A Slicing-Based Approach to Enhance Petri Net Reachability Analysis*

W.J. Lee and H.N. Kim

Software Engineering Department, ETRI
161 Kajong Dong, Yusong Gu, Taejon 305–350, South Korea
Email: {woojin,hnkim}@etri.re.kr

S.D. Cha

Department of EECS and Advanced Information Technology Research Center (AITrc)
Korean Advanced Institute of Science and Technology
373–1, Kusong Dong, Yusong Gu, Taejon 305–701, South Korea
Email: cha@salmosa.kaist.ac.kr

Y.R. Kwon

Department of Electrical Engineering & Computer Science (EECS)
Korean Advanced Institute of Science and Technology
373–1, Kusong Dong, Yusong Gu, Taejon 305–701, South Korea
Email: kwon@salmosa.kaist.ac.kr

Reachability graph analysis is one of the most widely used techniques to verify the behaviour of asynchronous and concurrent systems modeled in Petri nets. Unfortunately, a state explosion problem is often the bottleneck when applying Petri net modeling and analysis to large and complex industrial systems. This paper proposes an approach in which Petri net slices are computed based on structural concurrency inherent in the P/T net and compositional reachability graph analysis is performed. Petri net slices are proven to provide behavioural equivalence to P/T nets. This approach may enable verification of properties such as boundedness and liveness which may fail on unsliced P/T nets due to a state explosion problem. Effectiveness and scalability of our approach is demonstrated using both dining philosophers and feature interaction problems found in telecommunication software.

Key words and Phrases: Petri nets, Place/Transition nets, reachability analysis, Petri net Slice, compositional analysis, structural concurrency.

Classification of the paper: Software-Software Engineering-Software/Program Verification-Formal Methods

1. INTRODUCTION

Petri nets are widely used to model and verify the behaviour of asynchronous systems (Suzuki *et al*, 1990), concurrent systems, and real-time systems (Ghezzi *et al*, 1991; Bucci and Vicario, 1995). Petri net-based formalisms have been significantly extended to enhance expressiveness and to

* This work was partially supported by Korea Science and Engineering Foundation (KOSEF) through the Advanced Information Technology Research Centre (AITrc).

Manuscript received: March 2000
Associate Editor: Hugh Williams

develop powerful and automated analysis techniques. Place/Transition (P/T) nets are often regarded as low-level Petri nets, and extensions include high-level Petri nets such as Jensen's (1992) colored Petri nets(CPN), object-oriented Petri nets (Battiston and Cindio, 1993; Perkusich and Figueiredo, 1997), Lee *et al* (1998) constraints-based modular Petri nets(CMPNs) etc. Although motivations vary significantly, the majority of extended Petri net-based formalisms are compatible in that they can be converted into semantically equivalent P/T nets and that verification techniques developed for P/T nets such as reachability analysis, invariants analysis (Murata, 1989), net unfoldings (Esparza, 1993) can be applied on extended Petri net formalisms.

Reachability analysis is the most frequently employed technique to examine the intended system behaviour. Although simple and straightforward in concept, reachability analysis is often considered impractical due to a state explosion problem when applied to nontrivial and real world problems. Several approaches have been proposed to try minimizing the number of system states when generating a reachability graph. Reduction rules (Murata, 1989) were proposed to reduce the size of the original model. Jorgensen (1997) demonstrated the potential of verification based on state spaces reduced by equivalence relation. And compositional analysis approaches reduce the possibility of state explosion by incrementally generating reachable states, which was applied to Modular Petri nets (Damm *et al*, 1989; Valmari, 1990; Notomi and Murata, 1994; Christensen and Petrucci, 1995; Schreiber, 1995). Unfortunately, compositional analysis, while an attractive option, may not be applied directly on P/T nets since a system model must be composed as a set of modular Petri nets. However, the majority of industrial applications of Petri nets still rely on classical P/T nets or high-level extensions such as Colored Petri nets, and there are no systematic approaches to convert a P/T net or CPN into modular Petri nets.

Our work is an attempt to enable modelling and analysis of P/T nets on large and complex industrial systems which previously could not be analysed due to the state explosion problem. By developing partitioning criteria and a slicing algorithm for dividing a huge P/T net model into meaningful and manageable modules, the compositional analysis technique may be applicable to P/T net models. In order to enhance the efficiency of compositional reachability analysis, modules should be defined by considering cohesive internal dependency and independency among modules. A slicing algorithm has a role to find the concurrent units and to partition a huge model into Petri net slices while preserving the behaviours of the original model by using concurrent units. Reachability analysis of Petri net slices can be performed in a compositional approach since Petri net slices are modules with synchronising by shared transitions. An S-invariant property can be used in analysing boundedness checking since each Petri net slice is deduced from S-invariant.

This paper is structured as follows. Section 2 presents the concurrency in P/T nets and defines structural concurrency and concurrent units. In section 3, we propose a slicing algorithm which finds concurrent units and partitions the original model into Petri nets slices. And we prove that the original model and Petri nets slices have the same behaviours. We introduce the compositional reachability analysis technique by using Petri net slices in section 4. In order to show effectiveness and practicability of our approach, in section 5, we describe the dining philosophy problem and feature interaction problems of telecommunication systems by Petri net slices. Finally, in section 6, we provide conclusions and future work.

2. CONCURRENT UNITS IN P/T NETS

Petri nets have been used for describing the concurrent properties of a system. In a Petri net model, the concurrency is represented by relationships among transitions which are simultaneously enabled without any causal dependency. Figure 1 shows a P/T net model for the dining philosopher problem.

