# Integration and Analysis of Use Cases Using Modular Petri Nets in Requirements Engineering

Woo Jin Lee, *Student Member*, *IEEE*, Sung Deok Cha, *Member*, *IEEE*, and Yong Rae Kwon, *Member*, *IEEE*

**Abstract**—It is well known that requirements engineering plays a critical role in software quality. The use case approach is a requirements elicitation technique commonly used in industrial applications. Software requirements are stated as a collection of use cases, each of which is written in the user's perspective and describes a specific flow of events in the system. The use case approach offers several practical advantages in that use case requirements are relatively easy to describe, understand, and trace. Unfortunately, there are a couple of major drawbacks. Since use cases are often stated in natural languages, they lack formal syntax and semantics. Furthermore, it is difficult to analyze their global system behavior for completeness and consistency, partly because use cases describe only partial behaviors and because interactions among them are rarely represented explicitly. In this paper, we propose the Constraints-based Modular Petri Nets(CMPNs) approach as an effective way to formalize the informal aspects of use cases. CMPNs, an extension of Place/Transition nets, allow the formal and incremental specification of requirements. The major contributions of our paper, in addition to the formal definitions of CMPNs, are the development of: 1) a systematic procedure to convert use cases stated in natural language to a CMPN model; and 2) a set of guidelines to find inconsistency and incompleteness in CMPNs. We demonstrate an application of our approach using use cases developed for telecommunications services.

**Index Terms**—Use cases, scenarios, requirements engineering, Petri nets, incremental specification, use case dependency analysis, Petri nets slice.

## 1 Introduction

The importance of requirements engineering cannot be overemphasized. As stated eloquently by Brooks [4], "the hardest single part of building a software system is deciding what to build. … No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later." The ultimate goal of requirements engineering is to produce a requirements specification that is correct, consistent, complete, unambiguous, understandable, and traceable. To accomplish this goal, a specifier incrementally builds up the specification through iterative processes involving elicitation, specification, and validation. In order to accurately reflect users' real needs, requirements engineering processes should ideally involve the active participation of users.

A system must often support multiple users with diverse needs and viewpoints. But, a given group of users is most likely to be concerned about and state requirements only on those specific behaviors that are of particular interest to it. Although such groups of requirements might *seemingly* be an independent collection of functionalities, they are rarely truly independent in practice. Therefore, it is extremely important for the requirements engineering technique to support incremental specification of partial system behaviors derived from multiple viewpoints and to enable verification of consistency and completeness among the requirements. Otherwise, the incremental and partial requirements elicitation process will surely fail.

The use case approach, originally proposed by Jacobson et al. [16], [17] and based on the concept of scenarios,[1] is arguably one of the best known and most widely employed requirements elicitation techniques in the industry. System functionalities are stated as a collection of use cases, each of which represents a specific flow of events. The use case approach offers several practical advantages. First, use cases are easy to describe and understand. Second, they are scalable, in that the behavior of a large and complex system can be stated as a collection of independently and incrementally developed use cases. Third, it is relatively easy to provide requirements traceability throughout the design and implementation.

Unfortunately, the use case approach exhibits several shortcomings. First, use cases are often stated in natural languages lacking in formal syntax and semantics. In addition to the risk of ambiguity, the type and rigor of the analysis one may perform on informal requirements are clearly limited. Second, there are currently no systematic approaches to analyzing dependencies among use cases

• *The authors are with the Department of Computer Science, Korea Advanced Institute of Science and Technology, 373-1 Kusong dong, Yusong gu, Taejon 305-701, Korea.*
*E-mail: {woojin, cha, yrkwon}@salmosa.kaist.ac.kr.*

*Manuscript received 20 Dec. 1997; revised 3 July 1998. Recommended for acceptance by R. Kurki-Suonio and M. Jarke.*
*For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 107477.*

1. Scenarios and use cases are used interchangeably throughout this paper.

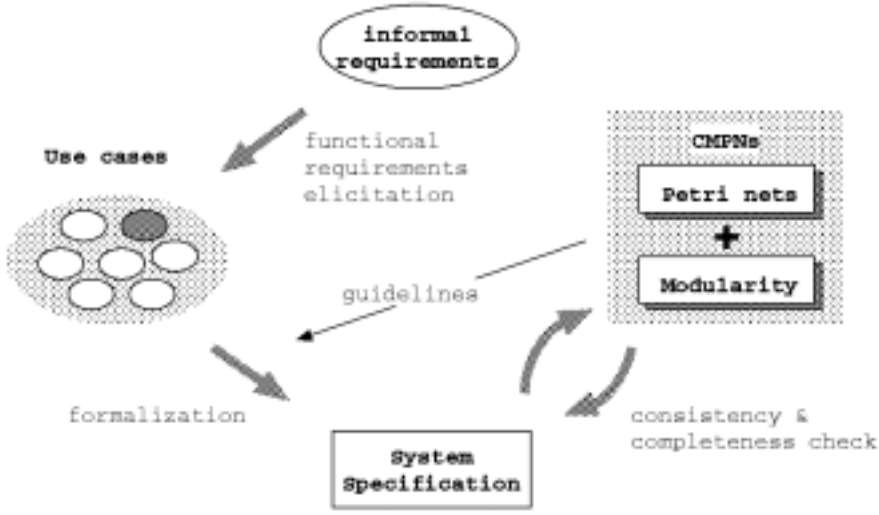



Fig. 1. Overview of our approach.

and to detecting flaws. To avoid costly fixes, to maximize software reliability, and to improve productivity of software development, incompleteness and inconsistency among use cases must be resolved prior to advancing to the design phase.

When developing formal notations for use cases, partiality is appropriate. The notation chosen for use case expression must be flexible enough to allow incomplete specification about actions because users may understand the required system behavior only partially. Yet the notation must be formal enough to allow for the detection of inconsistencies among the partial information provided. Insensitivity to change is also desirable. That is, when a new use case is introduced, it must not unnecessarily or excessively affect the existing use cases. Similarly, when an existing use case is modified or dropped, the scope of modification needs to be minimized. Otherwise, the incremental elicitation of the requirements becomes impractical.

In this paper, we propose an approach to overcoming the limitations of the use case approach, while preserving its advantages, by proposing a formal syntax and semantics for describing use cases. Our notation, referred to as Constraints[2]-based Modular Petri Nets (CMPNs), satisfies the requirements for partiality and insensitivity to changes. Our ultimate research goal is to develop systematic procedures to enable the intuitive, yet formal and incremental, description of requirements and to provide powerful analysis techniques for detecting flaws in requirements as early as possible in the development life cycle. Fig. 1 illustrates our approach.

The rest of our paper is organized as follows: In Section 2, we briefly review some of the earlier proposals made for formalizing use cases and identify their shortcomings. In Section 3, we justify a need for defining CMPNs by illustrating why existing Petri net formalisms, such as P/T nets or colored Petri nets, are inadequate to accomplish our goal. We provide an informal and intuitive introduction to CMPNs, as well as formal definitions and firing semantics. Slicing is introduced as a means of conducting efficient and compositional behavioral analysis of CMPNs. Section 4 describes a systematic procedure for converting use cases stated in natural language to a CMPN, based on an extended form of action-condition table. In Section 5, the analysis techniques for ensuring consistency and completeness in a CMPN model are discussed. We demonstrate the effectiveness of our approach by illustrating how incorrect feature interactions among various telecommunications services can be detected. Section 6 concludes our paper and suggests promising topics for further research.

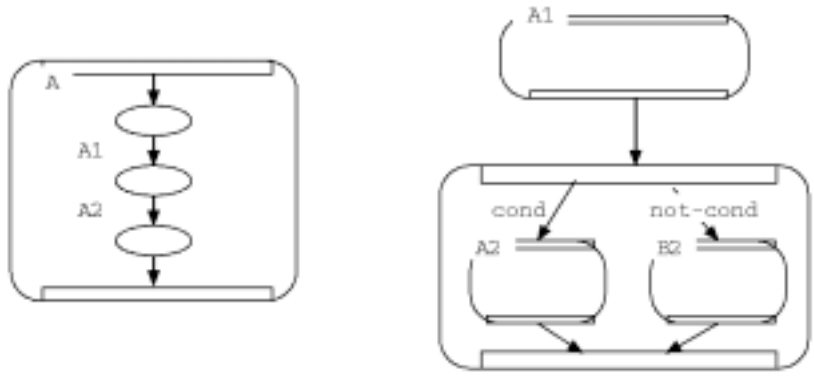
## 2 Related Work

Several researchers have tried to formalize the informal aspects of use cases. While earlier approaches focused primarily on developing the formal semantics of use cases, recent proposals have focused more on developing techniques to integrate and perform analysis on a set of use cases.

As an example of the former, Hsia et al. [12] used a BNF-like grammar to formally describe use cases. A graphical and tree-like representation, called the conceptual state machine, is used. This approach is effective when applied to a small number of relatively simple use cases. Specification based on multiple viewpoints can, in principle, be supported by developing a more complex grammar. However, this is likely to be too cumbersome to be useful on industrial applications where frequent changes to requirements are expected to occur and where iterative and incremental requirements elicitation techniques are needed.

Andersson and Bergstand [1] used Message Sequence Charts (MSCs). MSCs are frequently used to state the requirements for telecommunications software. The current MSC standard, MSC'96 [15], provides several features aimed at enhancing the expressiveness of individual MSCs. Examples include constructs to specify the conditional, iterative, or concurrent execution of MSC sections. Anticipated exceptions and required system responses can be

2. Use cases usually describe specific but partial event sequences that are closely related. For a use case $U_i$, other use cases may be regarded as specifying additional *constraints* under which $U_i$ may take place.




(a) (b)

Fig. 2. Handling overlapping scenarios in Glinz's statecharts approach. (a) A sequential statechart. (b) Handling overlapping scenarios.

specified, too. An MSC-based approach has advantages over the grammar-based approach in terms of scalability and understandability.

