

MULTI FDT INTERFACING

Štefan HUDÁK, Slavomír ŠIMOŇÁK

Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice, Letná 9, 042 00 Košice, tel. 055/633 5313, E-mail: hudak@tuke.sk, simonak@tuke.sk

SUMMARY

A widely recognized and accepted assertion in computer science community states that there will not exist a single formal description technique (FDT) for all aspects of the systems to be expressible. The paper tackle upon the issue of interfacing different FDTs for design and analysis phases of the discrete systems. Direct vs. indirect interfacing is treated. There is a number of reasons why a formal specification is useful, and it grows more important, as the designated system becomes larger and more complex. Very often that requirements are rather intuitive and imprecise, and are expressed in terms of problem domain they came from. Formal specification (of a system) is a formal expression of requirements on the system designed and thus lays between the requirements and destination code of the system. The advantage of formal description techniques (FDTs) usage is also the ability of validation of requirements understanding and detection of errors at early stages of system development [5]. For large and complex systems, testing cannot prove the right functionality of the system, because it is incomplete and thus does not check the behavior of the system in all reachable states.

Keywords: Formal Description Techniques (FDTs), Petri Nets, Process Algebra, B-AMN, Interfacing.

1. INTRODUCTION

The need of using FDT for discrete system design increases with complexity of designed system [10]. In the last decade, a great deal of work has been done on developing new FDTs and extending existing ones to suit better needs of a system designer. A number of tools which support design and analysis of systems using FDT have appeared and there are still further tools under development.

It should be pointed out that there is no general agreement on whether FDT should be used in design and analysis partially due to existing myths on FDTs and partially due to some requirements on formal thinking the FDTs pose onto users. Another arguments used by the opposition against FDT is that FDTs are suitable to cope only with toy examples and that they will fail in applying to systems of realistic size. The latter has been considered to be the main drawback of FDTs. On the other hand by using appropriate FDT, it is possible to detect some inaccuracies and to discover contradictions in specification of system in early phases of design yet in a systematic way. The choice of an appropriate FDT depends on properties of the designed system.

Different FDTs cover different sets of system properties and thus they express different views of the system. There is a strong belief among FDT experts that there will not exist any unique FDT to cover all desired aspects of designed system. The consequence of the latter is using a combination of FDTs that cover in a complementary way system properties. As an example can serve LOTOS and ESTELLE [4].

The process of creating a system formal description in a given FDT is usually time consuming. On the other hand, some property

(aspect) of the system can be more readily expressed in one FDT and it will require much more effort to get the same property expressed in another FDT. This is the reason why this work investigates possibilities of obtaining system description in different FDTs and to create ways how to go from one FDT to another. Thus, we deal with (multi)FDT interfacing. Namely we deal with a collection of three FDTs: Petri Nets, Process Algebra and B-AMN.

2. PETRI NETS

Petri Nets are used to describe processes as concurrent and interacting machines with actions and communications with their environment or user. Petri Nets are formal and graphical language, which is suitable for modeling systems with such properties like synchronisation, parallel operations, conflicts, or resource sharing.

Ordinary Petri Net is defined as a 4-tuple $N=(P, T, pre, post)$, where:

P – is a finite set of places, T – a finite set of transitions,

$pre: P \times T \rightarrow \{0,1\}$ is a preset function,

$post: P \times T \rightarrow \{0,1\}$ is a postset function.

Next, it is possible to define sets:

$\bullet t = \{p | pre(p,t) \neq 0\}$ is the set of pre-conditions of t

$t^\bullet = \{p | post(p,t) \neq 0\}$ is the set of post-conditions of t

$\bullet p = \{t | pre(p,t) \neq 0\}$ is the set of post-transitions of p

$p^\bullet = \{t | post(p,t) \neq 0\}$ is the set of pre-transitions of p

Very common, Petri Nets are to represent by oriented bipartite graphs with two types of vertices – places and transitions. For Petri Net depicted in *Fig 1* we have:

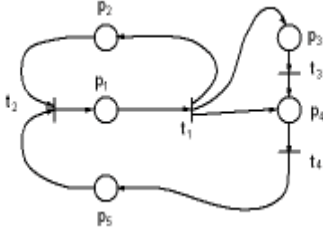


Fig.1 Example of Petri net

$P=\{p1, p2, p3, p4, p5\}$, $T=\{t1, t2, t3, t4\}$, and pre and $post$ functions are given in *Tab1*:

