Hybrid modeling and verification of Java based software

Konrad Kułakowski[†]

Institute of Automatics,
AGH University of Science and Technology
Al. Mickiewicza 30, 30-059 Cracow, Poland

Abstract. From the very beginning, notions such as bisimulation and formal methods like temporal logic HML or mu-Calculs were closely connected with process algebra CCS. Another formal method that is widely used for similar purposes is Petri nets formalism. The presented paper shows how the model given in the form of a Petri net could be transformed into an equivalent algebraic model. Some practical application of this method to the analysis of Java based software will be discussed.

1 Introduction

A typical software life-cycle proceeds from the phase of gathering requirements and forming specification to building and delivering a ready-to-use product. Usually it consists of several subsequent steps or phases in which a more and more detailed model of the system is built [1]. In this approach, it is useful to have a method of comparing the initial specification of the system with another (maybe more detailed) specification. One of the formal methods which can be used for this purpose is bisimulation [2].

The hybrid approach to software modeling and verification proposed in this paper is based on labelled Petri nets (LPN) and process algebra CCS. It shifts consideration about the model correctness from labelled Petri nets to the process algebra CCS. This allows for verification of the correctness of Petri nets by means of native algebraic mechanisms and notions. The shift is done by defining simple mapping between both formalisms. This approach does not need a compositional net semantics [3].
Correctness is understood as a relation of satisfying the specification by the given model [4]. Specification could be given in the form of Petri net, and in this case, we would say that the model satisfies the specification if both the LPN representing model and LPN representing specification are bisimilar. Bisimlarity checking is done in CCS. In order to do so, both Petri nets will be transformed into corresponding agent expressions. Specification could also be defined in the form of temporal logic formulas [5, 6]. The presented hybrid approach might also be useful for analysing Java software. In this paper, a sample program based on CyclicBarrier will be analysed in the context of the hybrid approach based on Petri nets and process algebra CCS.

[†]Author was partially supported by MNiSW under grant 4 T11C 035 24

2 Preliminary notions

Definicja 1. *Labelled Petri Net (Labelled Place-Transition net) is a tuple* $N = (P, T, F, K, W, L, M_0)$ *satisfying the following conditions:*

- P *is a finite, nonempty set of places.*
- T *is a finite, nonempty set of transitions.*
- $F \subseteq (P \times T) \cup (T \times P)$ *is a set of directed arcs of the net known as a flow relation.*
- $K{:}P \longrightarrow \mathbb{N} \cup \{\infty\}$ *is a function assigning a positive integer to every place. The value* $K(p)$*, where* $p \in P$*, indicates the maximum number of tokens that can be held by p. If* $p = \infty$*, this means that the maximum number of tokens is unlimited.*
- $W{:}F \longrightarrow \mathbb{N} \cup \{0\}$ *is a function denoting the weight of each arc.*
- $M_0{:}P \longrightarrow \mathbb{N} \cup \{0\}$ *is a function denoting an initial marking of the net.*
- $L : T \longrightarrow Act_{LPT}$ *is a labelling function which maps transitions to elements of a finite set of* Act_{LPT} *with a distinguished element* τ*.*

In the LPT net, an occurrence of transition $t_i \in$ T, and in consequence, marking the change from M to $M^{'}$, will be denoted $M \xrightarrow{t_i} M^{'}$, or $M[t_i\rangle M^{'}$. For every a $\in Act_{LPT}$,$M \xrightarrow{\mathrm{a}} M^{'}$ means that $M \xrightarrow{t_i} M^{'}$, such as $L(t_i) = a$. The net is called finite if $\forall p \in P\ K(p) \neq \infty$.

