4-th Int. Conf. on Dependability of Computer Systems, Brunow, Poland, June 30 - July 2, 2009, pp.123-132.

Siphon–based verification of component compatibility

W.M. Zuberek

Department of Computer Science and *Department of Applied Informatics*
Memorial University *University of Life Sciences*
St.John's, Canada A1B 3X5 *02-787 Warsaw, Poland*

Abstract

In component–based systems, two interacting components are compatible if any sequence of services requested by one component can be provided by the other. This concept of compatibility can be easily extended to a set of interacting components. Checking the compatibility of interacting components is essential for any dependable software system. Recently, an approach to verification of component compatibility has been proposed in which the behavior of individual components (at component interfaces) was modeled by labeled Petri nets. Moreover, the composition of interacting components was designed in such a way that all component incompatibilities were manisfested by deadlocks in the composed model. Consequently, the verification of component compatibility is performed by deadlock analysis of the composed net model. One of techniques for deadlock analysis is based on net structures called siphons. Siphon–based verification of component compatibility is the subject of this paper.

1. Introduction

In complex software architectures, the need to assess the compatibility and interoperability of the individual software components is becoming critical during the integration phase of the software production process. While manual and ad hoc strategies toward component integration have met with some success in the past, such techniques do not lend themselves well to automation, and a more formal approach toward the compatibility and interoperability assessment is needed. Such a formal approach would permit an assessment based on automated techniques and would also help promote the reuse of existing software components.

In component–based systems, two interacting components are compatible if any sequence of services requested by one component can be provided by the other. This concept of compatibility can be easily extended to a set of interacting components. Recently, an approach to verification of component compatibility has been proposed in which the behavior of individual components (at component interfaces) was modeled by labeled Petri nets [5]. Moreover, the composition of interacting components was designed in such a way that all component incompatibilities were manisfested by deadlocks in the composed model. Consequently, the verification of component compatibility is performed by deadlock analysis of the composed net model.

Petri nets [9, 8] are formal models of systems which exhibit concurrency or synchronizations of activities. Examples of such systems include multiprocessor systems, distributed databases, manufacturing and transportation systems, and many others. One important aspect of such systems is the existence (or absence) of deadlocks, i.e., a possibility of reaching a state in which no activity can be continued. Absence of deadlocks is critical in

systems which are expected to operate in a continuous way, such as life-support systems, supervisory control systems (e.g., in a nuclear plant), transportation control systems, and so on. A systematic and efficient method of deadlock detection is of primary importance for such systems [1].

Known methods of deadlock detection in Petri net models include reachability analysis which systematically explores all possible states that can be reached from the initial state(s) looking for deadlock states, and structural analysis which determines deadlock existence (or absence) analysing the structure of Petri net models. Reachability analysis is quite straightforward, but can be used only for models with finite and reasonably small state spaces [8]. Structural analysis uses *siphons* for deadlock detection [3, 6, 10].

Siphon–based analysis does not depend upon the number of reachable markings and can be used for deadlock detection in nets with infinite state spaces, but it can easily become quite inefficient if the number of siphons is large (for some net models, the number of siphons grows exponentially with the net size). It appears, however, that instead of analyzing all siphons, only a small set of minimal siphons provides the same information about the absence (or existence) of deadlocks. Moreover, the large number of siphons can be significantly reduced by simple reductions of net models which do not affect the existence (or absence) of deadlocks. This paper formally introduces the concept of equivalent siphons and proposes two types of net transformations which reduce equivalent siphons making the siphon-based deadlock detection quite attractive from a practical point of view. Large-scale applications of the proposed approach can be found in [4].

The objective of this paper is to discuss the siphon-based verification of component compatibility, including the algorithmic aspects of finding the essential siphons in Petri nets, and use the essential siphons for effective deadlock analysis of net models.

Section 2 recalls basic concepts of Petri nets. Section 3 introduces siphons and shows how to find the set of minimal siphons. Basis siphons are briefly discussed in Section 4 and siphon equivalence as well as the reduction of equivalent siphons through simple net transformations in Section 5. Section 6 outlines a two-level algorithm for siphon-based deadlock detection, and Section 7 concludes the paper.