P	T	$Pre(P,T)$	P	T	$post(P,T)$
p_1	t_1	1	p_1	t_2	1
p_2	t_2	1	p_2	t_1	1
p_5	t_2	1	p_3	t_1	1
p_3	t_3	1	p_4	t_1	1
p_4	t_4	1	p_4	t_3	1
			p_5	t_4	1

Tab.1 $pre()$ and $post()$ functions

Marking of Petri Net $N=\{P, T, pre, post\}$ is defined to be a function $m:P \rightarrow \mathbb{N}$. We say that the condition represented by the place p holds, iff $m(p) \neq 0$. Transition $t \in T$ is enabled in a net with marking $m \in \mathbb{N}^{|P|}$, if the following condition is satisfied: $p \in \bullet t : m(p) \geq pre(p,t)$, where $|P|$ stands for cardinality of P . If the transition is enabled, it can fire. The result of executing a transition t is a new marking $m' \in \mathbb{N}^{|P|}$ and $m'(p)=m(p)-pre(p,t)+post(p,t)$.

Expressing power of Petri Nets for complex system description was proven in many cases, but there was stated it has some limitations [3]. For that purpose the original definition of Petri Nets was modified in many ways – Generalized Petri Nets (GPN), Capacity Bounded Petri Nets (CBPN), Predicate Transition Nets (PrTN), Environmental Relationship Nets (ER Nets) or Time ER Nets (TER Nets) [9,4,3].

3. PROCESS ALGEBRAS

Process terms are used as an abstract concurrent programming language and their algebraic structure emphasizes compositionality [6]. By *Process algebra* we mean a formal system for the study of concurrent communicating processes in an algebraic framework. R.Milner, with his Calculus of Communicating Systems (CCS), is considered to be the initiator of the field of process algebra. Another very important contribution to the field of the theory of concurrency is CSP by C.A.R.Hoare. J.Baeten is the founder of the PA in their contemporary setting.

BPA
$x+y=y+x$
$(x+y)+z=x+(y+z)$
$x+x=x$
$(x+y)z=xz+yz$
$(xy)z=x(yz)$
$\delta+x=x$
$\delta \cdot x = \delta$
PA
$x y=x y+y x$
$a x=ax$
$ax y=a(x y)$
$(x+y) z=x z+y z$
ACP
$x y=x y+y x+x y$
$ax b=(a b)x$
$A bx=(a b)x$
$ax by=(a b)(x y)$
$(x+y) z=x z+y z$
$a (y+z)=x y+x z$

Tab. 2 Axioms of ACP

The simplest process algebra is BPA (Basic Process Algebra), whose elements are BPA-expressions, which are built from *atomic actions* ($a, b, c \dots$) and basic constructors ($+, \cdot$), by the axioms (*Tab 2*). We consider idealization that atomic actions are events without positive duration in time and constructors ($+, \cdot$) are alternative and sequential composition respectively. The axiom system of BPA together with the axioms for δ is called BPA_δ . The process δ denotes 'deadlock'. PA (Process Algebra) includes, in addition to axioms of BPA, also axioms for *parallel composition*. If x, y are processes, their 'parallel composition' $x||y$ is the process that first chooses whether to do a step in x or y , and continues as the parallel composition of the remainders of x, y . The steps of x, y are interleaved, using an auxiliary operator $|$ (left merge). Parallel composition, as introduced in PA, by the operator $||$, does not involve communication in the process $x||y$. Some actions in one process may need an action in another process for an actual execution. ACP (Algebra of Communicating Processes) offers *merge with communication* which uses the same notation as *free merge* ($||$), because free merge is an instance of merge with communication (by choosing the communication function trivial $a|b = \delta$) [1].

4. B AMN LANGUAGE

The B AMN (Abstract Machine Notation) was developed by J.R.Abril in half of 80-ties in cooperation with research group of British Petroleum International [5]. Currently we can see increasing interest of B AMN using in both, the industry and universities. The method is considered to be a formal method with ability to express needs

of system in exact style. The B is more than just formal description. It allows to verify properties and internal consistency of system specification. B AMN supports specification, and all the refinement and design steps subsequent to specification. B is termed a wide-spectrum language or method, because it includes both executable descriptions and highly abstract mathematical descriptions. The means by which B AMN specifies state transitions are *generalised substitutions*. A generalised substitution is an abstract mathematical programming construct, corresponding to assignments to state variables, via the following operators (Tab 3):