Definicja 2. *A reachability graph of the Petri net* $N = (P, T, F, K, W, L, M_0)$ *is a pair* $G = (V, A)$ *over* T*, where:*

- $V = [M_0\rangle$ *is a set of vertices,*
- T *– is a set of transitions ,*
- $A = \{(M, t, M') : M, M' \in [M_0\rangle \wedge M[t\rangle M'\}$ *is a set of arcs labelled by the names of transitions such that* $(M, t, M') \in A$ *if execution of* t *cause a change of marking from* M *to* M'*.*

The notion of bisimulation was originally defined for process algebra CCS [7], and it also could be easily defined for Petri nets [8]. In both formalisms, bisimulation is defined over the space of states of a system. In CCS there is a set of all possible derivatives of the given agent, whilst in the Petri net, this is a set of all possible markings of the net. Thus, whenever we say that two nets are bisimilar, we mean that their initial markings are bisimilar.

Definicja 3. *Let* LPT_1 *and* LPT_2 *be labelled Petri Nets. A strong bisimulation is the relation* B *between the markings of* LPT_1 *and* LPT_2*, such that for all* $(M_1, M_2) \in B$ *and for all* $a \in Act_{LPT}$*:*

- *if* $M_1 \xrightarrow{a} M_1'$ *then* $M_2 \xrightarrow{a} M_2'$ *for some* M_2' *such that* $(M_1', M_2') \in B$ *and*
- *if* $M_2 \xrightarrow{a} M_2'$ *then* $M_1 \xrightarrow{a} M_1'$ *for some* M_1' *such that* $(M_1', M_2') \in B$

The simlar definition could be given for a weak bisimulation. We would say that M_1 is strongly bisimilar to M_2 and donote it $M_1 \sim M_2$, and M_1 is weakly bisimilar (observable equivalent) to M_2 and denote it $M_1 \approx M_2$. Two Petri nets are in strong (weak) bisimulation if their initial markings are in strong (weak) bisimulation.

Notions like strong and weak bisimulation were originally defined in the context of process algebra CCS. Appropriate definitions, as well as a detailed description of the formalism, might be found in various positions [9, 10].

3 Transformation of Labelled Petri Nets to CCS

Let N be a finite labelled Petri net and $G_N = (V, A)$ be its reachability graph. Because N is finite, G_N is also finite; i.e. both sets V and A are finite. Let us enumerate the set of vertices as follows: $V = (M_0, \ldots, M_r)$. Let $O(M)$ be a set of all vertices from V that can be directly reached from M; i.e.:

$$O(M) = \{M_i \in V \,:\, M[t\rangle M_i \in A\}$$

For every vertex $M \in V$ let $\Phi(M)$ be the agent in the form:

$$\Phi(M) = \Sigma_{M_i \in O(M)} L(t_i).\Phi(M_i)$$

where t_i is a transition such that $M[t_i\rangle M_i \in A$.

We would say that the symbol Φ denotes transformation of the given Petri net N, with the initial marking M_0, to the corresponding CCS agent $\Phi(M_0)$ according to the scheme presented above.

Let us consider the net N_1 (Fig.1).

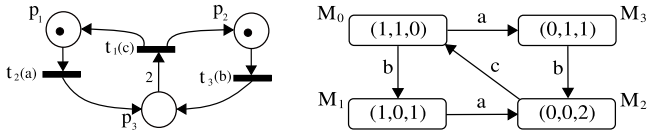

Fig. 1. Petri net $N_1$and its reachability graph

Transitions $t_1, t_2, t_3,$ of N_1 are labelled correspondingly: $L(t_1) = c$, $L(t_2) = a$, $L(t_3) = b$. The reachability graph of N_1 contains four vertices that denote four possible markings: $M_0, \ldots, M_3$ (Fig.1). According to the assumed transformation algorithm, the algebraic equivalent of N_1 is the CCS agent $\Phi(M_0) \stackrel{df}{=} \Phi(N_1)$ defined as follows:
$\Phi(M_0) \stackrel{df}{=} \text{a}.\Phi(M_3) + \text{b}.\Phi(M_1)$, $\Phi(M_1) \stackrel{df}{=} \text{a}.\Phi(M_2)$, $\Phi(M_3) \stackrel{df}{=} \text{b}.\Phi(M_2)$, $\Phi(M_2) \stackrel{df}{=} \text{c}.\Phi(M_0)$.
Transformation Φ is consistent with the relations of strong and weak bisimulations, which means that if two Petri nets are bisimilar, their algebraic equivalents are also bisimilar.