2. Basic concepts of Petri nets

A Petri net (also known as *net structure*) $\mathcal{N}$ is a triple $\mathcal{N} = (P, T, A)$ where P is a finite set of places (which represent conditions), T is a finite set of transitions (which represent events), $P \cap T = \emptyset$, A is a set of directed arcs which connect places with transitions and transitions with places, $A \subseteq P \times T \cup T \times P$, also called the *flow relation* or *causality relation* (and sometimes represented in two parts, a subset of $P \times T$ and a subset of $T \times P$).

For each transition $t \in T$, and each place $p \in P$, the input and output sets are defined as follows:

$$Inp(t) = \{p \in P \mid (p,t) \in A\}, \quad Inp(p) = \{t \in T \mid (t,p) \in A\},$$
$$Out(t) = \{p \in P \mid (t,p) \in A\}, \quad Out(p) = \{t \in T \mid (p,t) \in A\}.$$

The dynamic behavior of nets is represented by markings, which describe the distribution of the so called tokens over the places of a net. Under certain conditions these token distributions can change, transforming one marking into another.

A *marked Petri net* $\mathcal{M}$ is a pair $\mathcal{M} = (\mathcal{N}, m_0)$, where $\mathcal{N}$ is a net structure, $\mathcal{N} = (P, T, A)$, and m_0 is the *initial marking function*, $m_0 : P \to \{0, 1, ...\}$, which assigns a nonnegative number of *tokens* to each place of the net. Marked nets are also equivalently defined as $\mathcal{M} = (P, T, A, m_0)$.

Let any mapping $m : P \to \{0, 1, ...\}$ be called a *marking function* in $\mathcal{N} = (P, T, A)$.

In marked nets, a condition represented by a place p is satisfied at a marking m if $m(p) > 0$, and then p is said to be *marked* by m. If all input places of a transition t are marked, t is *enabled*. The set of all transitions enabled by a marking m is denoted $E(m)$.

If all (input) conditions of an event are satisfied (i.e., if the transition representing this event is enabled), the event can *occur*. An occurrence of an event removes (simultaneously) a single token from all input places of the transition representing this event, and (also simultaneously) adds a single token to all output places of this transition. This creates a new marking function. An occurrence of an event represented by t (i.e., t's firing) is thus a transformation of the (current) marking function m into a new marking function m' which is *directly reachable* from m by firing t, $m \stackrel{t}{\mapsto} m'$.

A marking m_j is *generally reachable* (or just reachable) from a marking m_i in $\mathcal{M}$, $m_i \stackrel{*}{\mapsto} m_j$, if m_j is reachable from m_i by a sequence of directly reachable markings (the general reachability relation is the reflexive transitive closure of the direct reachability relation).

The *set of reachable markings*, $\mathbf{M}(\mathcal{M})$, of a marked net $\mathcal{M}$ is the set of all markings which are (generally) reachable from the initial marking m_0:

$$\mathbf{M}(\mathcal{M}) \;=\; \{m \;\mid\; m_0 \stackrel{*}{\mapsto} m\}.$$

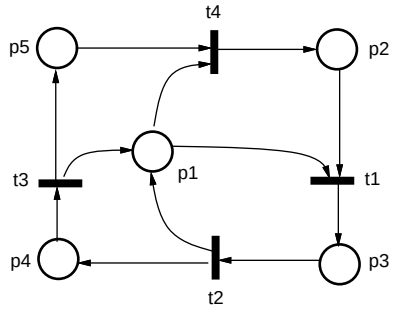
The set of reachable markings can be finite or infinite; if it is finite, the net $\mathcal{M}$ is *bounded*, otherwise the net is *unbounded*:

$$\mathcal{M} \text{ is bounded} \Leftrightarrow \exists\, k > 0\ \forall\, m \in \mathbf{M}(\mathcal{M})\ \forall\, p \in P : m(p) \leq k.$$

Each net $\mathcal{N} = (P, T, A)$ can conveniently be represented by a connectivity (or incidence) matrix $\mathbf{C} : P \times T \to \{-1, 0, +1\}$ in which places correspond to rows, transitions to columns, and the entries $\mathbf{C}[p, t]$, $p \in P$, $t \in T$, represent the directed arcs:

$$\mathbf{C}[p,t] = \begin{cases} -1, & \text{if } (p,t) \in A \wedge (t,p) \notin A, \\ +1, & \text{if } (t,p) \in A \wedge (p,t) \notin A, \\ 0, & \text{otherwise.} \end{cases}$$

For example, Fig.1 shows an example net [3] and its connectivity matrix.