Dano et al. [10] used tabular notations as well as Petri nets as intermediate representations to derive the dynamic behavioral specification for an object from use cases. One or more tables corresponding to a use case are initially developed. These tables, which are combined into a Petri net, can properly specify the dynamic behavior of an object. The analysis techniques on interactions and dependencies among several objects remain unsupported, which is true for all of the previous approaches as well.

In contrast, several researchers have proposed analysis techniques for a set of interacting use cases. Glinz [11] used statecharts to represent use cases. The relationships among use cases are represented using one of the following constructs: sequence, alternation, iteration, or concurrency. This approach assumed a disjointedness among the use cases and did not support overlapping scenarios where the same event sequences appear in multiple use cases. That is, when overlapping scenarios are later identified, existing use cases (and, therefore, corresponding statecharts) have to be modified to maintain the disjointedness. For example, statecharts connected by a sequence relation may need to be further decomposed into more detailed statecharts connected by sequence as well as by alternation constructs, as illustrated in Fig. 2. This approach does not satisfy the insensitivity property unless all of the overlapping scenarios are known in advance. This "forced" (and potentially unnatural) modification of statecharts is not supportive of traceability.

Other approaches used to analyze dependencies include timed automata [28], finite state automata [23], and MSCs [22]. They are adequate for describing use cases individually and can even analyze the interactions among a small number of use cases. However, the larger the number of use cases there are to analyze, the more difficult it becomes to grasp and analyze the global system behaviors since the brute-force approach of considering all possible combinations quickly leads to the state explosion problem.

## 3 Constraints-Based Modular Petri Nets (CMPNs)

### 3.1 Motivations

Petri nets have been used extensively and successfully in various applications such as protocol or performance analysis. The well-known strengths of Petri nets include their visual and easily understandable representation, their well-defined semantics, their ability to model concurrent and asynchronous system behavior, the variety of mature analysis techniques they offer (e.g., reachability, deadlock, safety, invariant, etc.), and the availability of software tools to assist modeling and analysis. Several extensions (e.g., time, probability, etc.) have been proposed to the basic formalism. Research trends in Petri nets include compositional modeling and analysis [31] in which various subsystems are modeled separately and behavioral analysis performed collectively.

We strongly believe Petri nets to be well-suited to overcoming the limitations caused by the informal aspects of the use case approach. Use cases can be conceptually considered as a set of interacting and concurrently executing threads, and if use cases can be transformed into a Petri nets-based formalism, existing analysis techniques can be readily applied to detect anomalies.

Unfortunately, both classical Petri nets, known as place/transition nets (P/T nets) [27], and such high-level Petri nets as colored Petri nets (CP-nets) [19] are inadequate for our purpose. First, they do not provide adequate language constructs supporting modular and incremental specifications. Fig. 3 illustrates a P/T net model for the interactions between basic call processing (BCP) and call forwarding (CF) in a telecommunications model that will be used throughout this paper. The prefixes *o-*, *t-*, and *cf-* indicate activities involving originating, terminating, and forwarding parties, respectively. This model, which is quite simplistic compared to industrial applications, is not only difficult to understand but also quite sensitive to changes. Suppose that we wished to integrate another telecommunications service, say call waiting, with this model or that we needed to replace an existing call forwarding service with an enhanced version. In either case, the model would need to be extensively modified. Second, it is difficult to perform

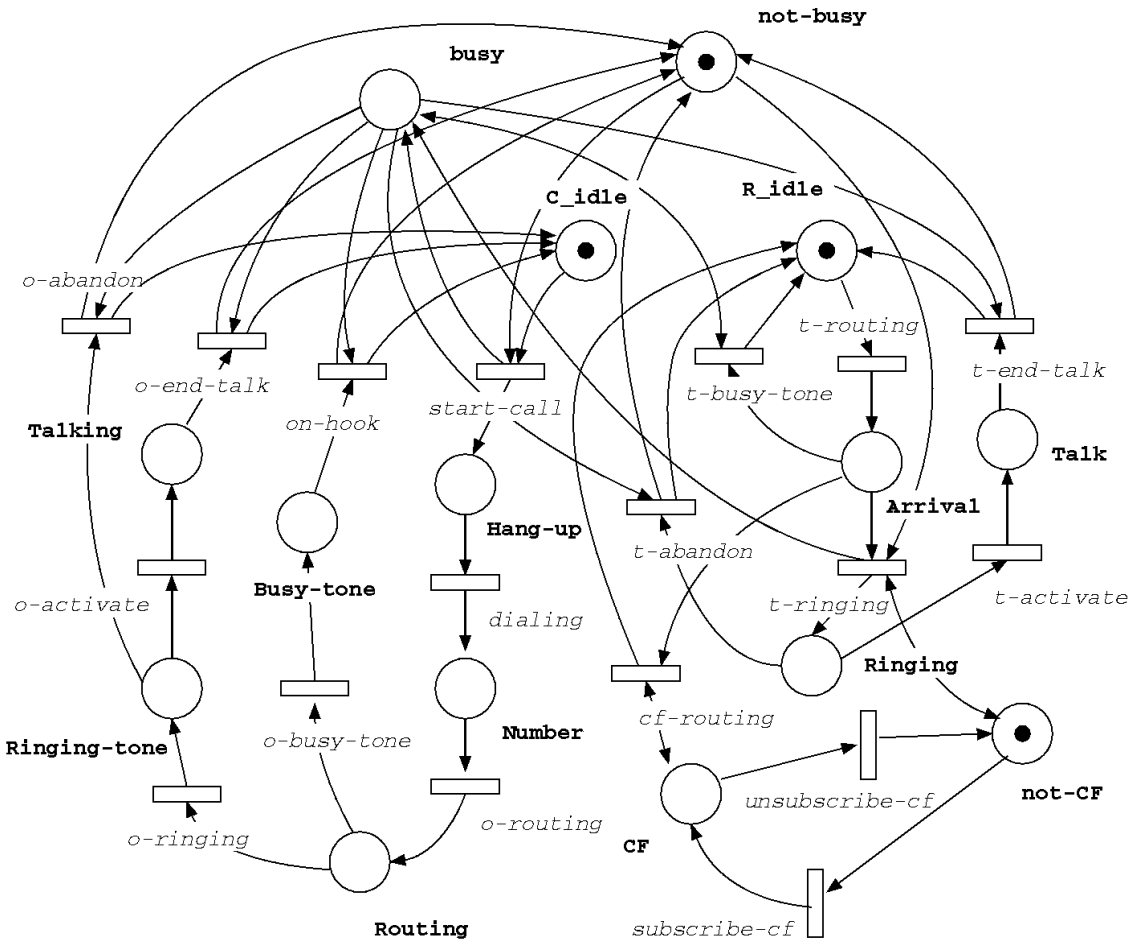



Fig. 3. P/T net for basic call processing and call forwarding.

selective behavioral simulation tailored to satisfy the interests of specific users. Third, traceability is poorly supported since all use cases are blended into a global P/T net model.

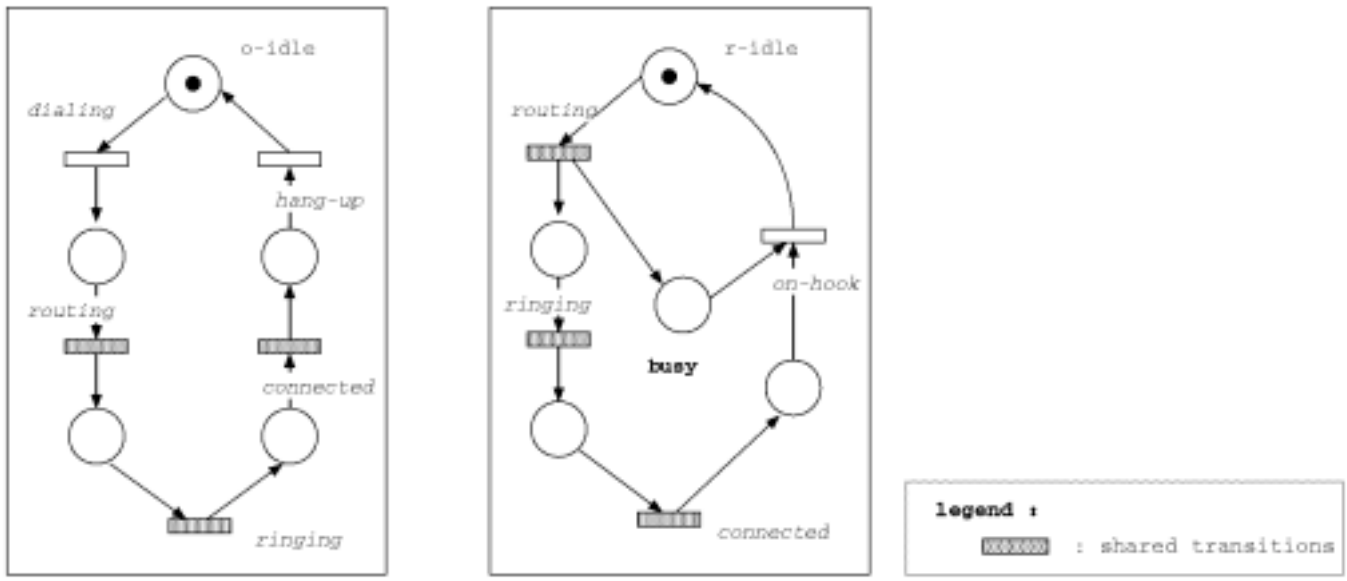
High-level Petri nets significantly improve the expressiveness of P/T nets while providing the same level of analytical capability. CP-nets [19], perhaps the most widely used high-level Petri nets formalism in industry, have been successfully used to model and analyze several systems, such as the NORAD command center and an electronic funds transfer system. As opposed to P/T nets, where all tokens are of Boolean type, CP-net tokens can be of arbitrary color (type) and complexity. Because CP-net notations are based on a functional programming language SML [29], arc expressions allow concise specification of complex token manipulations.