After spending time thinking, a philosopher takes a left and right fork in arbitrary order, and eats spaghetti. To avoid deadlock situations in the dining philosopher problem, we add an additional functionality into the model: although a philosopher has taken a fork, he or she can abandon trying to take the other fork, release the taken fork, and return to the thinking state.

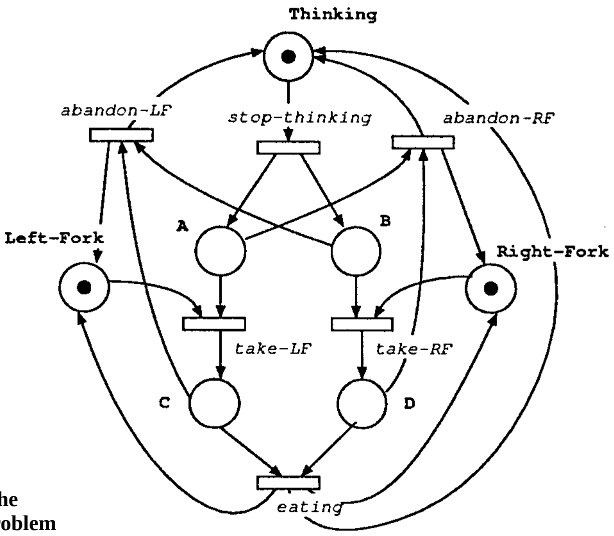

Figure 1:
A P/T net model for the dining philosopher problem

Concurrency in Petri net models are divided into structural concurrency and token concurrency. Structural concurrency is a behavioural characteristic appearing in structural connections among places and transitions. In Figure 1, transitions *take-LF* and *take-RF* are simultaneously firable if places A, B, Left-Fork, and Right-Fork have tokens. Finding structural concurrency is performed by checking the causality in the model without considering the markings of places. If two transitions have no causal relationship, the two transitions are structurally concurrent. Token concurrency is not relevant to the structural architecture but markings of places. If places Thinking, B, and Right-Fork have tokens, transitions *stop-thinking* and *take-RF* are concurrently firable, although they have causal dependency. Usually, a Petri net model may have both structural concurrency and token concurrency simultaneously. However, we consider only structural concurrency in partitioning the model since token concurrency may appear among all the pairs of transitions on the assumption that all the places in the model have sufficient tokens.

In a Petri net model, there may be tokens in several places, which have their own control threads. Since transitions may join several control threads by collecting incoming arcs or may divide a single control thread by distributing several outgoing arcs, these control threads are complicatedly interrelated. Although a token interacts with other ones by transitions, its trajectory can be deterministically and independently definable.

A concurrent unit can be detected by a trajectory of a virtual token, which means that a trajectory may start from a virtual token of an arbitrary place, not always from an initial token. Trajectories of virtual tokens may be obtained by using S-invariant concepts (Reisig, 1985), which represent the sets of places which do not change their token count during transition firings. A trajectory of a token is a special S-invariant where the token count is one. Figure 2 shows an example of concurrent units in the dining philosopher model. The first concurrent unit is the trajectory of a token in Left-Fork, which is the trajectory of the left fork, and the second one is the trajectory of a token in Thinking, which means that a philosopher handles the right fork.

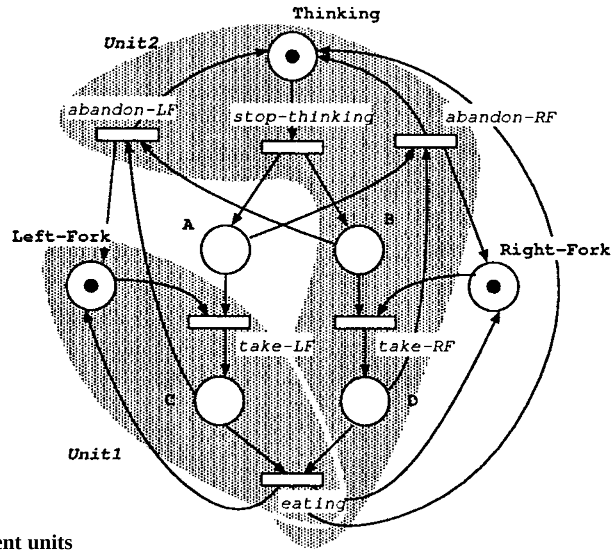

Figure 2:
An example of concurrent units

S-invariants are formally defined as follows. In general, the arc weights are positive integer and default arc weight is 1.

Definition 2.1 S-invariants
Let N be a P/T net. A place vector, $i : S_N \rightarrow Z$ *is called an S-invariant of N iff* $\underline{N}' \cdot i = 0$, *where* $\underline{N}'$ *is a matrix representation of N (transitions* x *places).*

Let the number of places in a Petri net model be *n*. The solution of the matrix equation, $\underline{N}' \cdot i = 0$, can be obtained in time complexity $O(n^3)$ (Reisig, 1985). Solution vectors, that is, S-invariants may include negative integers. And two solution vectors may be merged to be another one. Special S-invariants which can be used in constructing concurrent units, *minimal invariants,* are defined as follows.

Definition 2.2 *Minimal Invariants*
Positive S-invariants which are not contained in other S-invariants are called minimal invariants.

In Figure 2, $Unit_1$, $Unit_2$, {*Right–Fork, D*} and {*Thinking, A, C*} are all minimal invariants.