$\mathbf{C}$	t_1	t_2	t_3	t_4
p_1	-1	$+1$	$+1$	-1
p_2	-1	0	0	$+1$
p_3	$+1$	-1	0	0
p_4	0	$+1$	-1	0
p_5	0	0	$+1$	-1

Fig.1. An example Petri net and its connectivity matrix.

One of the most important properties of many concurrent systems is the absence of *deadlocks*; intuitively, a deadlock is a configuration in which the system cannot continue its operation, it becomes *dead*.

A marking m in net $\mathcal{N}$ is *dead* if no transition is enabled by m, i.e., if $E(m) = \emptyset$. A marked net $\mathcal{M}$ contains a *deadlock* if its set of reachable markings contains a dead marking:

$$\mathcal{M} \text{ contains a deadlock } \Leftrightarrow\ \exists\, m \in \mathbf{M}(\mathcal{M}) : E(m) = \emptyset.$$

Two basic methods of deadlock detection in Petri nets are reachability analysis and structural analysis. If the net is bounded, reachability analysis can be used for exhaustive exploration of the marking space [8, 9]. If a net is unbounded, or if the space of reachable markings is finite but unreasonably large, the structural approach can be used.

3. Siphons and minimal siphons

The structural approach to deadlock detection is based on siphons [7, 8] (in early publications siphons were called structural deadlocks). Siphons are defined as such subsets of places $P_i \subseteq P$, for which:

$$Inp(P_i) \subseteq Out(P_i),$$

where $Inp(P_i)$ and $Out(P_i)$ are the input and output sets of P_i:

$$Inp(P_i) = \bigcup_{p \in P_i} Inp(p), \quad Out(P_i) = \bigcup_{p \in P_i} Out(p).$$

The characteristic property of siphons is that once a siphon P_i becomes unmarked under a marking m (i.e., m assigns zero tokens to all places of P_i), P_i remains unmarked for all markings reachable from m.

Each siphon P_i can be represented by its characteristic n-element vector V_{P_i} (n is the number of transitions), $V_{P_i} : T \to \{0, -1, +1, \star\}$, where:

$$\forall t \in T \ : V_{P_i}(t) = \begin{cases} 0 & \text{if } t \notin \mathrm{Inp}(P_i) \cup \mathrm{Out}(P_i); \\ -1 & \text{if } t \in \mathrm{Out}(P_i) - \mathrm{Inp}(P_i); \\ +1 & \text{if } t \in \mathrm{Inp}(P_i) - \mathrm{Out}(P_i); \\ \star & \text{if } t \in \mathrm{Inp}(P_i) \cap \mathrm{Out}(P_i). \end{cases}$$

It should be observed that if P_i is a siphon, its characteristic vector does not contain any "+1" elements.

The characteristic vector of a set of places P_i can be obtained by a "merge" operation, denoted $\oplus$, performed on the rows of the connectivity matrix that correspond to places in P_i; the definition of the binary operation $\oplus$ is based on the operations on the input and output sets of places (for simplicity it is assumed that the nets do not contain self-loops *(p,t), (t,p)*):

$\oplus$	0	-1	$+1$	$\star$
0	0	-1	$+1$	$\star$
-1	-1	-1	$\star$	$\star$
$+1$	$+1$	$\star$	$+1$	$\star$
$\star$	$\star$	$\star$	$\star$	$\star$

Also, if P_i and P_j are two siphons in a net $\mathcal{N}$, then $P_i \cup P_j$ is also a siphon in $\mathcal{N}$. A minimal siphon is defined as a siphon which does not contain any other siphon [11] [12]. Quite often the set of all siphons can be rather large, but the set of minimal siphons contains just a few siphons.