To support modularity, CP-nets provide such features[3] as substitution transitions and fusion places. Substitution transitions, a notational convenience designed to allow modular representation of large and complex models, utilize the port concept. When a substitution transition is declared, its input and output places are considered as ports. The substitution transition is then further expanded on another page and additional (and internal) places and transitions can be declared as needed. However, all "atomic" transitions model events at the same level of abstraction, although they may appear on different and hierarchically organized pages.

Fusion places are used to avoid the clustering of too many input and output arcs. Fusion places appearing on different pages are considered the same and the firing semantics of CP-nets are unaffected. While clearly useful in improving the understandability of CP-net models, fusion places alone, from the viewpoint of formalizing use cases, are insufficient to overcome the weaknesses identified earlier. For example, if we were to model use cases $U_1$ through $U_6$ (see Fig. 9) in CP-nets, the Petri nets corresponding to the first three use cases would have to be combined into one because the transition *o-ringing* would be common among them.

Shared transitions are often used when introducing modularity to Petri nets [9], [7], [8], [6]. In this approach, the subsystems are modeled separately and assumed to operate independently and concurrently unless their activities are synchronized at shared transitions. Fig. 4 illustrates how shared transitions are used to model the behavior of the originating and receiving parties during call connection. The line becomes busy on the receiving side as soon as

3. Although a paper by Huber et al. [13] briefly introduces the concept of fusion transition (also referred to as shared transition in our paper) as another feature to support modularity, this feature is left undefined in his authoritative book on CP-nets [19] and remains unsupported by the Design/CPN CASE tool [30]. Therefore, it seems fair to conclude that fusion transitions are currently not part of CP-net formalism.




(a) (b)

Fig. 4. A transition sharing model. (a) Originating party. (b) Receiving party.

routing is completed but prior to the generation of the first ring. This status is indicated by having a token in the place *busy*. Suppose that we chose to introduce another use case $U_{busy}$ specifying what to do with a busy line. The model corresponding to $U_{busy}$ would need to reference the status of the place *busy*. If only transitions can be shared, one of the following approaches must be taken:

- Combine all the Petri nets corresponding to use cases which must share system variables. The combined model can become quite complex, and traceability is poorly supported.
- Further decompose existing models so that the places corresponding to shared system variables appear together on the same, but separate, model fragment. This approach is undesirable since a Petri net corresponding to a use case may end up being scattered on multiple pages. This approach is also quite sensitive to changes. Suppose that we were later told to drop the use case $U_{busy}$ that had previously forced a rather unnatural decomposition of the model. The model fragments might need to be combined again, and frequent use case changes could easily result in a chaotic requirements engineering process. Since requirements rarely remain frozen in practice, it is obvious that shared transitions alone are inadequate.

Another trend in Petri nets research is to introduce object-oriented concepts. Examples include PROTOB [3], LOOPN [20], and PAM [2]. The required behavior of each object is modeled in a separate Petri net, and additional constructs are used to describe the relationships among the objects. Unfortunately, objects and use cases capture requirements at different levels of abstraction and often state requirements based on different viewpoints. That is, an object is likely to have several use cases associated with it. A use case specification, on the other hand, may include the interaction of several objects. Furthermore, message passing among objects does not naturally represent the sharing of system status values or synchronization of events among use cases.

Critical examination of various proposals on Petri net formalisms reveals that they are inadequate to formalizing the informal aspects of the use case approach and to satisfying such properties as partiality and insensitivity to changes. CMPNs, our proposed extension to P/T nets, are designed to bridge such gaps.

### 3.2 CMPNs: Definitions

Constraints-based Modular Petri nets (CMPNs) consist of a set of constraint nets $C_n$. A constraint net, modeling an individual use case, consists of internal structures as well as an external interface. CMPNs are structured on a two-level hierarchy to naturally reflect the organization of the requirements specification stated as a collection of use cases. It is true that the requirements for extremely large and complex systems may require a use case organization of more than two levels of hierarchy. For example, use cases corresponding to each subsystem might be grouped separately. However, such grouping is designed to improve understandability and maintainability and has no direct impact on behavioral analysis.

The internal structures of a constraint net are the same as those of a P/T net, and the external interface specifies its name. Shared places and transitions[4] can be considered parts of the external interface. However, they can be automatically identified and, therefore, need not be declared explicitly and redundantly. Fig. 5 is an example of CMPNs which consist of six constraint nets.

DEFINITION 1. *For a character set* $\Sigma$, *a constraint net is a 6-tuple* $C_n = (P, T, F, W, L, M_0)$, *where*

4. Shading is applied in our paper to visually highlight shared places and transitions.

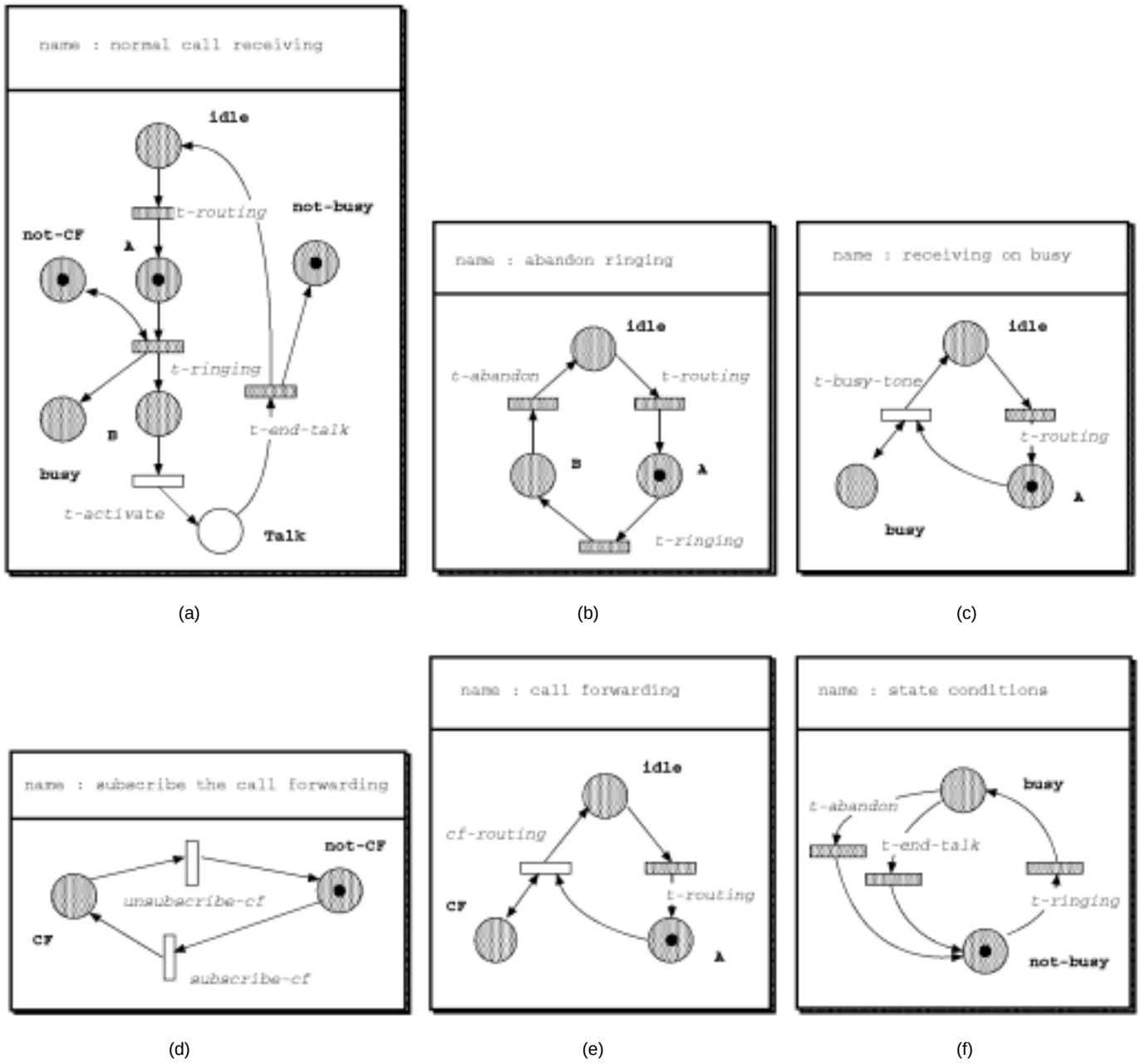



Fig. 5. CMPNs for receiving functions of the BCP. (a) Cn1, (b) Cn2, (c) Cn3, (d) Cn4, (e) Cn5, (f) Cn6.

- *P, T, F, and W are a set of places, transitions, arcs, and weights associated with arcs, respectively, whose definitions are the same as the ones for standard P/T nets* [27],
- $L : P \cup T \rightarrow \Sigma^{+}$ *is a label function that associates a distinct label taken from strings* ($\Sigma^{+}$) *with each place and transition of P and T, and*
- $M_0 : P \rightarrow Nat$ *is the initial marking.*

It should be noted that labels associated with a constraint net, if explicitly specified, are required to be distinct.

DEFINITION 2. *Let* $\left\{C_{n_i} \mid i = 1 \ldots n\right\}$ *be a set of constraint nets,* Node *be a set of all the places and transitions in* $C_{n_i}\left(\bigcup_{i=1}^{n} P_i \cup \bigcup_{i=1}^{n} T_i\right)$. *R(x, y), the equivalence relation in node indicating that x and y have the same label, is defined as follows:*

$$R(x, y) \equiv L(x) = L(y).$$

*An equivalence class of* $x\left(\in \bigcup_{i=1}^{n}\left(P_i \cup T_i\right)\right)$ *modulo R is defined as follows:*

$$\hat{x} = \left\{y \in Node \mid (x, y) \in R\right\}.$$

This definition declares that places and transitions whose labels appear more than once are considered shared and that they should be treated as one.

