Kharkov National University of Radioelectronics. Lviv, Ukraine: Lviv, The Institute of Electrical and Electronics Engineers, Inc., 2008, pp. 59–64.

[10] K. Biliński, M. Adamski, J. Saul, and E. Dagless, "Petri-net-based algorithms for parallel-controller synthesis," *IEE Proceedings – Computers and Digital Techniques*, vol. Vol. 141, no. No. 6, pp. 405–412, 1994.

[11] A. Karatkevich, *Dynamic Analysis of Petri Net-Based Discrete Systems*, ser. Lecture Notes in Control and Information Sciences. Berlin: Springer-Verlag, 2007, vol. 356.

[12] A. Bukowiec and L. Gomes, "Partitioning of Mealy finite state machines," in *Preprints of the 4th IFAC Workshop Discrete-Event System Design DESDes'09*, Gandia Beach, Spain, 2009, pp. 21–26.

[13] A. Węgrzyn, "Parallel algorithm for computation of deadlocks and traps in Petri nets," in *10th IEEE International Conference Emering Technologies and Factory Automation ETFA'05*, vol. 1, Universita di Catania. Catania, Italy: Piscataway, IEEE Operation Center, 2005, pp. 143–148.

[14] J. H. Gallier, *Logic for Computer Science: Foundations of Automatic Theorem Proving*. New York: Harper & Row Publishers, 1985. [Online]. Available: http://www.cis.upenn.edu/ jean/gbooks/logic.html

[15] J. Tkacz, "State machine type colouring of Petri net by means of using a symbolic deduction method," *Measurement Automation and Monitoring*, vol. Vol. 53, no. No. 5, pp. 120–122, 2007.

[16] M. Adamski, "Petri nets in ASIC design," *Applied Mathematics and Computer Science*, vol. Vol. 3, no. No. 1, pp. 169–179, 1993.

[17] M. Zwoliński, *Digital System Design with VHDL*, 2nd ed. New Jersy: Prentice Hall, 2004.

[18] M. Puczyńska, G. Łabiak, and P. Wolański, "Programowa implementacja konwersji sieci petriego na język VHDL," in *Materiały III Krajowej Konferencji Naukowej Reprogramowalne Układy Cyfrowe RUC 2000*, Szczecin, Poland, 2000, pp. 285–291.