It can be shown [3] that in a deadlocked net, all unmarked places constitute a siphon. Therefore the siphon-based approach to deadlock detection systematically checks if the net contains a proper siphon (a siphon is proper if its input set is a (proper) subset of the output set) that can become unmarked by some firing sequence, and if such a siphon is

identified, the initial marking is modified by the firing sequence, and the check continues for the remaining (marked, proper) siphons until a deadlock is identified, or until no further progress can be done. Linear programming can be used to find the firing sequence that minimizes the number of tokens in each proper siphon of the analyzed net (if such a sequence exists) [3, 10].

For siphon–based deadlock analysis there is no need to check all the siphons of a net. Instead, the set of minimal siphons can be used because if any siphon becomes unmarked during the analysis, than (at least) one of minimal siphons must also become unmarked.

Finding the set of minimal siphons can follow the observation that each siphon in a net $\mathcal{N}$ is such a collection of places that the merge of the rows corresponding to these places cannot contain any element "+1". So, starting from a place p with its characteristic vector $C[p, *]$, i.e., the row of the connectivity matrix which corresponds to p, in consecutive steps each element "+1" is eliminated by including in the siphon a place with "−1" on the same position (and then the merging operation combines "+1" with "−1" into "$\star$", possibly introducing one or more new "+1" elements). For the net in Fig.1, the process of finding siphons which include place p_3 is illustrated in Fig.2 in a form of a "decision tree" in which siphons correspond to paths from the root to terminal nodes.

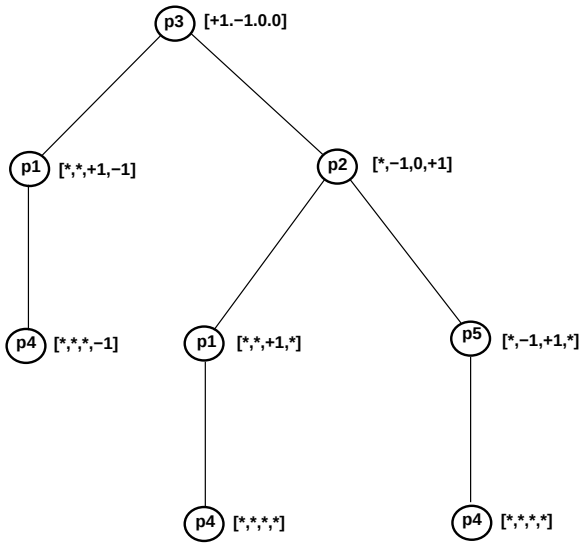


Fig.2. Finding siphons which include p_3.

In the first step, p_3 is included in the siphon which – at this stage – has the characteristic vector $[+1, -1, 0, 0]$. The only element "+1" (representing the arc from t_1 to p_3) can be eliminated by either adding p_1 to the siphon, or by adding p_2. In the first case the characteristic vector becomes $[\star, \star, +1, -1]$ (obtained by merging $[+1, -1, 0, 0]$ with $[-1, +1, +1, -1]$), in the second case the characteristic vector becomes $[\star, -1, 0, +1]$. For the set $\{p_1, p_3\}$, the "+1" element in the characteristic vector requires adding p_4 to the siphon, and then the characteristic vector becomes $[\star, \star, \star, -1]$, which means that the set $\{p_1, p_3, p_4\}$ is a siphon, and a minimal one (it does not contain a simpler siphon). Similarly, $\{p_1, p_2, p_3, p_4\}$ is another siphon, as is $\{p_2, p_3, p_4, p_5\}$.