DEFINITION 3. *A Constraints-based Modular Petri net is defined as a set of constraint nets,* $M_N = \left\{C_{n_i} \mid i = 1 \ldots n\right\}$ *satisfying the following conditions:*

- *$P_i$, $T_i$, $F_i$, and $W_i$ should be disjoint for all the $C_{n_i}$,*
- *The same label should not be used for both places and transitions:*

$$\forall C_{n_i}, \forall C_{n_j},\ \exists p \in P_i,\ \exists t \in T_j \text{ such that } L_i(p) = L_j(t),$$

- *$W_i$ should be consistently defined in $C_{n_i}$ as follows:*

$$\forall (p,t) \in F_i, \forall (p',t') \in F_j \text{ such that}$$
$$p' \in \hat{p} \text{ and } t' \in \hat{t} \Rightarrow W_i(p,t) = W_j(p',t'),$$

- *The initial markings for shared places should be the same:*

$$\forall C_{n_i}, \forall p_i \in P_i, \forall p_j \in P_j,$$
$$L_i(p_i) = L_j(p_j) \Rightarrow M_{0_i}(p_i) = M_{0_j}(p_j).$$

In the above CMPNs definition, both places and transitions may be shared. Our definition assumes that places and transitions whose labels appear in two or more constraint nets are shared. It also requires that weights associated with shared arcs, as well as the initial placement of tokens for shared places, be the same.

Whenever a new use case is added, its behavior is modeled separately, thus providing superior traceability when compared to other approaches. In such cases, the existing CMPN structure remains the same although some of places and transitions may become shared to accurately reflect the dependencies between the existing use cases and the newly introduced use case. In some cases, shared places may need to be added.

DEFINITION 4. *Let $M_N = \{C_{n_i} \mid i = 1 \ldots n\}$ be a CMPN, then*

- *$L_g = \bigcup_{i=1}^{n} L_i$ is the global label function,*
- *$M_g = \bigcup_{i=1}^{n} M_{0_i}$ is the global marking, and*
- *$W_g$ is the global weight function, $\forall x, y \in \bigcup_{i=1}^{n} (P_i \cup T_i)$,*

$$W_g(\hat{x}, \hat{y}) =$$
$$\begin{cases} W_i(x', y'), \text{ if } \exists (x', y') \in F_i \text{ such that } x' \in \hat{x} \text{ and } y' \in \hat{y} \\ 0, \text{ otherwise.} \end{cases}$$

In a given constraint net $C_{n_i}$, the decision on whether or not a shared transition $t_s$ is locally enabled can be made without having to consult other constraint nets in which $t_s$ occurs. If all the preconditions needed for the transition $t$ in a constraint net to occur are met [27], it is said to be *M-enabled*.

DEFINITION 5. *Let $M_N = \{C_{n_j} \mid j = 1 \ldots n\}$ be a CMPN model. In a constraint net $C_{n_j}$,*

- *An internal transition $t_i \in T_j$ $(|\hat{t}_i| = 1)$ is $M_g$-*enabled *iff $t_i$ is M-enabled.*
- *A shared transition $t_s \in T_j$ $(|\hat{t}_s| > 1)$ is $M_g$-*enabled *iff $\forall C_{n_k}$ such that $L_j(t_s) = L_k(t')$, $t'$ is M-enabled in $C_{n_k}$.*

If an internal transition $t_i$ is M-enabled in $C_{n_i}$, it is also enabled in the global marking, since $t_i$ appears in no other constraint nets. On the other hand, a shared transition $t_s$ is enabled only when it is enabled in all the constraint nets in which the label of $t_s$ appears. For example, take the global marking shown in Fig. 5 as the current marking:

$$M_{g_1} = M_1 \cup M_2 \cup M_3 \cup M_4 \cup M_5 \cup M_6, \text{ where}$$
$$M_1{}^5 = (A, \textit{not-CF}, \textit{not-busy}),\ M_2 = (A),\ M_3 = (A),$$
$$M_4 = (\textit{not-CF}),\ M_5 = (A), \text{ and } M_6 = (\textit{not-busy}).$$

The enabled transitions are *subscribe-cf* and *t-ringing*, where the former is an internal transition and the latter is shared. When firing an internal transition $t_i$, the token movements are not necessarily limited to the constraint net in which $t_i$ occurs. The transition $t_i$ may either deposit or consume tokens to or from the shared place $p_s$. Since the $p_s$ appearing on different constraint nets is actually the same place, token manipulations caused by $t_i$ need to be reflected to other constraint nets as well so that consistency in global markings is maintained. For example, when the internal transition *subscribe-cf* is fired, the updated global marking $M_{g2}$ is defined as follows:

$$M_{g_2} = M_1 \cup M_2 \cup M_3 \cup M_4 \cup M_5 \cup M_6, \text{ where}$$
$$M_1 = (A, \textit{not-busy}),\ M_2 = (A),\ M_3 = (A),$$
$$M_4 = (\textit{CF}),\ M_5 = (A, \textit{CF}), \text{ and } M_6 = (\textit{not-busy}).$$

It should be noted that the configuration of $C_{n_1}$ was changed from (*A*, *not-busy*, *not-CF*) to (*A*, *not-busy*) although no transitions in $C_{n_1}$ were fired. The firing of a shared transition is similarly carried out. For example, the next global marking $M_{g_3}$ reached after firing *t-ringing*, given the current global marking $M_{g_1}$, is defined as follows:

$$M_{g_3} = M_1 \cup M_2 \cup M_3 \cup M_4 \cup M_5 \cup M_6, \text{ where}$$
$$M_1 = (\textit{not-CF}, \textit{busy}, B),\ M_2 = (B),\ M_3 = (\textit{busy}),$$
$$M_4 = (\textit{not-CF}),\ M_5 = (),\ \text{and } M_6 = (\textit{busy}).$$

The firing semantics of CMPNs can be formally defined as follows:

DEFINITION 6. *Let $M_N = \{C_{n_i} \mid i = 1 \ldots n\}$ be a CMPN model. An $M_g$-enabled transition $t \in T_i$ yields new markings $M_j'$ in all the $C_{n_j}$ and $M_g'$ as follows:*

- *$\forall p \in \bigcup_{j=1}^{n} P_j$,*

5. Markings for constraint nets are defined as functions. For example, mathematically correct representation of $M_1$ would be {(*idle*, 0), (*A*, 1), (*not-CF*, 1), (*busy*, 0), (*B*, 0), (*Talk*, 0), (*not-busy*, 1)}. However, we use a simplified notation, (*A*, *not-CF*, *not-busy*), to enhance readability by listing only the labels for the places containing tokens.

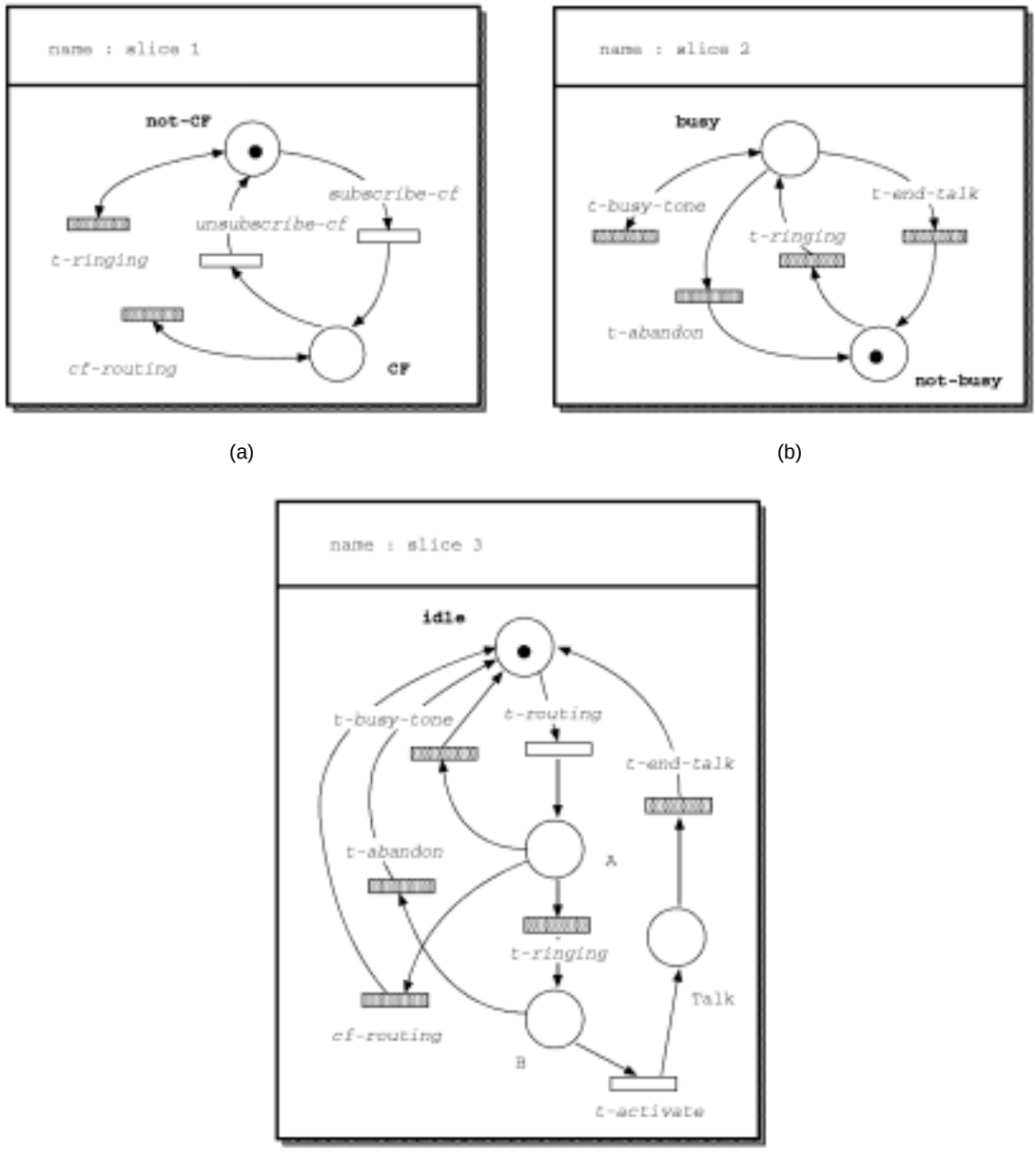



Fig. 6. Minimal CMPN slices for the CMPN model shown in Fig. 5. (a) Slice 1. (b) Slice 2. (c) Slice 3.

$$M'_j(p) = \begin{cases} M_j(p) - W_g(\hat{p},\hat{t}) \ \textit{iff} \ L_g(p) \in \left({}^\circ t - t^\circ\right), \\ M_j(p) + W_g(\hat{t},\hat{p}) \ \textit{iff} \ L_g(p) \in \left(t^\circ - {}^\circ t\right), \\ M_j(p) - W_g(\hat{p},\hat{t}) + W_g(\hat{t},\hat{p}) \ \textit{iff} \ L_g(p) \in {}^\circ t \cap t^\circ, \\ M_j(p), \ \textit{otherwise}, \end{cases}$$

- $M'_g(p) = \bigcup_{j=1}^{n} M'_j(p)$.