Skip	No operation
S1 S2	do S1 or S2
P S	if P behave as S
P==>S	only if P holds, then do S
@v.S	do S for some v
S1;S2	do S1 then S2
S1 S2	do S1 and S2
WHILE E DO S	do S , while E holds
INVARIANT I	
VARIANT e END	

Tab. 3 B AMN operators

In Tab.3 *S*, *S1*, *S2* stands for generalised substitutions, *e* expression *I*, *E* and *P* predicates, and *v* a variable or list of variables. Only a subset of these constructs can be used at the various development stages [5]. The semantics of generalised substitutions is given by means of predicate transformers by E.W.Dijkstra [2]. The latter describe how the substitution transforms pre-states into post-states. The concept of *abstract machine* is close to object class or to ADA package. It encapsulates a set of mathematical items, constants, sets, variables and also operations on

MACHINE	N(p)
CONSTRAINTS	C
SETS	St
CONSTANTS	k
PROPERTIES	B
VARIABLES	v
DEFINITIONS	D
INVARIANT	I
ASSERTIONS	A
INITIALISATION	T
OPERATIONS	
y ← op(x) =	PRE P
	THEN S
...	END
END	

Tab. 4 Machine specification

these variables into a named module. The variables of a machine can be modified by the operations of that machine only. A general specification-level machine has a form given by Tab 4.

There are some additional clauses which may be added to a machine definition for the purpose that this machine uses various features of other machines. B AMN provides following mechanisms as optional clauses in machines, refinements or implementations: SEES, USES, INCLUDES, EXTENDS, REFINES, IMPORTS.

5. FDT INTERFACING

It was stated in the introduction to this paper that different FDTs provide different views of systems.

5.1 System representations

Representing a system in one of mentioned FDTs brings some advantages and also disadvantages. For example by representing a system by Petri Net, we can easily get invariants of the system or its graphical representation, which helps us to understand its behavior. On the other hand, the problem of decomposition of such system representation is not trivial [4]. Also missing is a way of abstraction from internal actions, which means processes are not treated as 'black boxes' in which only communication behavior is important.

Process terms, due to their algebraic structure, emphasize compositionality, that is, how more complex terms are composed from simpler ones. The expression power of process terms is stronger than power of Petri Nets languages. Thus Petri Nets can describe a subclass of the all terms only. It is impossible to find a semantics which represents every process term by a finite ordinary Petri Net. This is because process terms are Turing powerful, whereas finite nets are not [6].

B AMN provides refinement of specification and also composition of single abstract machines is possible in B AMN, but the machine designer must define invariants of the system.

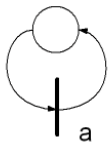
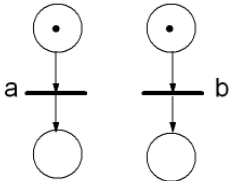
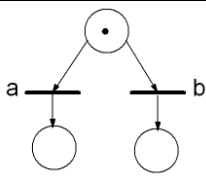
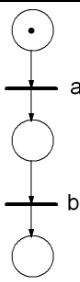
Representation of closed CCS terms by Petri Nets is used in [7] for fast deadlock detection in CCS systems. For such representation, the following assumptions must hold:

A1: Choices are guarded by input actions.

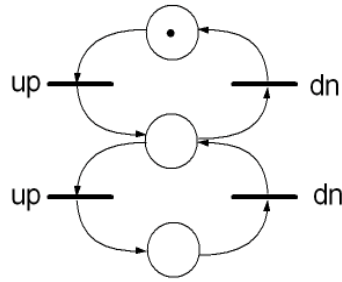
A2: Each pair of action – co-action names occurs only in two processes.

A3: Each component of the composition is a cyclic process.

Equivalence between handshakes of a closed CCS systems and firing of visible transitions of corresponding Petri Net makes it possible to analyze system for deadlocks by construction of reachability tree.

P1 $\mu X.a.X$	N1 	M1 WHILE true DO a END
P2 $A \parallel b$	N2 	M2 $A \parallel b$
P3 $a+b$	N3 	M3 CHOICE a OR b
P4 $a.b$	N4 	M4 $a;b$

Tab. 5 Elementary constructs

Process term: $C2=ZERO$ $ZERO=up.ONE$ $ONE=up.dn.ONE+dn.ZERO$ or $C2=\mu X.up. \mu Y(up.dn.Y + dn.X)$	Petri Net: 	B-AMN: MACHINE CNT2 CONSTANTS MAX, MIN PROPERTIES MAX:=2 & MIN:=0 VARIABLES VAL INVARIANT VAL: NAT & VAL: 0..2 INITIALISATION VAL:=0 OPERATIONS up = PRE VAL < MAX THEN VAL:= VAL+1 END; dn = PRE VAL > MIN THEN VAL:= VAL-1 END END
---	--	--

Tab. 6 An example

5.2 Direct and indirect interfacing

Methods of interfacing can be split down into two main classes: *direct* and *indirect interfacing*. The first class contains methods, where system description in one FDT is transferred into another FDT directly. That, in ideal case, yields a situation as described in *Fig 2a*, where two-way conversions among all the three mentioned formalisms are described.