A systematic method of finding all minimal siphons simply repeats the process shown in Fig.2 for consecutive places. In the following algorithm *Siphons* is the set of minimal siphons, S is a siphon, n is the number of places, m is the number of transitions, and $\mathbf{C}$ is the connectivity matrix:

```
Siphons := {};
for i := 1 to n do
    search(C[i, *], {i})
od;
```

where a recursive procedure *search* is as follows (*done* is a local variable of *search*, and the parameter passing mechanism is assumed to be BY VALUE):

```
proc search(V, S);
var done := true;
begin
    for j := 1 to m do
        if V[j] = +1 then
            done := false;
            for k := 1 to n do
                if C[k, j] = −1 then
                    search(V ⊕ C[k, *], S ∪ {k})
                fi
            od
        fi
    od;
    if done then checksiphons(S) fi
end
```

Procedure *checksiphons* adds the new siphon S to *Siphons* only is there is no simpler siphon there; it also deletes all siphons which are supersets of S:

```
proc checksiphons(S);
begin
    for each R in Siphons do
        if R ⊆ S then return fi;
        if S ⊂ R then Siphons := Siphons − {R} fi
    od;
    Siphons := Siphons ∪ {S}
end
```

4. Basis siphons

Minimal siphons rarely determine the deadlocks in Petri nets. Therefore another type of siphon is needed which is known as a basis siphon [2]. Basis siphons are siphons from which all other siphons can be obtained by the union operation. Minimal siphons are basis siphons, but usually some basis siphons are not minimal - the number of basis siphons is typically larger than the number of minimal siphons.

Basis siphons can be determined by a procedure very similar to the one presented in the previous section; the only difference is that the set of minimal siphons needs to be determined for each place independently and then combined into a set *BSiphons*:

```
BSiphons := {};
for i := 1 to n do
    Siphons := {};
    search(C[i, *], {i});
    BSiphons := BSiphons ∪ Siphons
od;
```

5. Equivalent siphons

For siphon–based deadlock analysis, the number of siphons can be further reduced by eliminating equivalent siphons.

Two siphons S_i and S_j are equivalent, $S_i \sim S_j$, if for each reachable marking either both siphons are marked or both are unmarked:

$$S_i \sim S_j \;\Leftrightarrow\; \forall m \in \mathbf{M}(\mathcal{M}) : (\mathrm{mark}(S_i, m) = 0 \Leftrightarrow \mathrm{mark}(S_j, m) = 0)$$

where $\mathrm{mark}(S_i, m) = \Sigma_{p \in S_i} m(p)$.

Since the siphon–based deadlock detection looks for unmarked siphons, each class of equivalent siphons can be reduced to just a single siphon, simplifying the detection approach.

Equivalent siphons can be eliminated using some structural properties of nets.

A simple path in a net $\mathcal{N}$ is a sequence of transitions and places $t_{i_0} p_{i_1} t_{i_1} p_{i_2} ... p_{i_k} t_{i_k}$ which are not connected to other places and transitions:

$$(\forall 1 \le j \le k \;:\; Inp(p_{i_j}) = \{t_{i_{j-1}}\} \wedge Out(p_{i_j}) = \{t_{i_j}\}) \wedge (\forall 1 \le j < k \;:\; Inp(t_{i_j}) = \{p_{i_j}\} \wedge Out(t_{i_j}) = \{p_{i_{j+1}}\}).$$

A simple path from t_i to t_j is denoted $path(t_i, t_j)$.

There are two classes of paths that can be used for siphon reduction, parallel paths and alternate paths.

Parallel paths are simple paths which connect the same transitions, as shown in Fig.3.

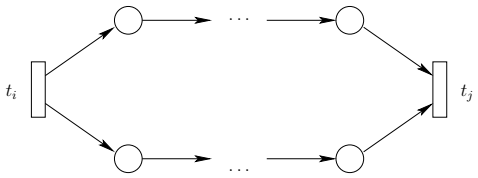


Fig.3. Parallel paths in a Petri net.

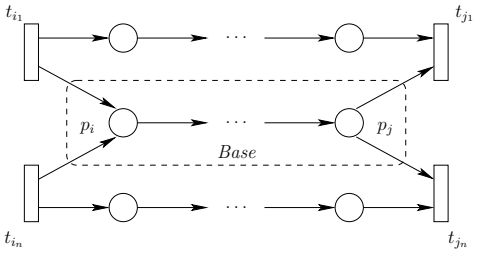


Fig.4. Alternate paths in a Petri net.