In the definition given above, ${}^\circ t$ and $t^\circ$ represent the label sets of pre- and post-places globally related to the transition $t$, respectively. For example, in Fig. 5, ${}^\circ$*t-ringing* = {*A*, *not-busy*, *not-CF*} and *t-ringing*$^\circ$ = {*busy*, *B*, *not-CF*}.

The reachability analysis of CMPNs is straightforward and can be easily automated. A brute-force approach is to combine all the constraint nets, to generate an equivalent P/T net based on the concept of observational equivalence [25], and to apply known analysis techniques. This approach, though feasible, is clearly undesirable because of state explosion.

The operational semantics of CMPNs do not provide the ability to perform a compositional analysis directly on CMPNs due to the use of shared places. In order to overcome such a limitation and to reduce the complexity associated with behavioral analysis, we propose utilizing the concept of Petri net slices. CMPN slices are defined as a restricted CMPN in which place sharing does not occur. Reference [21] describes a slicing algorithm. Intuitively stated, the algorithm first computes a set of places (e.g., {*CF, not-CF*}, {*busy, not-busy*}, etc.) in which the number of tokens does not change during the transition firings. This concept is known as the S-invariants [27]. The slices are computed by selecting those elements in the set containing the least

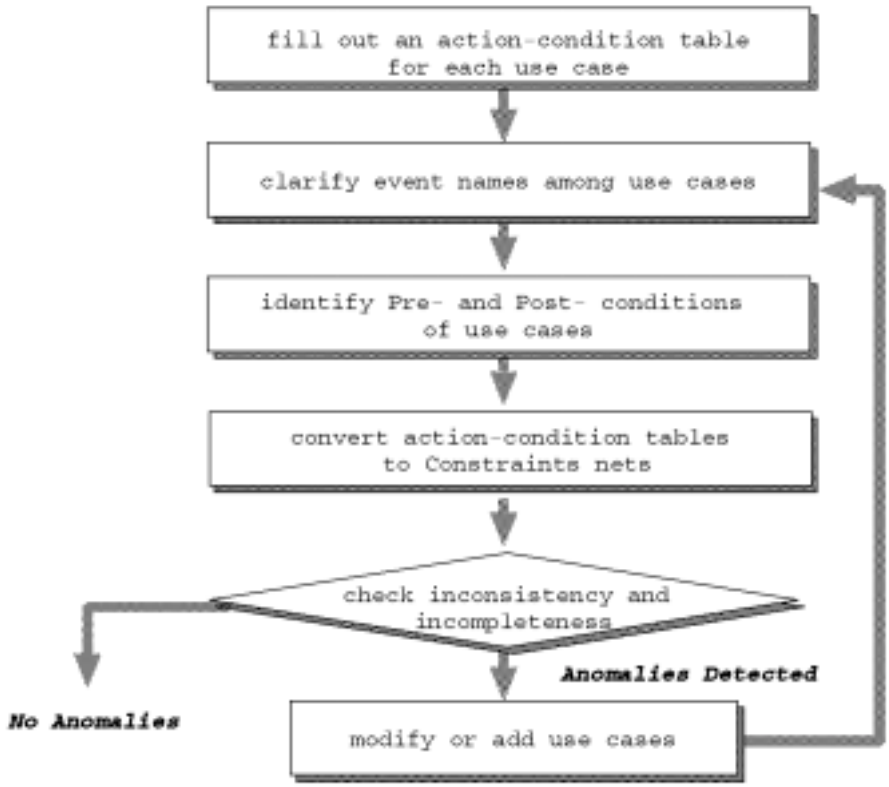



Fig. 7. Use cases sonversion to CMPNs and analysis.

number of places until all the places in the CMPNs are covered, if possible.

Fig. 6 illustrates the results of applying the slicing algorithm to the BCP CMPN shown previously in Fig. 5. For example, slice 1, containing two places *CF* and *not-CF*, is initially computed along with the transitions connected to their places by input and output arcs. Slice 2, containing two places, is obtained next. It should be noted that transition sharing is allowed in the CMPN slices, as indicated by the appearance of the transition *t-ringing* in both slices.

## 4 Use Case Integration Based on CMPNs

Use cases are generally stated in natural languages and need to be converted to CMPNs. In this section, we propose a systematic procedure for performing such a conversion. Fig. 7 illustrates the procedure, which involves the use of action-condition tables. This conversion process cannot be completely automated and interactions between users and domain experts will still be needed.

This procedure is illustrated using six use cases dealing with basic call processing (BCP) in telecommunications software:

- Caller
  - $U_1$ = (off hook; dial number; routing; wait for response; call connected; on hook) /* normal conversation */
  - $U_2$ = (off hook; dial number; routing; wait for response; on hook) /* no one answers */
  - $U_3$ = (off hook; dial number; routing; other party is busy; on hook) /* line is busy */
- Callee
  - $U_4$ = (call arriving; phone ringing; off hook; on hook) /* normal receiving */
  - $U_5$ = (call arriving; phone ringing; the ringing stops) /* abandoned call */
  - $U_6$ = (call arriving; send busy signal) /* busy handling */

**Step 1: Fill out the action-condition table.** Tabular notations have been previously used to annotate use cases [26].[6] A sequence of actions included in the use case is specified. To improve understandability, it is customary to assign each table a name and to provide an informal description. Table 1 is an example of the action-condition table corresponding to the use case $U_4$. In our notation, columns to specify pre- and post-conditions associated with actions are added.

**Step 2: Clarify event names.** Action names initially given in the use case description may need to be changed. For example, different users may use different terms to indicate the same action. Similarly, the same name might have been mistakenly used to refer to distinct actions. For example, action *on hook* in $U_1$ occurs when a call is successfully completed. On the other hand, action *on hook* in $U_2$ occurs when the caller hangs up before response. Although both actions represent the same physical movement, they occur in different situations and must be treated as such. Last, if action names are stated at different levels of abstraction, they need to

6. Since use case dependency analysis involves both users and domain experts, it is beneficial to adopt notations both groups are familiar with.

TABLE 1
AN ACTION-CONDITION TABLE FOR NORMAL CALL RECEIVING

| Name : call receiving : $U_4$ | | | |
|---|---|---|---|
| Informal Description:<br>When a call arrives, a user picks up the phone,<br>engages in a conversation, and finally hangs up. | | | |
| Actions | Event Names | **Preconditions** | **Postconditions** |
| call arriving<br>phone ringing<br>off hook<br>on hook | | | |

TABLE 2
AN ACTION-CONDITION TABLE FOR NORMAL CALL RECEIVING

| Name : call receiving: $U_4$ | | | |
|---|---|---|---|
| Informal Description:<br>When a call arrives, a user picks up the phone,<br>engages in a conversation, and finally hangs up. | | | |
| Actions | **Event Names** | Preconditions | Postconditions |
| call arriving<br>phone ringing<br>off hook<br>on hook | **t-routing**<br>**t-ringing**<br>**t-activate**<br>**t-end-talk** | | |

TABLE 3
AN ACTION-CONDITION TABLE FOR NORMAL CALL RECEIVING

| Name: call receiving: $U_4$ | | | |
|---|---|---|---|
| Informal Description:<br>When a call arrives, a user picks up the phone,<br>engages in a conversation, and finally hangs up. | | | |
| Actions | Event Names | **Preconditions** | **Postconditions** |
| call arriving<br>phone ringing<br>off hook<br>on hook | t-routing<br>t-ringing<br>t-activate<br>t-end-talk | | **busy**<br>**not-busy** |

be modified. Otherwise, it would be difficult to perform meaningful dependency analysis. Table 2 shows the action-condition table of $U_4$ after clarifying the event names.

**Step 3: Identify pre- and post-conditions.** Actions specified in use cases are carried out only when a specific set of conditions is satisfied. The identification of pre- and post-conditions proceeds in two steps. First, users identify relevant state variables[7] and specify preconditions for use cases as a predicate involving the state variables. For example, in $U_6$, although not stated explicitly, the line must be busy for $U_6$ to occur. Second, post-conditions are specified by examining the effects each use case has on the set of known state variables. See Table 3 for an example.

**Step 4: Convert action-condition tables to CMPNs.** Each action-condition table is converted to a constraint net. Since use cases are considered concurrent units of system's functionalities, each has its own control thread. Fig. 8a shows a constraint net converted from Table 3. Events occurring in multiple use cases, identified in Step 2, are declared as shared transitions. Examples include transitions *t-routing* and *t-ringing*, as shown in Fig. 8a and 8b.