3. SLICING A PETRI NET MODEL

A Petri net model is partitioned into concurrent units using minimal invariants. In order to preserve all the information in the original model, uncovered places should be added into minimally-connectable concurrent units since minimal invariants may not cover all the places.

```
SliceSet = $\phi$;
SetofInvariant = find_minimal_invariants(N);
do {
    small_invariant = find_smallest_invariant(SetofInvariant);
    SliceSet = SliceSet $\cup$ { small_invariant };
    SetofInvariant = SetofInvariant - { small_invariant };
} until ( Place(SetofInvariant) $\subseteq$ Place(SliceSet) OR Place(N) == Place(SliceSet) );
if ( Place(SliceSet) $\neq$ Place(N) ) {
    Uncovered_Place_Set = Place(N) - Place(SliceSet);
    for $\forall p$ $\in$ Uncovered_Place_Set do {
        slice = find_minimally_connected(SliceSet, p);
        slice = slice $\cup$ { p };
    }
}
```

Figure 3: The Slicing Algorithm for Petri net models

Figure 3 shows a slicing algorithm which slices a Petri net model into a set of Petri net slices using minimal invariants. At first, S-invariants are computed and then the minimal invariants are selected among S-invariants (*find_minimal_invariants*). In the minimal invariant set, the algorithm chooses an invariant which has the minimal number of elements (*find_smallest_invariant*) and adds it into *SliceSet* until *SliceSet* covers all the places in *N* (*Place*(*N*) == *Place*(*SliceSet*)) or there exists no minimal invariant which includes a new place (*Place*(*SetofInvariant*) $\subseteq$ *Place*(*SliceSet*)). If the minimal invariant set becomes empty without covering all the places in *N*, for each uncovered place, it should be added into a slice to which it is connected by a minimal number of transitions (*find_minimally_connected*). In this manner, the slicing algorithm ensures that every place in *N* belongs to some slices.

The complexity of the slicing algorithm is mainly dependent on three time-consuming procedures: *find_minimal_invariant*, *find_smallest_invariant*, and *find_minimally_connected*. The time complexity of *find_minimal_invariant* is the same as that of finding S-invariants, which is described in the previous section. Finding a smallest invariant is similar to sorting invariants in criteria of the number of elements. In *find_minimally_connected* procedure, one transition-connectable places, which are connected to the uncovered place through a transition, are checked if they are members of any SliceSet. If one of them is included in some SliceSet, the uncovered place is added into the SliceSet. Otherwise, the procedure proceeds to two transitions-connectable places. Therefore, the time complexity of the slicing algorithm is $O(n^3)$ for *n* number of places in a P/T net.

A modular Petri net, *Petri net slices*, is defined using *SliceSet* as follows.

Definition 3.1 *Petri net slices*
Let N = (P, T, F, W, M) be an original P/T net where P represents a set of places, T is a set of transitions, F is a set of arcs, W is weight functions of arcs, and M is the initial marking. And let SliceSet = $\{P\_Slice_i \mid i = 1 \ldots n\}$ *be a set of place sets which are obtained by the slicing algorithm. Petri net slices are defined as* $\{Petri\ net\ slice_i \mid i = 1 \ldots n\}$*, where Petri net* $slice_i = (P_i, T_i, F_i, L_i, W_i, M_i)$ *satisfies the following:*

- $P_i = P\_slice_i$,
- $T_i = \{ t \in T \mid \forall p \in P_i, \exists (p, t) \in F \text{ or } \exists (t, p) \in F \}$,

- $F_i = \{ (p, t) \in F \text{ or } (t, p) \in F \mid \forall p \in P_i, \exists t \in T_i \}$,
- $L_i : T_i \rightarrow \Sigma^+$ *is a label function of transitions, where* Σ *is a character set. We assign the name of a transition (e.g. tl) in N to a label of t, that is,* $L_i(t) = tl$,
- $W_i(p) = W(p)$, *and* $M_i(p) = M(p)$, $\forall p \in P_i$.

In a single Petri net model, a transition *t* is enabled if the places connected by incoming arcs of *t* (denoted as $^\bullet t$) have sufficient tokens. In Petri net slices, shared transitions whose labels appear in several slices are fired by synchronising the same labels. Following defines enabledness of a transition in Petri net slices.

- $L_i(t)$ is not shared: a transition *t* is enabled if the precondition of *t* is satisfied.
- $L_i(t)$ is shared among several Petri net slices: a transition *t* is enabled if *t* is enabled in all the slices which have the label of *t*.

Figure 4 shows the result of applying the slicing algorithm to the model in Figure 2. The slices shown in Figure 4(a) and (b) are S-invariants containing Left-Fork place and Right-Fork place, respectively. Figure 4(c) represents the behaviour for taking the left fork or abandoning the taken right fork. Figure 4(d) represents the behaviour for taking the right fork. In the dining philosopher model, there exists no uncovered place.