Theorem 1. *For two finite labelled Petri nets LPN_1, LPN_2 if $LPN_1 \sim LPN_2$, thus also $\Phi(M_0^{(1)}) \sim \Phi(M_0^{(2)})$ where $M_0^{(1)}$ and $M_0^{(2)}$ are initial markings of LPN_1 and LPN_2.*

Proof. Let us assume $LPN_1 \sim LPN_2$. Satisfying the equivalence $\Phi(M_0^{(1)}) \sim \Phi(M_0^{(2)})$ requires:

- $\forall a : \Phi(M_0^{(1)}) \xrightarrow{\text{a}} A_1^{(1)}$ exists $A_1^{(2)}$, such that $\Phi(M_0^{(2)}) \xrightarrow{\text{a}} A_1^{(2)}$ and $A_1^{(1)} \sim A_1^{(2)}$
- $\forall b : \Phi(M_0^{(2)}) \xrightarrow{\text{b}} B_1^{(2)}$ exists $B_1^{(1)}$, such that $\Phi(M_0^{(1)}) \xrightarrow{\text{b}} B_1^{(1)}$ and $B_1^{(1)} \sim B_1^{(2)}$

If for some a exists transition $\Phi(M_0^{(1)}) \xrightarrow{a} A_1^{(1)}$, it means that by knowing the construction of the agent $\Phi(M_0^{(1)})$, we also know that $M_0^{(1)}[t_i\rangle M_1^{(1)}$, where $L(t_i) = a$ and $\Phi(M_1^{(1)}) = A_1^{(1)}$. In other words, $M_0^{(1)} \xrightarrow{a} M_1^{(1)}$. Because $LPN_1 \sim LPN_2$ there is $M_0^{(1)} \sim M_0^{(2)}$. Thus, on the basis of these two facts, there is $M_2^{(2)}$ such that $M_0^{(2)} \xrightarrow{a} M_1^{(2)}$. Let us denote $A_1^{(2)} = \Phi(M_1^{(2)})$. Considering the construction of $\Phi(M_0^{(2)})$, we know that there is transition $\Phi(M_0^{(2)}) \xrightarrow{a} \Phi(M_1^{(2)})$.

The question arise whether $A_1^{(1)} \sim A_1^{(2)}$, i.e. if $\Phi(M_1^{(1)}) \sim \Phi(M_1^{(2)})$ (and similarly, if $B_1^{(2)} \sim B_1^{(1)}$). Let us note that $M_1^{(1)} \sim M_1^{(2)}$ (as a consequence of $LPN_1 \sim LPN_2$). In other words, using the same reasoning as presented above, but this time applied to the nets $LPN_1(M_1^{(1)})$ i $LPN_2(M_1^{(2)})$ – i.e. to net LPN_1 with marking $M_1^{(1)}$ and net LPN_2 with marking $M_1^{(2)}$, we may show that $\Phi(M_1^{(1)}) \sim \Phi(M_1^{(2)})$, which is of course true if and only if $A_2^{(1)} \sim A_2^{(2)}$ and $B_2^{(2)} \sim B_1^{(2)}$,... and so on. By repeating this operation, we prove equivalences: $A_3^{(1)} \sim A_3^{(2)}$,$A_4^{(1)} \sim A_4^{(2)}$,... and correspondingly $B_3^{(2)} \sim B_3^{(1)}$,$B_4^{(2)} \sim B_4^{(1)}$,... etc.