When converting each use case to a constraint net, output places connected to a shared transition may need to be shared if any of the following conditions holds:

- Successor events are the same. This case occurs when multiple use cases share a common sequence of events as shown in the transitions *t-routing* and *t-ringing* appearing in Fig. 8. Since the intermediate states, $A1$ and $A2$ in Fig. 8a and 8b, respectively, are apparently the same, they are converted to a shared place with the name *Arrival* (see Fig. 8c and 8d).
- Distinct successor events occur, but they occur selectively. This case occurs if the domain experts realize that the two use cases need to be executed in a mutually exclusive manner as is the case for the transitions *t-activate* and *t-abandon* shown in Fig. 8a and 8b, respectively. In order to preserve the executing threads of selective use cases, not only the immediate predecessor places (*Ring4* and *Ring5* in our example) but also the initial places (*Start4* and

7. It should be noted that global variables are assumed to be of the Boolean type in this paper to simplify analysis and that such restrictions can be relaxed.

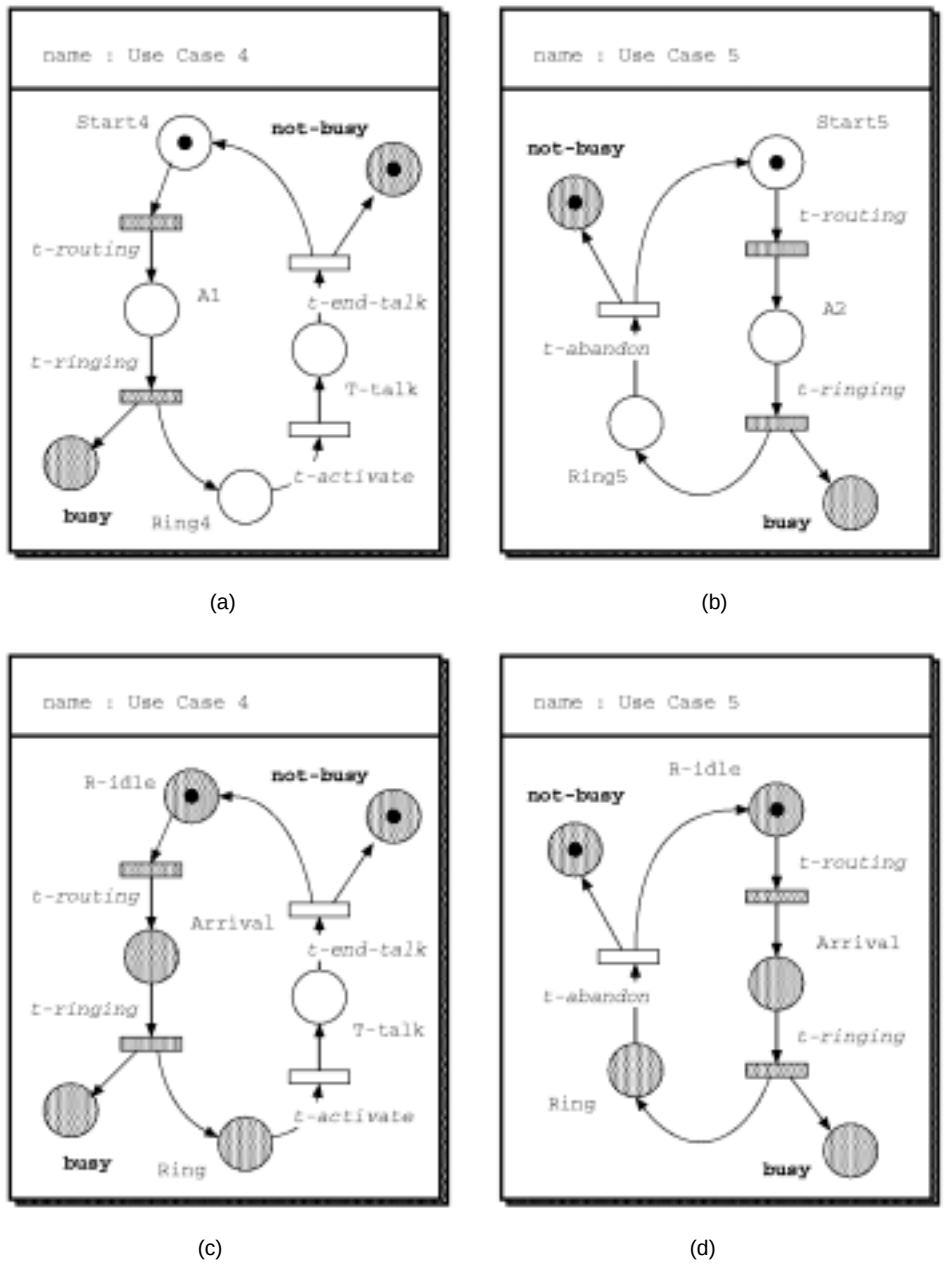



Fig. 8. Converting action-condition tables to constraint nets. (a) A constraint net for U4. (b) A constraint net for U5. (c) A constraint net for U4. (d) A constraint net for U5.

*Start5*) are declared as shared places and renamed as *Ring* and *R-idle*, respectively (see Fig. 8c and 8d).

Transitions, internal or shared, may reference or update the status of binary state variables. Such conditions are declared as a special type of shared place which we call a *toggle place*. *Busy* and *not-busy*, shown in Fig. 8a, are examples. A toggle place consists of a pair of places where one place is the negation of the other. Whenever a token is deposited to a toggle place, a token is automatically removed from the counterpart, and vice versa. An example of a toggle place is *busy* and *not-busy*. A formal definition of a toggling constraint net preserving the consistency of the toggle place is given as follows:

DEFINITION 7. *Let* $M_N = \left\{ C_{n_i} \mid i = 1 \ldots n \right\}$ *be a CMPN. A toggling constraint net* $C_n = (P, T, F, W, L, M)$ *for p satisfies the following:*

- $P = \{p, not\text{-}p\}$,
- $T = \left\{ L_g(t) \middle| \, t \in \bigcup_{i=1}^{n} T_i, p \text{ or } not\text{-}p \in \left( {}^{\circ}t \cup t^{\circ} \right) - \left( {}^{\circ}t \cap t^{\circ} \right) \right\}$
- $F = \{(p, t) \in P \times T \mid p \in (t^{\circ} - {}^{\circ}t), t \in T\} \cup \{(t, p) \in T \times P \mid p \in (t^{\circ} - {}^{\circ}t), t \in T\} \cup \{(not\text{-}p, t) \in P \times T \mid not\text{-}p \in (t^{\circ} - {}^{\circ}t), t \in T\} \cup \{(t, not\text{-}p) \in T \times P \mid not\text{-}p \in (t^{\circ} - {}^{\circ}t), t \in T\}$
- *L is a label function:* $\forall x \in P \cup T, L(x) = x$
- $W : F \rightarrow Nat$ *is a set of weight functions:* $\forall (x, y) \in F, W(x, y) = 1$
- $M_0$ *is the initial marking:* $\forall p' \in P, M_0(p') = M_g(p')$.

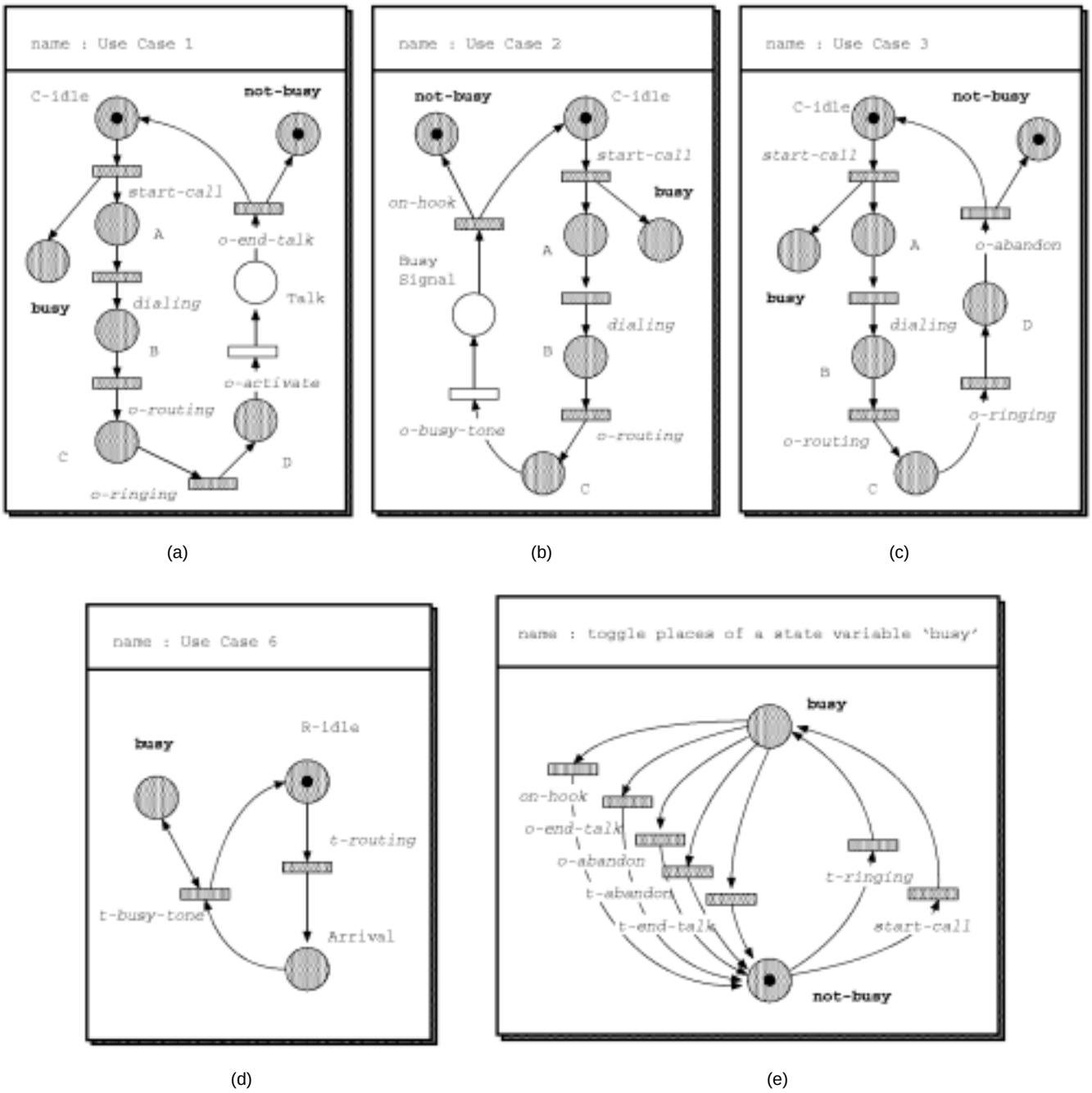



Fig. 9. CMPNs of basic call processing. (a) A constraint net for U1. (b) A constraint net for U2. (c) A constraint net for U3. (d) A constraint net for U6. (e) A constraint net for toggle places.