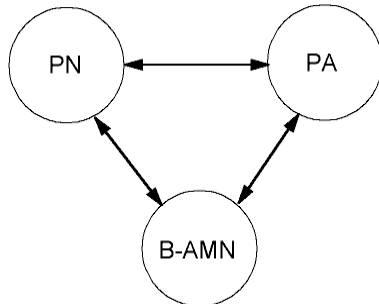


Fig. 2a Direct interfacing

In the second class, there are methods which use services of the other FDT to make the conversion possible. Example of this transformation method is on *Fig 2b*.

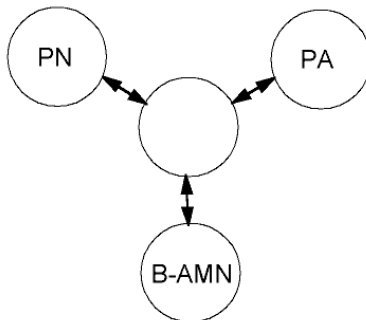


Fig. 2b Indirect interfacing

5.3 Interfacing semantics

Tab. 5 contains semantics of the elementary constructs.

6. EXAMPLE

As an example serves three state (0, 1, 2) counter described by means of all the three formalisms mentioned in paper (*Tab. 6*).

7. CONCLUSION

This paper presents the idea of looking at the system from three different points of view, namely by using three different FDTs: Petri Nets, Process Algebras, and B-AMN. Existing elementary constructions presented in this paper together with other properties of the selected FDTs support a belief that such interfaces can be constructed.

REFERENCES

- [1] Baeten, J.C.M.: Applications of process algebra, Cambridge University Press, 1990.
- [2] Dijkstra, E. W.: A Discipline of Programming. Prentice Hall, 1976.
- [3] Hudák, Š.: Rozšírenia Petriho Sietí, Habilitačná práca, VŠT EF Košice, pp.107, 1980.
- [4] Hudák, Š.: Reachability analysis of systems based on Petri Nets, Elfa Košice, pp.272, 1999.
- [5] Lano, K.: The B Language and Method, Springer – Verlag, 1996.
- [6] Olderog, E. R.: Nets, terms and formulas, Cambridge University Press, 1991.
- [7] Olszewski, J.: Fast Deadlock Detection in CCS Systems Using Petri Nets, Department of Software Engineering, School of Computer Science and Engineering, University of New South Wales, Sydney, Australia.
- [8] Olszewski, J.: Fast Deadlock Detection in CCS Systems Using Petri Nets, Department of Software Engineering, School of Computer Science and Engineering, University of New South Wales, Sydney, Australia.
- [9] Šimoňák, S.: Formálne metódy špecifikácie a analýzy časovo-kritických systémov, Písomná práca k dizeračnej skúške, TU FEI Košice, 2000.
- [10] Vokorokos, L.: Diagnosis of mechanical machineries using the parallel computer system. East-Slovak printers l.t.d. 2000. p. 152. ISBN 80-7099-619-6, Slovakia.

BIOGRAPHY

Štefan Hudák was born on August 25, 1939. He graduated from Moscow Institute of Energy, Faculty of Radioengineering in 1962. He obtained his PhD degree in Technical Cybernetics in 1977 from Slovak Technical University and DrSc degree in Theoretical Informatics in 1997 from T.Schevchenko University, Kiev, Ukraine. He is now Professor of Computing and Informatics at Faculty of Electrical Engineering and Informatics, Technical University in Košice. His interests are in automata theory, formal description techniques, Petri Nets and time-critical systems.

Slavomír Šimoňák was born on 23.9.1974. In 1998 he graduated (MSc.) at the Department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics at Technical University in Košice. He is now PhD candidate in Informatics at DCI FEI TU. His scientific research is focusing on formal methods for design and analysis of discrete systems. In addition, he also investigates problems related to time-critical systems.