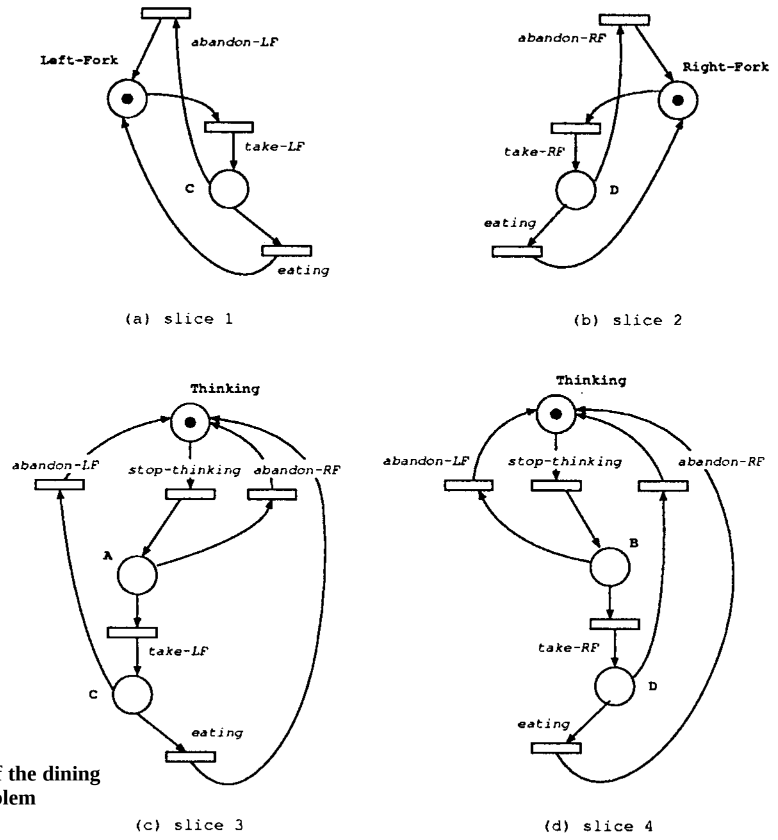

Figure 4:
Petri net slices of the dining philosopher problem

In order to show that the original model and the Petri net slices obtained by applying the slicing algorithm have equivalent behaviours, we define the concept of behavioural equivalence of Petri net models. In the definition, we concern only observational behaviour, where the system behaviours are viewed as black boxes by considering only externally visible behaviours without considering the internal structures of the models (Milner, 1989).

Definition 3.2 *Behavioural Equivalence of two P/T nets (P ~ Q) :*
P and Q are behaviourally equivalent iff there exists an one-to-one mapping between the reachability graphs of P and Q.

Theorem 3.1 *Let N be a Petri net and S be the Petri net slices obtained by the slicing algorithm. The original model N and the slice model S are behaviourally equivalent (N ~ S)*

Mapping of RGs can be characterised by mapping between occurrence sequences of corresponding transitions. And transition occurrences are determined by pre/post-places of transitions. Therefore, equivalent RGs have the same initial marking and the same pre/post-conditions of the corresponding transitions.

Since the slicing algorithm preserves input and output flows information of places, all the corresponding places of N and S have the same initial tokens and the same behaviours. Hence, we show only the equivalence of pre/post-conditions of t_N and its corresponding label tl_S.

(Proof)
Let $N = (P,T,F,W,M)$ be a Petri net model and $S = \{C_{n_i} | i = 1...n\}$, where $C_{n_i} = (P_i, T_i, F_i, L_i, W_i, M_i)$, which is obtained by applying the slicing algorithm to N, be Petri net slices.

Let tl_S be a uniquely corresponding label of a transition t_N. Since a transition label can have multiple instance transitions in several C_{n_i}, pre/post-place sets of label tl_S are differently denoted as $°tl_S$ and $tl_S°$, and defined by the unions of pre/post-place sets ($^\bullet t_{C_{n_i}}$ $/t^\bullet{}_{C_{n_2}}$) of multiple instance transitions $t_{C_{n_i}}$.

We will first show that $^\bullet t_N$ is equal to $°tl_S$ by showing $°tl_S \subseteq {}^\bullet t_N$ and $^\bullet t_N \subseteq °tl_S$ on the assumption that the label tl_S is the name of t_N.

Case 1: $°tl_S \subseteq {}^\bullet t_N$

For each C_{n_i}, if there exists a transition $t_i \in T_i$ such that $L_i(t_i) = tl_S$, it must be an instance of t_N. Since incoming arc information of t_i was derived from the arc information of t_N through the slicing algorithm, the pre-places of t_i should be a subset of the pre-places of t_N. Therefore, $°tl_S$ is a subset of $^\bullet t_N$ since $^\bullet t_i$ is a subset of $^\bullet t_N$ for all C_{n_i}.

Case 2: $^\bullet t_N \subseteq °lt_S$

In order to show the subset relation of two sets, we proof the following equivalent implication relation, $\exists p \in {}^\bullet t_N \Rightarrow p \in °tl_S$. For each $p \in {}^\bullet t_N$, the place p should be included in one or more C_{n_i}. Assume that the place p is contained in C_{n_2}. Then a transition t_2 such that $L_2(t_2) = tl_S$ should be included in T_2 since the incoming arc information (t_N) of p should be included in C_{n_i}. That is, $p \in {}^\bullet t_2$. Therefore, p is also included in $°tl$ since $^\bullet t_2$ is a subset of $°tl$.

Similarly, the equation $t_N^\bullet = tl_S^\circ$ can be proved.

4. COMPOSITIONAL REACHABILITY ANALYSIS OF PETRI NET SLICES

Compositional analysis can be applicable to a system model which is composed of several modules. After generating the reachability graph of each module and analysing the local properties of the modules, the local reachability graphs are reduced and composed to analyse the global properties of the system. Compositional generation and analysis of reachability graphs are well described in Yeh's (1993) work.