It can be easily shown that for equally marked parallel paths π_1 and π_2 (i.e., paths π_1 and π_2 which contain the same numbers of token), if the places of π_1 constitute a subset of a siphon S_i, then there exists another siphon S_j which includes places of π_2, and $S_i \sim S_2$. So, one of these parallel paths can be removed from the net eliminating one or more equivalent siphons.

An alternate path is a collection of disjoint simple paths, $path(t_{i_1}, t_{j_1})$,..., $path(t_{i_n}, t_{j_n})$, $t_{i_\ell} \neq t_{i_k}$, $t_{j_\ell} \neq t_{j_k}$, for $1 \le \ell < k \le n$, with an additional simple path (called the base) $path(p_i, p_j)$ connected to all transitions t_{i_ℓ} and t_{j_ℓ}, $(t_{i_\ell}, p_i) \in A$, $(p_j, t_{j_\ell}) \in A$, $1 \le \ell \le n$, as shown in Fig.4.

It can be shown [4] that for alternate paths π_1, ..., π_n with base π_0, if places of π_1 are included in a siphon S_i, then there exists another siphon S_j which includes places of π_0, and $S_i \sim S_j$. Consequently, the base π_0 can be removed from the net without affecting the existence (or absence) of deadlocks.

The reduction of equivalent siphons can be performed in such a way that first all parallel and alternate paths are removed from a net model, and then siphons are determined for the simplified net. These siphons can then be used for deadlock detection in the original net or in the simplified net.

6. Siphon-based deadlock detection

The deadlock detection process is performed in two stages; first the (hopefully small) set of minimal siphons is used for checking the deadlock by finding firing sequences which remove tokens from as many siphons as possible. If this does not result in a deadlock, and no further progress can be made using minimal siphons, a switch is made to use basis siphons (since each deadlock corresponds to one or a union of several unmarked basis siphons).

The deadlock detection procedure [4] is implemented as a recursive boolean procedure which is invoked as "$deadlock(m_0, \mathcal{S}_M, \mathcal{S}_B)$" where m_0 is the initial marking function, $\mathcal{S}_M$ is the (reduced) set of minimal siphons of the analyzed net and $\mathcal{S}_B$ is the (reduced) set of its basis siphons:

```
function deadlock(m, X, Y) : boolean;
begin
    if enable(m) = {} then return TRUE fi;
    if X ≠ {} then
        for each x in X do
            (v, n) := LPminimize(x, m);
            if nonzero(v) ∧ n = 0 ∧ feasible(v, m) then
                m' := m + C × v;
                X' := marked(X, m');
                if deadlock(m', X', Y) then return TRUE fi
            fi
        do
    fi;
    if Y ≠ {} then return deadlock(m, Y, {}) fi;
    return FALSE
end
```

The function *enable(m)*, as before, returns the set of transition enabled by m. *LPminimize* is the linear programming procedure which minimizes the number of tokens assigned to the siphon x by the marking m by firing appropriate transitions; it returns a firing vector v (indicating, for each transition, the number of times it needs to fire to minimize the number of tokens assigned to siphon x) and n, the final number of tokens in the siphon x. If v is a nonzero vector and $n = 0$ and if the firing vector v is feasible at m (i.e., there exists a firing sequence that begins at m and corresponds to v), then the current siphon can become unmarked, and the checking continues for a reduced set of siphons X' and a modified marking m'; $\mathbf{C}$ is the connectivity (or incidence) matrix of the analyzed net, and the function *marked(X,m)* returns the set of all those siphons in X which are marked by m. The last section of *deadlock* performs the switch from minimal siphons to basis siphons by invoking *deadlock* with "switched" parameters (and continuing deadlock detection).

If the function *deadlock* returns TRUE, the analyzed net contains a deadlock (a simple modification of the function can provide a firing sequence creating this deadlock); if the function returns FALSE, the net is deadlock–free.