After the r-th iteration, we have to prove that $\Phi(M_r^{(1)}) \sim \Phi(M_r^{(2)})$. These two agents satisfy the relation $\sim$ if $A_{r+1}^{(1)} \sim A_{r+1}^{(2)}$ and $B_{r+1}^{(1)} \sim B_{r+1}^{(2)}$. If in steps l_1, l_2, such that $0 < l_1 < r$, $0 < l_2 < r$, the equivalences $A_{l_1}^{(1)} \sim A_{l_1}^{(2)}$ and $B_{l_2}^{(1)} \sim B_{l_2}^{(2)}$ have been proven, where $A_{r+1}^{(1)} = A_{l_1}^{(1)}$, $A_{r+1}^{(2)} = A_{l_1}^{(2)}$, $B_{r+1}^{(1)} = B_{l_2}^{(1)}$ and $B_{r+1}^{(2)} = B_{l_2}^{(2)}$, we may stop our reasoning at this point. If not, this situation must happen at the latest for $r = pq$, where p is the number of vertices in the reachability graph G_{LPN_1} and q - is the number of vertices in the reachability graph G_{LPN_2}. This is because, if we reach r-th step of out reasoning $(r = pq)$, this means that for every pair of agents $\Phi(M_i^{(1)})$, $\Phi(M_j^{(2)})$, where $M_i^{(1)}$, $M_j^{(2)} \in V_{LPN_1}, V_{LPN_2}$, it is true that $\Phi(M_i^{(1)}) \sim \Phi(M_j^{(2)})$.

According to the presented reasoning scheme, the following theorems can also be proved.

Theorem 2. *For two finite, labelled Petri nets LPN_1 and LPN_2, if $LPN_1 \approx LPN_2$, thus also $\Phi(M_0^{(1)}) \approx \Phi(M_0^{(2)})$, where $M_0^{(1)}$ and $M_0^{(2)}$ are initial markings of LPN_1 and LPN_2.*

Theorem 3. *For every two finite labelled Petri nets LPN_1,LPN_2 if $\Phi(M_0^{(1)})$ op $\Phi(M_0^{(2)})$, then LPN_1 op LPN_2, where op $\in \{\sim, \approx\}$, and $M_0^{(1)}$,$M_0^{(2)}$ are initial markings of LPN_1 and LPN_2.*

4 Hybrid modeling – case study

Petri Nets are very often used in the modeling of reactive systems behaviour. The Petri Net could act both as a specification or as a design. When specification is considered, usually a small Petri Net is used. It defines activities which determine basic system functionality.

The ideas of strong and weak bisimulation could be implemented into the Petri Net formalism. Thanks to the presented transformation, analysis of bisimilarities on the basis of the net's algebraic representation is possible. It also allows us to: model comparisons

in terms of bisimulation property and project validation against specification which might be written in the form of a set of temporal formulas.

Different model comparisons

Let us consider a model of a system that consists of several different processes responsible for data processing and one process responsible for printing the result. In our example, data processing consists of two separate activities denoted correspondingly by a and b. Data processing is complete if both activities are done. A result printing is represented by activity c. The order of activities, a and b is not important. Both sequences a, b and b, a are allowed. The only requirement is that a and b must take place before c. In our mini-system, these activities will be modeled by actions; i.e. the activity is complete if an action occurs.

Assuming that actions a and b occur independently (e.g. in a separate processes), a situation like the one described above is modeled by the net N_1 shown in the figure 1. In general, N_1 depicts the system in which two different processes do action a and b, and next they wait until c is done.

This very popular synchronisation model was reflected in modern Java in the form of a *CyclicBarrier* class [11]. It provides a convenient synchronisation aid that allows one thread to wait until other threads complete their tasks. Of course the requirement that before c both actions a and b must occur might be fulfilled differently. Let us consider a *NaivyApp* simple sequential application which may perform two possible scenarios: a, b, c and b, a, c. In deed, these application also meets the requirements that both actions a and b must occur before c.

The behaviour of *NaivyApp* is modeled by the net N_2 shown in the figure 2. This net corresponds to a simple sequential program that performs the actions a, b, c or b, a, c repeatedly in turn.