Figure 5 describes the composition procedure of two reachability graphs (RGs). Figures 5(a) and (b) represent the reachability graphs of slice1 and slice3 shown in Figure 4, respectively. In combining two RGs, the labels of transitions have an important role. Transitions with the same label can be synchronously performed only when they are ready in each RG. On the other hand, when non-shard transitions are ready in its own RG, they can occur regardless of the states of other RGs. In Figure 5(c), for example, a shared transition, *take-LF*, only occurs at state RS1.4, which indicates that slice1 is at state RS1 and slice3 is at state RS4.

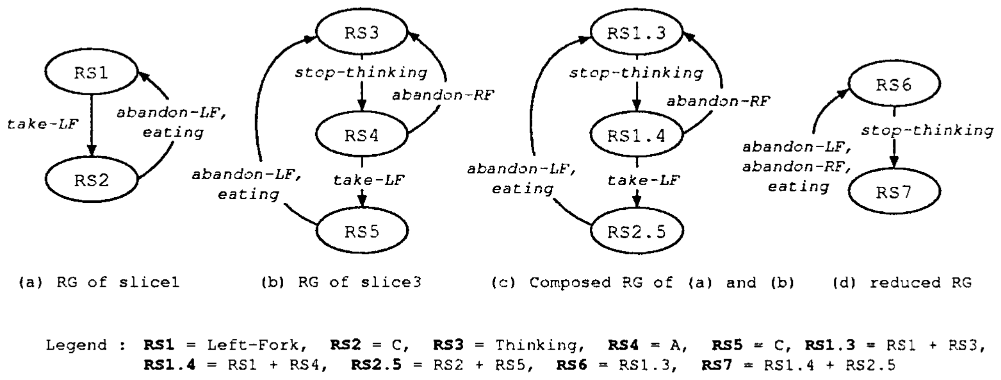
RS1
take-LF
abandon-LF, eating
RS2
RS3
stop-thinking
abandon-RF
RS4
abandon-LF, eating
take-LF
RS5
RS1.3
stop-thinking
abandon-RF
RS1.4
abandon-LF, eating
take-LF
RS2.5
RS6
stop-thinking
abandon-LF, abandon-RF, eating
RS7
(a) RG of slice1
(b) RG of slice3
(c) Composed RG of (a) and (b)
(d) reduced RG
Legend : RS1 = Left-Fork, RS2 = C, RS3 = Thinking, RS4 = A, RS5 = C, RS1.3 = RS1 + RS3,
RS1.4 = RS1 + RS4, RS2.5 = RS2 + RS5, RS6 = RS1.3, RS7 = RS1.4 + RS2.5

Figure 5: Composition procedure of two reachability graphs

Among the transition labels in Figure 5(c), the label *'take-LF'* does not appear in other RGs of slice2 and slice4 as shown in Figure 4. These transitions are called 'local' transitions. Since local transitions have no impacts on interactions with other slices, they can be reducible. By reducing the *take-LF* transition, RS1.4 and RS2.5 states are merged to RS7 state, as shown in Figure 5(d).

Reachability analysis of Petri net slices is performed on the hierarchical reachability graph of slices. Liveness property of transitions should be hierarchically checked since the behaviours of shared transitions are characterised by those of other slices. On the other hand, Boundedness property of places are checked in each reachability graph of slices. The characteristic of S-invariants is that all the outgoing and incoming arc information of a place are preserved in each S-invariant. Therefore, the global behaviours of a place can be predictable in local slices. If there is an ω-place(a place which can have an unlimited number of tokens) in the reachability graph of a slice, we can find that the place is globally unbounded.

5. CASE STUDY

In this section, we apply Petri nets slices to dining philosophers problems and feature interaction problems of telecommunication systems (Keck and Kuehn, 1998). As an ideal concurrent example, we choose dining philosophers problems to illustrate the efficiency of Petri net slices in handling concurrency during generating reachability graphs. And feature interaction problem is chosen to show applicability of our approach to a practical example and to briefly mention how to use Petri net slices in analysing system properties such as unboundedness and nondeterminism.

5.1 Dining Philosophers problems

P/T nets, Modular Petri nets (Notomi and Murata, 1994), and Petri net slices approaches are used to describe dining philosophers problems. The P/T net model shown in Figure 1 is extended for describing several philosophers. In Modular Petri net approach, although a modeller can divide an entire problem into modules by his/her own partitioning criteria, we assume that the model is composed of subnets of each philosopher and each fork. Petri net slices are obtained by applying the slicing algorithm to the P/T net model. A single philosopher case has four slices. In generating compositional reachability graphs, the number of system states varies on the composition orders (Yeh, 1993). In order to fairly compare Modular Petri nets and Petri net slices approaches, we apply the same composition order when generating compositional reachability graphs. Table 1 shows the numbers of reachable states and state transitions of three approaches.

phil. #	P/T nets	Modular PN		PN slices	
		# modules	# states(trans.)	# modules	# states(trans.)
1	5(8)	3	15(24)	4	18(29)
2	18(46)	4	20(40)	8	38(61)
3	76(291)	6	55(134)	12	58(93)
4	322(1644)	8	77(195)	16	78(157)
5	1364(8705)	10	99(256)	20	98(157)
6	5778(44250)	12	121(317)	24	118(189)
7	24476(218687)	14	143(378)	28	138(221)
8	103682(1058712)	16	165(439)	32	158(253)
9	439204(5045373)	18	187(500)	36	178(285)
10	-	20	209(561)	40	198(317)
20	-	40	429(1171)	80	398(637)
30	-	60	649(1781)	120	598(957)
50	-	100	1089(3001)	200	998(1597)

Table 1: The numbers of reachable states and state transitions for dining philosopher models

As shown in Table 1, while the numbers of reachable states and state transitions grow exponentially in P/T nets, the numbers in Modular Petri nets and Petri net slices grow slowly. That is, the compositional approach is an effective method to reduce state explosion. The number of modules in Petri net slice approach is larger than that of Modular Petri nets. That is, Petri net slice approach can support finer granularity of concurrency. Therefore, as the number of philosophers grows, the number of reachable states in Petri net slices is a little less than that in Modular Petri nets, and large differences are shown in the numbers of state transitions.