The linear programming procedure returns a firing vector minimizing the number of tokens in the analyzed siphon. Such a firing vector may have no implementation in the form of a firing sequence, i.e., it may be infeasible for a given marking m. Therefore the feasibility of firing vectors is verified by a recursive (boolean) function *feasible(v,m)* which systematically checks the existence of firing sequence starting from m that corresponds to the firing vector v:

```
function feasible (v,m) : boolean;
begin
    if zero(v) then return TRUE fi;
    for each t in enable(m) do
        if v[t] > 0 then
            v' := v;
            v'[t] := v'[t] − 1;
            m' := fire(m, t);
            if feasible(v', m') then return TRUE fi
        fi
    od;
    return FALSE
end
```

If the function returns TRUE, the firing vector v is feasible for marking m; if the returned value is FALSE, such a firing sequence does not exist, and v in infeasible for m. A simple modification of this function returns the firing sequence (if it exists) rather than the value TRUE.

7. Concluding remarks

The paper proposes an efficient method for deadlock detection which is based on reduced sets of minimal and basis siphons. The experience shows that the performance of this method is quite satisfactory, although many further improvements are possible. For example, the discussed reductions of parallel and alternate paths do not guarantee that all equivalent siphons are eliminated, so further reduction can be possible. There are many possible improvements to the procedure of finding (minimal and basis) siphons, and so on.

It can be shown that the ordering of siphons can affect the performance of the deadlock detection algorithm discussed in Section 6. Predicting the most efficient ordering of analyzed siphons (which may be dynamic, i.e., which may be different at different stages of the algorithm) can be an interesting research topic.

The paper does not address the details of using linear programming for finding the firing vectors which minimize the token count in siphons; these details can be found in [4, 10].

Acknowledgement

The Natural Sciences and Engineering Research Council of Canada partially supported this research through grant RGPIN-8222.

References

[1] B. Bordbar, K. Okano, "Testing deadlock-freeness in real-time systems: a formal approach"; *Formal Approaches to Software Testing* (Lecture Notes in Computer Science 3395) pp.95-109, 2004.

[2] E.T. Boer, T. Murata, "Generating basis siphons and traps of Petri nets using the sign incidence matrix"; *IEEE Trans. on Circuits and Systems, I – Fundamental Theory and Applications*, vol.41, no.4, pp.266-271, 1994.

[3] F. Chu, X. Xie, "Deadlock analysis of Petri nets using siphons and mathematical programming"; *IEEE Trans. on Robotics and Automation*, vol.13, no.6, pp.793-804, 1997.

[4] D.C. Craig, "Compatibility of software components – modeling and verification"; Ph.D. Thesis, Department of Computer Science, Memorial University, St.John's, Canada A1B 3X5, 2006.

[5] D.C. Craig, W.M. Zuberek, "Compatibility of software components – modeling and verification"; Proc. Int. Conf. on Dependability of Computer Systems, Szklarska Poreba, Poland, pp.11-18, 2006.

[6] J. Ezpeleta, J.M. Colombo, J. Martinez, "A Petri net based deadlock prevention policy for flexible manufacturing systems"; *IEEE Trans. on Robotics and Automation*, vol.11, no.2, pp.173-184, 1995.

[7] M. Hack, "Analysis of production schemata by Petri nets"; Project MAC Technical Report TR–94, 1972.

[8] T. Murata, "Petri nets: properties, analysis, and applications"; *Proceedings of the IEEE*, vol.77, no.4, pp.541-580, 1989.

[9] W. Reisig, *Petri nets – an introduction* (EATCS Monographs on Theoretical Computer Science 4); Springer-Verlag 1985.

[10] M. Silva, E. Teruel, J. Couvreur, "Linear algebra in and linear programming techniques for the analysis of place/transition net systems"; *Lectures on Petri nets – basic models* (Lecture Notes in Computer Science 1491), pp.309-373, Springer-Verlag 1998.

[11] S. Tanimoto, M. Yamaguchi, T. Watanabe, "Finding minimal siphons in general Petri nets"; *IEICE Trans. on Fundamentals in Electronics, Communications, and Computer Science*, vol.E79-A, no.11, pp.1817-1824, 1996.

[12] M. Yamaguchi, T. Watanabe, "Algorithms for extracting minimal siphons containing specified places in a general Petri net"; *IEICE Trans. on Fundamentals in Electronics, Communications, and Computer Science*, vol.E82-A, no.11, pp.2566-2575, 1999.