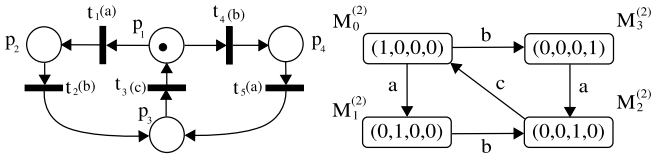

Fig. 2. Net N_2 and its reachability graph

Because both nets N_1 and N_2 represent the programs that satisfy our informal specification, the question comes up whether their behaviours are the same. In order to answer this question, the reachability graph of N_2 is built (figure 2). It enable us to construct agent $\Phi(N_2)$, which is the algebraic representation of N_2.

$\Phi(M_0^{(2)}) \stackrel{df}{=} \text{a}.\Phi(M_1^{(2)}) + \text{b}.\Phi(M_3^{(2)})$, $\Phi(M_1^{(2)}) \stackrel{df}{=} \text{b}.\Phi(M_2^{(2)})$, $\Phi(M_2^{(2)}) \stackrel{df}{=} \text{c}.\Phi(M_0^{(2)})$, $\Phi(M_3^{(2)}) \stackrel{df}{=} \text{a}.\Phi(M_2^{(2)})$

According to the theorem 3, the nets N_1 and N_2 are bisimilar if agents $\Phi(M_0)$ and $\Phi(M_0^{(2)})$ are bisimilar. A quick automatic check proves that $\Phi(M_0) \sim \Phi(M_0^{(2)})$ [12], and thereby it will be shown that $N_1 \sim N_2$.

5 Summary

For several years, a significant increase of demand for reliable multi-threaded software can be be observed. As a result, libraries supporting the building of concurrent applications for many programming languages are available [13, 11] (e.g. a recent version of Java incorporates the new *java.util.concurrent* package). This trend also make stronger a need for the creation of convenient and versatile formal methods that support specification and design of concurrent software.

The hybrid modeling technique presented above helps to achieve this goal. It facilitates using bisimulation in the context of models given in the form of Petri nets. Because of transforming a net to an appropriate CCS agent, it is possible to proceed with further analysis in well defined algebraic formalism, including suitable tools such as CWB [12].

Defining algorithms and methods that shorten the distance between formal methods such as Petri nets or CCS algebra and the Java language will pose a challenge to author in the near future.

References

1. McConnell, S.: Code Complete. Microsoft Press, Redmond, WA. (1993)
2. Bruns, G.: Distributed Systems Analysis. Prentice Hall (1997)
3. Goltz, U.: CCS and Petri Nets. In: Semantics of Systems of Concurrent Processes, Berlin - Heidelberg - New York, Springer (1990) 334–357
4. Kułakowski, K.: Konstrukcja i Analiza Oprogramowania Sterowników Wspomagana Metodami Formalnymi. PhD thesis, Akademia Górniczo-Hutnicza (2003)
5. Groote, J., Voorhoeve, M.: Operational semantics for petri net components (2003)
6. Fencott, C.: Formal Methods for Concurrency. International Thomson Computer Press, Boston, MA, USA (1995)
7. Milner, R.: Communication and Concurrency. Prentice-Hall (1989)
8. Jancar, P., Esparza, J.: Deciding finiteness of Petri Nets up to Bisimulation. Lecture Notes in Computer Science **1099** (1996)
9. Milner, R.: A Calculus of Communicating Systems. Volume 92 of LNCS. Springer-Verlag (1980)
10. Fidge, C.: A comparative introduction to CSP, CCS and LOTOS. Technical report (1994)
11. Lea, D.: Concurrent Programming in Java. The Java Series. Addison-Wesley, Reading, MA (1997)
12. Moller, F., Stevens, P.: Edinburgh Concurrency Workbench user manual (version 7.1). (Available from http://homepages.inf.ed.ac.uk/perdita/cwb/)
13. Niño, J., Hosch, F.A.: An Introduction to Programming and Object Oriented Design Using Java 1.5. Second edn. Wiley, Hoboken, NJ (2005) With CD-ROM.