5.2 Feature Interaction Problem

In telecommunication systems, the term *feature interaction* refers to situations, where different service features or instances of the same service feature affect each other. This can take place within the same service as well as between features of different services. As the increasing demand for new telecommunication services has led to a rapidly growing number of new services, as well as to the enhancement of existing services with new service features, there are strong demands on formal approaches that describe the behaviours of service features and detect undesirable interactions among them.

In this section, we describe basic call processing (BCP) and call forwarding (CF) service features by P/T nets and analyse the interaction between BCP and CF service features by using Petri net slices.

There are six representative use cases dealing with basic call processing in the tele-communication software, where the first three use cases are extracted in a caller's viewpoint and the others are described in a callee's viewpoint. CF service feature has two use cases; one is to subscribe or unsubscribe the CF service feature and the other represents the functionality of call forwarding. The following shows eight use cases corresponding to BCP and CF service feature.

- use case 1 : a caller calls, talks with, and hangs up.
- use case 2 : a caller dials a number, but no one answers.
- use case 3 : a caller calls, but the destination line is busy.
- use case 4 : a callee picks up phone, talks with and hangs up.
- use case 5 : before picking up the phone, ringing stops.
- use case 6 : while talking with someone, another call arrives.
- use case 7 : an incoming call is forwarded to the CF-registered destination.
- use case 8 : a user can subscribe or unsubscribe CF service feature.

Figure 6 shows a P/T net model representing eight use cases. The shaded part represents use case 7, which represents the call forwarding service feature.