Fig. 9a, 9b, 9c, and 9d are obtained by applying the procedure described above to use cases $U_1$ through $U_3$ and $U_6$, respectively. Fig. 9e illustrates how the value of the toggle place *busy* is manipulated by various transitions. It is worth noting that the constraint net representing the toggle place manipulation can be automatically generated.

Once use cases are converted to the CMPN, Petri nets analysis techniques can be used to detect such anomalies as incompleteness or inconsistency. Should such analysis techniques detect flaws, the use cases would need to be revised and another iteration of CMPN analysis undertaken.

## 5 CMPNs Analysis: Consistency and Completeness

A variety of Petri nets analysis techniques can be applied to a CMPN model to detect such errors as inconsistency or incompleteness. Simulation can potentially reveal the incorrect behavior of use cases. As noted earlier, users are most likely to be interested in analyzing the behaviors of only a selected subset of use cases and the interactions among them. A significant advantage offered by our approach is that a simulation can be tailored to the user's specific viewpoints and its complexity minimized. That is, it is sufficient to perform simulation involving only those portions of CMPN slices containing places or transitions corresponding

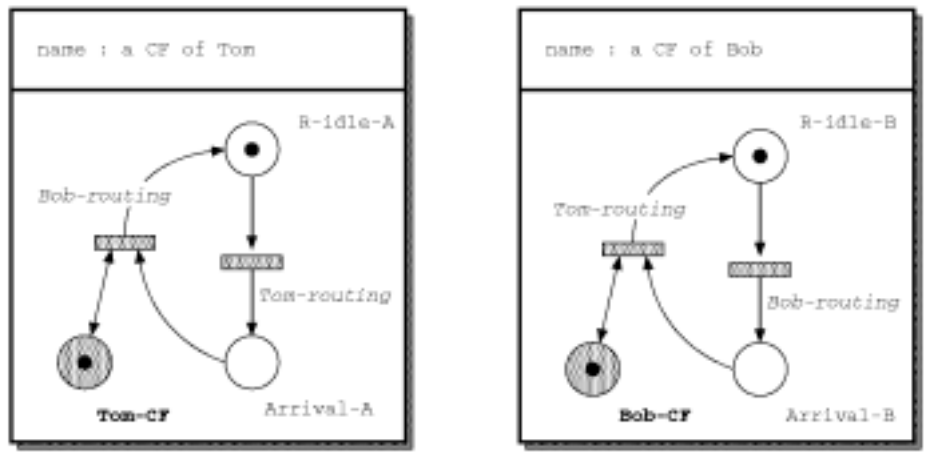



Fig. 10. Deadlock in circular call forwardings.

to the use cases in which a user or users are particularly interested. The simulation of the rest of the CMPN slices can be hidden. Likewise, the consistency and completeness analysis can be applied to either the CMPN or the CMPN slices.

A CMPN model is said to be inconsistent if there exists a set of transitions that are never enabled. This type of flaw is analogous to unreachable code in programs. Since use cases are expected to reflect genuine needs, it is reasonable to require that CMPNs do not contain transitions that are never enabled.

Another type of inconsistency occurs if there are deadlocks. Take the call forwarding (CF) service as an example. In our example, shown in Fig. 10, incoming calls are unconditionally[8] directed to another phone. Circular forwarding clearly doesn't make sense, and such anomalies are detected by applying well-known deadlock detection algorithms.

Criteria or heuristics to detect the incompleteness of CMPNs are summarized as follows:

**Nondeterminism:** If the reachability analysis reveals the presence of nondeterministic execution paths, the CMPN may be incomplete because users may have forgotten to fully specify the constraints associated with the use cases. It must be emphasized that nondeterministic execution paths may have been introduced on purpose and that the final decision can be made only by the domain experts.

Consider the minimal slice set (Fig. 11) obtained from the CMPNs corresponding to BCP and CF services. The presence of a token in the *arrival* place in slice 3 indicates that three transitions, *t-busy-tone*, *t-ringing*, and *cf-routing* are simultaneously enabled. Since slice 2 indicates that the first two transitions are never enabled at the same time, we are left with two possibilities of nondeterministic execution between the following transition pairs: *(t-busy-tone; cf-routing)* and *(t-ringing; cf-routing)*. Close examination reveals that we have detected a flaw in the use cases. That is, the use cases for BCP and CF were *correct when analyzed in isolation*. However, when the CF service was introduced, the CMPN for BCP should have been modified to properly integrate the CF service. For example, a constraint net corresponding to the use case $U_4$ might have been modified so that the place *not-CF-on* could be connected to the transition *t-ringing* with both input and output arcs. That is, the event *t-ringing* should have been generated only when the call forwarding service was currently turned off.

Dependencies in telecommunications services can be quite subtle when diverse features are introduced. Conflicts may occur when a customer subscribes to several services. Similarly, conflicts may occur due to interaction among customers subscribing to the same or different sets of services. The detection of such interaction flaws, called the feature interaction problem [14], [24], [5] in the telecommunications industry, is known to be a difficult problem. Traditional techniques such as inspection can, in principle, detect such flaws. However, if minor changes to use cases occur frequently, as is likely in industrial projects, repeated manual inspection is highly unlikely to be cost-effective. On the other hand, the completeness analysis of CMPNs can be fully automated.

**Missing toggle place references:** Another criteria for completeness is that references to state variables, modeled as toggle places, must be complete. This is analogous to the requirements completeness criteria for reactive systems [18] in which the union of trigger conditions must always yield tautology. If a reference to the toggle place *CF-on* is made in a constraint net $C_{n_i}$, it is reasonable to expect that there must exist another constraint net which specifies what the system must do when such a condition is not satisfied.

**Toggle place values never modified:** It makes little sense if the values assigned to state variables are never changed during system operation. Since state variables are modeled as toggle places, CMPNs must contain transitions that are capable of removing or depositing a token from or to the toggle places, respectively. Otherwise, the

8. Modern telecommunications switches allow conditional (e.g., only when the line is busy or no one answers the phone within a specified duration) forwarding of calls. In order not to excessively complicate our example, we chose not to model the conditional call forwarding capability.

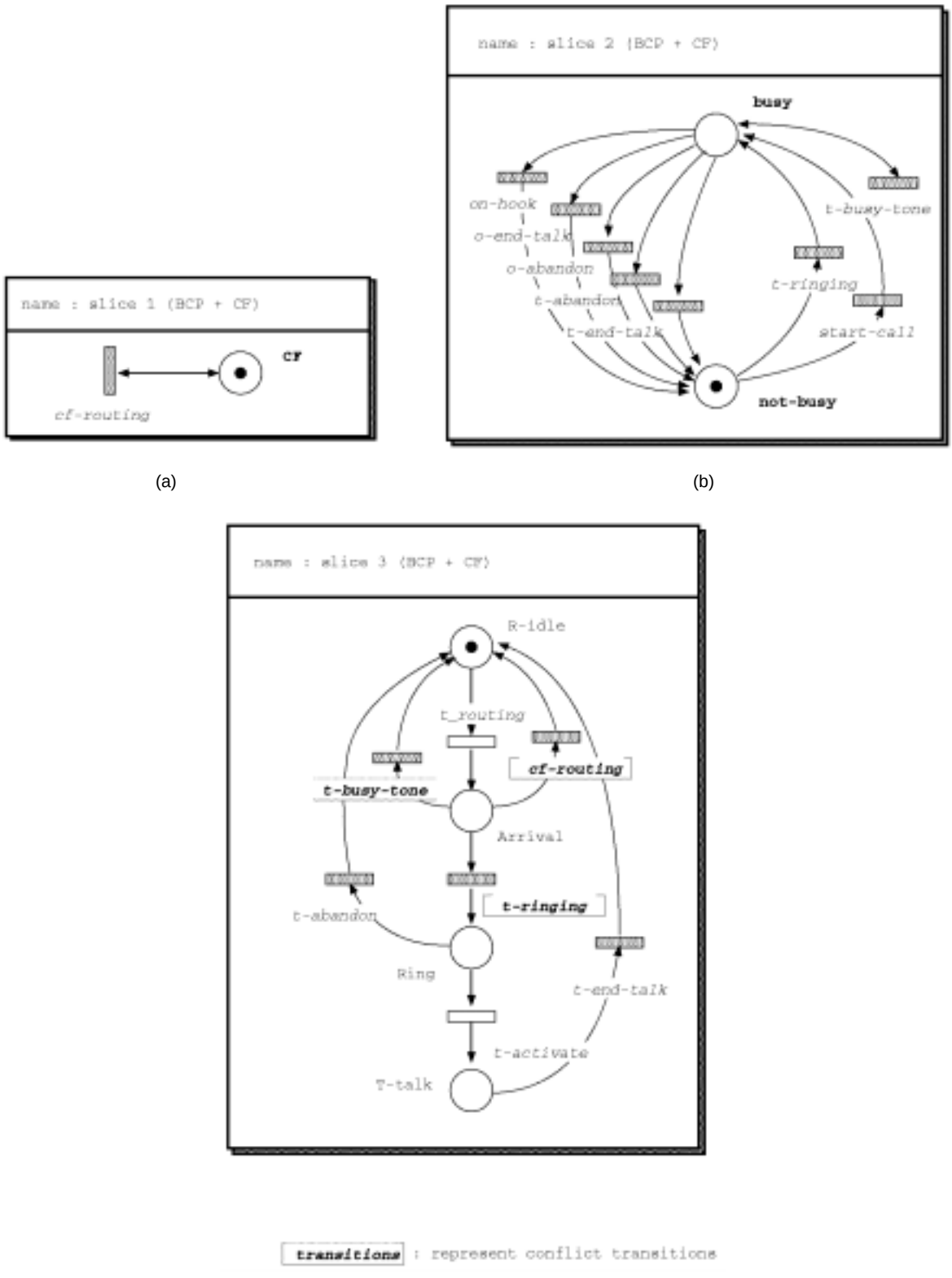



Fig. 11. Slice sets of basic call processing and call forwarding. (a) Slice 1. (b) Slice 2. (c) Slice 3.