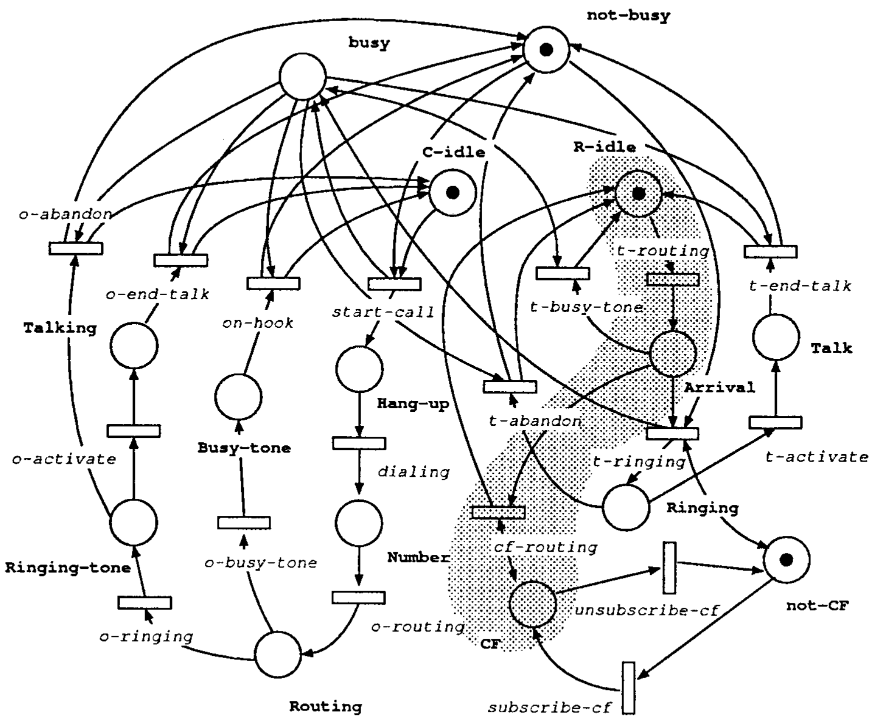

Figure 6:
A P/T net for BCP and CF

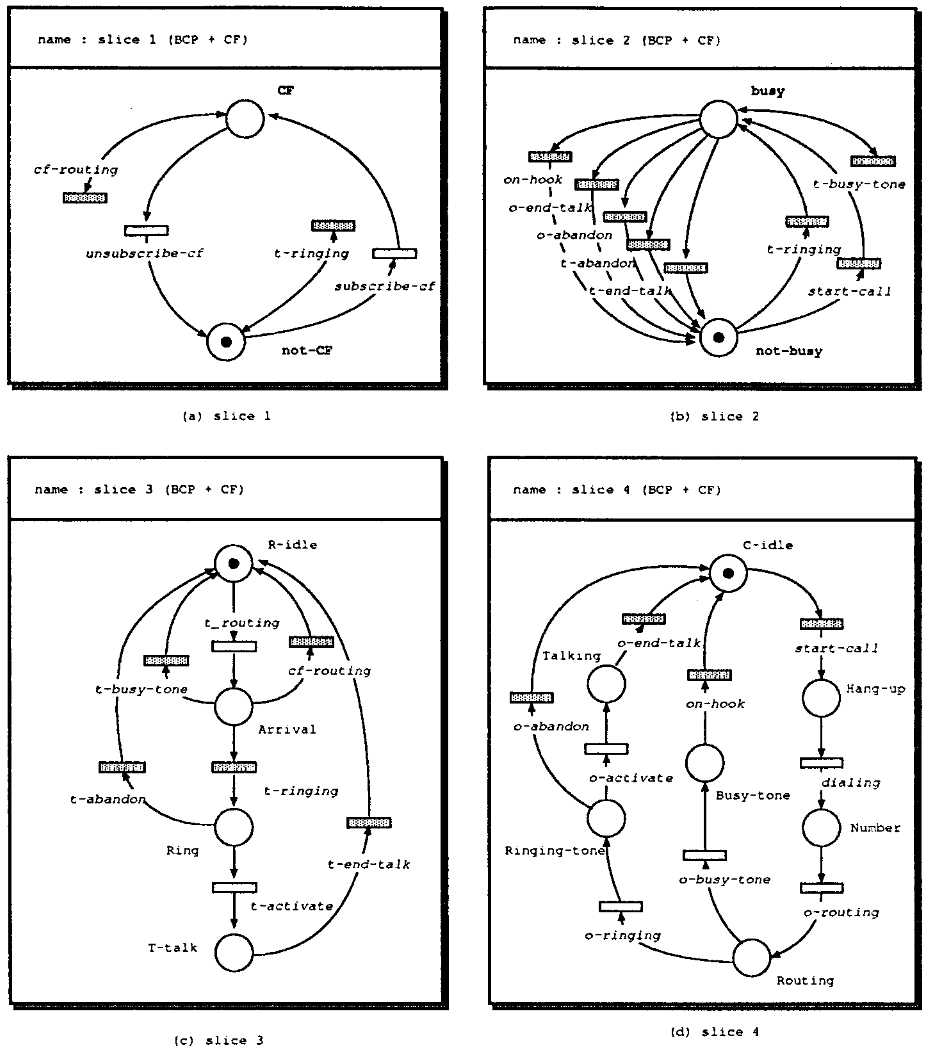

Figure 7: Slice sets of the BCP and CF model

After applying the slicing algorithm to the P/T net model for BCP and CF, we obtain the Petri net slices shown in Figure 7. The slice of Figure 7(a) is related with forwarding functionality, and Figure 7(b) shows a slice for *busy* and *not-busy* toggling conditions. Figures 7(c) and (d) represent the behaviours of a callee and a caller, respectively.

In Figure 7, each slice has no unbounded place. Since boundedness property of places can be locally checkable as mentioned in the previous section, there is no unbounded place in the BCP and CF model. Nondeterminism among transitions can be checked on compositional reachability tree. In the reachability graphs of Petri nets slices, there may be many locally-nondeterministic situations since the interactions with other slices are not considered. In the composed reachability graphs, however, these nondeterministic situations may be resolved by synchronising shared transitions.

In Figure 7(c), there are two nondeterministic situations: {*t-abandon*, *t-activate*} and {*t-busy-tone*, *t-ringing*, *cf-routing*}. A nondeterministic situation between *t-abandon* and *t-activate* is an external caller's choice inherited from BCP model, which is not related with interactions among BCP and CF. Among nondeterministic situations in {*t-busy-tone*, *t-ringing*, *cf-routing*}, the cases {*t-ringing*, *cf-routing*} and {*t-ringing*, *t-busy-tone*} are resolved by merging with RGs of slice1 and slice2, respectively. However, the nondeterministic situation of *t-busy-tone* and *cf-routing* is not resolved in composed RGs, which means that BCP's busy-tone signaling and CF's cf-routing can nondeterministically occur when another call arrives during conversation. This nondeterminism is caused by the modeler's mistake, that is, he forgot adding *not-CF* condition to *t-busy-tone* transition in Use Case 6 during introducing the call forwarding SF into BCP model.

As shown in the above example, the role of Petri nets slices approach is to divide a huge P/T net model into several manageable modules for applying compositional reachability analysis techniques. In Petri net slices, Petri net model analysers can focus on each local slice and incrementally proceed the analysing procedure to composed models for verifying interactions of slices.

6. CONCLUSIONS AND FUTURE WORK

We proposed the Petri nets slice approach in order to partition huge P/T net models into manageable modules, so that the partitioned model can be analysed by compositional reachability analysis technique. The Petri net slice approach can be used for avoiding state explosion in reachability analysis of a huge P/T net model and for efficiently analysing system properties in compositional approach. Our approach is well suited to concurrent or distributed systems, which are described by P/T nets or P/T nets-transformable high-level Petri nets.

As future work, we have a plan to extend Petri net slice concept to be directly applicable to Colored Petri nets, which have been frequently used in practical applications. We have implemented the slicing algorithm in the form of a text-based package. We have a plan to connect this package with a new graphical Petri net tool or existing Petri net tools such as Design/CPN (University of Aarhus, 1996). And we will enhance the compositional reachability analysis techniques on the Petri net slices and implement supporting packages.

REFERENCES

SUZUKI, T., SHATZ, S.M., and MURATA T. (1990): A Protocol Modeling and Verification Approach Based on a Specification Language and Petri Nets, *IEEE Trans. on Software Engineering*, 16(5), May: 523-536.

GHEZZI, C., MANDRIOLI, D., MODASCA, S., and PEZZE, M. (1991): A Unified High-Level Petri Net Formalism for Timed-Critical Systems, *IEEE Trans. on Softw. Eng., 17(2)*: 160-172.

BUCCI, G. and VICARIO, E. (1995): Compositional Validation of Time-Critical Systems Using Communicating Time Petri Nets, *IEEE Trans. on Softw. Eng., 21(12)*: 969-992.

JENSEN, K. (1992): *Coloured Petri Nets: Basic Concepts, Analysis methods and Practical Use. Volume 1*, Springer-Verlag.

BATTISTON, E. and CINDIO, F.D. (1993): Class Orientation and Inheritance in Modular Algebraic Nets, *Proc. of IEEE Conf. on System, Man, and Cybernetics, 2*: 712-723.

PERKUSICH A. and FIGUEUREDI J. (1997): G-Nets: a Petri Net Based Approach for Logical and Timing Analysis of Complex Software Systems, *J. of Syst. and Softw., 39:* 39-59.

LEE, W.J., CHA, S.D., and KWON, Y.R. (1998): Integration and Analysis of Use Cases Using Modular Petri Nets in Requirements Engineering, *IEEE Trans. on Softw. Eng., 24(12):*1115-1130.

MURATA, T. (1989): Petri nets : Properties, Analysis and Applications, *Proc. of the IEEE, 77(4):*541-580.

ESPARZA, J. (1993): Model Checking using net unfoldings, *TAPSOFT '93(LNCS #668)*, 613-628.

JORGENSEN, J.B. and KRISTENSEN, L.M. (1997): Verification of Coloured Petri Nets Using State Spaces with Equivalence Classes, *Proc. of the Workshop on Petri Nets in Sys. Eng.(PNSE'97)*, September.

DAMM, W., DOHMEN, G., GERSTNER, V., and JOSKO, B. (1989): Modular Verification of Petri Nets: The Temporal Logic Approach, *LNCS #430*, 180-207.

VALMARI, A. (1990): Compositional State Space Generation, *Proc. of Appl. and Theory of Petri Nets '89.*
NOTOMI, M. and MURATA, T. (1994):Hierarchical Reachability Graph of Bounded Petri Nets for Concurrent-Software Analysis, *IEEE Trans. on Softw. Eng., 22(5)*: 325-336.
CHRISTENSEN, S. and PETRUCCI, L. (1995): Modular State Space Analysis of Coloured Petri Nets, *Proc. of Appl. and Theory of Petri Nets '95(LNCS #935),* 201-217.
VALMARI A. (1994): Compositional Analysis with Place-Bordered Subnets, *Proc. of Appl. and Theory of Petri Nets '94(LNCS #815),* 531-547.
SCHREIBER G. (1995): Functional Equivalences of Petri Nets, *Proc. of Appl. and Theory of Petri Nets '95(LNCS #935),* 432-450.
REISIG, W. (1985): *Petri Nets:An Introduction,* Springer-Verlag.
YEH, W.J. (1993): *Controlling State Explosion in Reachability Analysis,* PhD. Thesis, Purdue University, December.
MILNER, R. (1989): *Communication and Concurrency,* Prentice-Hall.
KECK D.O. and KUEHN P.J. (1998): The Feature and Service Interaction Problem in Telecommunication Systems: A Survey, *IEEE Trans. on Softw. Eng., 24(10):* 779-796.
UNIVERSITY OF AARHUS (1996): *Design/CPN User's Guide : Version 3.0,* University of Aarhus.

BIOGRAPHICAL NOTES

Woo Jin Lee received his B.S. degree in computer science from KyungPook National University, Korea, in 1992 and M.S. and Ph.D. degrees in computer science from the Korea Advanced Institute of Science and Technology (KAIST), Korea, in 1994 and 1999. He is currently a senior member of engineering staff in Electronics and Telcommunication Research Institute (ETRI) Computer & Software Technology Laboratory (CSTL).

His research interests include requirements engineering, safety-critical systems, component-based software engineering and process modelling.

Sung Deok (Stephen) Cha received B.S., M.S. and Ph.D. degrees in information and computer science from the University of California, Irvine in 1983, 1986 and 1991. He taught at California State University of Borthridge and Long Beach from 1985 to 1990. From 1990 to 1994, he was a member of technical staff at the Hughes Aircraft Company, Ground Systems Group and the Aerospace Corporation where he worked on various projects on software safety and computer security. In 1994, he became a faculty member of the KAIST Computer Science department.

His research interest include software safety, formal methods and computer security.

Yong Rae Kwon received B.S. and M.S. degrees in physics from Seoul National University, Korea, in 1969 and 1971 respectively, and a Ph.D. in physics from the University of Pittsburgh in 1978. He taught as an instructor at Korea Military Academy from 1971 to 1974. He was on the technical staff of Computer Science Corporation from 1978 through 1983 working on the ground support software systems for NASA's satellite projects. He joined the Faculty of Computer Science of KAIST in 1983.

His research interests include verification of real-time parallel software, object-oriented technology for real-time systems and quality assurance for highly dependable software.

Heung-Nam Kim received a B.S. degree in electronic engineering from Seoul National University, Korea, in 1980, an M.S. in computer science from Bell State University in 1989, and Ph.D. in computer science from Pennsylvania State University in 1996.

He has been working for ETRI CSTL since 1983 and is currently the leader of the embedded software research team.

His research interests include video compression algorithm, real-time systems, software engineering and distributed multimedia systems.