CMPNs are surely incomplete. In our example, a constraint net for the CF specifies what happens when the service is activated. However, our CMPN does not specify when the CF service can be deactivated and how such deactivation affects the BCP. Hence, it is incomplete.

**Slices with no shared transitions:** When the minimal CMPN slices are computed, they are likely to contain shared transitions which serve as synchronization points among concurrently executing CMPN slices. Otherwise, a slice is assumed to operate on its own without ever

having to interact with the rest of the system. The presence of a system component that never interacts with the rest of the system is likely, although not conclusively, to be incorrect.

## 6 Conclusion and Future Work

Scenario-based or use case approaches are popular for several reasons: their intuitive appeal to practitioners, their scalability, understandability, and traceability. However, several weaknesses must be addressed before use case approaches can be effectively used in applications demanding high levels of assurance. We have identified the two most significant limitations:

1) lack of formal syntax and semantics, and
2) lack of analytic procedures to detect flaws resulting from use case interactions.

As an effective way to formalize the use case approach, we have proposed *the Constraints-based Modular Petri nets approach* (CMPNs) and presented informal, as well as formal, definitions and operational semantics. We believe that the research reported in this paper is significant because:

- We have demonstrated that existing Petri net formalisms, P/T nets or CP nets, are inadequate in formalizing use cases and that CMPNs can overcome such weaknesses, and
- We have developed a set of guidelines to determine if CMPNs are consistent and complete so that flaws in a use cases specification can be detected at the earliest possible opportunity.

We have demonstrated an application of our approach using real-world examples found in telecommunications software development. While our research offers improvements in formalizing the informal aspects of the use case approach, there are some issues that are worthy of further research. First, additional CMPN analysis methods are needed to detect flaws currently not covered. Data on the type and frequency of known errors in industrial applications of use cases would be helpful. Second, software tools to support CMPN-based modeling and analysis are needed because the productivity gains one can expect when applying our approach manually are limited. Finally, CMPN formalism itself could be extended. Promising areas for extension include support for timing analysis and system variables that are not of the Boolean type. Somé et al. [28] have demonstrated how such extensions can be introduced to timed automata and CMPNs need to be extended similarly.

## Acknowledgments

The authors are grateful to J.E. Hong for fruitful discussion on the initial version of this paper. Prof. Matt Jaffe of Embry-Riddle Aeronautical University and Prof. Tim Shimeall of the Naval Postgraduate School provided useful suggestions on how we could better present our ideas. We thank Prof. Matthias Jarke for his effort in coordinating the review process of our paper. We also thank three anonymous reviewers for their detailed, critical, yet constructive, suggestions. Our special note of appreciation goes to the one of the three reviewers who has thoroughly reviewed our paper three times and made significant contributions in improving the quality of our paper. Finally, we would like to acknowledge that this research was supported, in part, by the Center for Artificial Intelligence Research at KAIST.

## References

[1] M. Andersson and J. Bergstand, "Formalizing Use Cases with Message Sequence Charts," Master's thesis, Lund Inst. of Technology, 1995.
[2] H. Bachatene and M. Coriat, "PAM: A Petri Net-Based Abstract Machine for the Specification of Concurrent Systems," *Proc. Int'l Workshop Object-Oriented Programming and Models of Concurrency*, Turin, Italy, June 1995.
[3] M. Baldassari and G. Bruno, "PROTOB: An Object Oriented Methodology for Developing Discrete Event Dynamic Systems," *High-Level Petri Net: Theory and Application*, K. Jensen and G. Rozenberg, eds., Apr. 1991.
[4] F. Brooks, "No Silver Bullet: Essence and Accidents of Software Engineering," *Computer*, vol. 19, no. 4, Apr. 1987. **pages?**
[5] L. Brothers et al., "Feature Interaction Detection," *Proc. IEEE Int'l Conf. Comm.*, pp. 1,553-1,557, Geneva, Switzerland, May 1993.
[6] G. Bucci and E. Vicario, "Compositional Validation of Time-Critical Systems Using Communicating Time Petri Nets," *IEEE Trans. Software Eng.*, vol. 21, no. 12, Dec. 1995. **pages?**
[7] S. Christensen and N. Hansen, "Coloured Petri Nets Extended with Channels for Synchronous Communication," *Application and Theory of Petri Nets '94*, 1994.
[8] S. Christensen and L. Petrucci, "Modular State Space Analysis of Coloured Petri Nets," *Application and Theory of Petri Nets '95*, 1995.
[9] W. Damm, G. Döhmen, V. Gerstner, and B. Josko, "Modular Verification of Petri Nets: The Temporal Logic Approach," *Lecture Notes in Computer Science*, vol. 430. Springer-Verlag, 1989.
[10] B. Dano, H. Briand, and F. Barbier, "An Approach Based on the Concept of Use Case to Produce Dynamic Object-Oriented Specifications," *Proc. Symp. Requirements Eng. '97*, 1997.
[11] M. Glinz, "An Integrated Formal Model of Scenarios Based on statecharts," *Proc. European Software Eng. Conf. '95*, pp. 254-271, Spain, Sept. 1995.
[12] P. Hsia, J. Samuel, J. Gao, D. Kung, Y. Toyoshima, and C. Chen, "Formal Approach to Scenario Analysis," *IEEE Software*, pp. 33-41, Mar. 1994.
[13] P. Huber, K. Jensen, and R. Shapiro, "Hierarchies in Coloured Petri Nets," *Proc. 10th Int'l Conf. Applications and Theory of Petri Nets 1989*, pp. 192-209, Bonn, Germany, 1989.
[14] Y. Inoue, K. Takami, and T. Ohta, "Method for Supporting Detection and Elimination of Feature Interaction in a Telecommunication System," *Proc. Int'l Workshop Feature Interactions in Telecommunications Software Systems*, pp. 61-81, 1992.
[15] ITU-T, *Recommendation Z.120*, ITU-Telecommunication Standardization Sector, Geneva, Switzerland, May 1996 (review draft version).
[16] I. Jacobson, "Object Oriented Development in an Industrial Environment," *Proc. OOPSLA '87*, Oct. 1987.
[17] I. Jacobson, M. Christerson, P. Jonsson, and G. Overgaard, *Object-Oriented Software Engineering—A Use Case Driven Approach*. Addison-Wesley, 1992.
[18] M. Jaffe, "Completeness, Robustness, and Safety of Real-Time Requirements Specification," PhD thesis, Univ. of California, Irvine, 1988.
[19] K. Jensen, *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, vol. 1. Springer-Verlag, 1992.
[20] C. Lakos, "The Object Orientation of Object Petri Nets," *Proc. Int'l Workshop Object-Oriented Programming and Models of Concurrency*, Turin, Italy, June 1995.
[21] W.J. Lee, S.D. Cha, and Y.R. Kwon "Compositional Reachability Analysis of Petri Nets Models Using Petri Nets Slices," Technical Report TR-KAIST-SE-98, Korean Advanced Inst. of Science and Technology, 1998.
[22] S. Leue and P. Ladkin, "Implementing and Verifying Scenario-Based Specifications Using Promela/XSpin," *Proc. Second Workshop SPIN Verification Systems*, New Jersey, Aug. 1996.
[23] F. Lustman, "A Formal Approach to Scenario Integration," *Annals Software Eng.*, vol. 3, 1997.

[24] J. Mierop, S. Tax, and R. Janmaat, "Service Interaction in an Object Oriented Environment," *Proc. Int'l Workshop Feature Interactions in Telecommunications Software Systems*, pp. 133-152, St. Petersburg, Fla., Dec. 1992.
[25] R. Milner, *Communication and Concurrency*. Prentice Hall, 1989.
[26] C. Potts, K. Takahashi, and A. Antón, "Inquiry-Based Requirements Analysis," *IEEE Software*, pp. 21-32, Mar. 1994.
[27] W. Reisig, *Petri Nets: An Introduction*, pp. 62-63. Berlin, Heidelberg: Springer-Verlag, 1985.
[28] S. Somé, R. Dssouli, and J. Vaucher, "Toward an Automation of Requirement Engineering using Scenarios," *J. Computing and Information*, vol. 2, no. 1, pp. 1,110-1,132, 1996.
[29] J. Ullman, *Elements of ML Programming*. Prentice Hall, 1994.
[30] Univ. of Aarhus, *Design/CPN*, version 3.0, 1996.
[31] A. Valmari, "Compositional State Space Generation," *Advances in Petri Nets '93*, 1993.

**Woo Jin Lee** received the BS degree in computer science from KyungPook National University, Korea, in 1992 and the MS degree in computer science from Korea Advanced Institute of Science and Technology (KAIST), Korea, in 1994. He is currently a PhD candidate at KAIST. His research interests include requirements engineering, safety-critical systems, and process modeling. He is a student member of the IEEE.

**Sung Deok (Stephen) Cha** received the BS, MS, and PhD degrees in information and computer science from the University of California, Irvine, in 1983, 1986, and 1991, respectively. He taught at California State University Northridge and Long Beach from 1985 to 1990. From 1990 to 1994, he was a member of the technical staff at 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 Korea Advanced Institute of Science and Technology Computer Science Department. His research interests include software safety, formal methods, and computer security. He is a member of the IEEE and the IEEE Computer Society.

**Yong Rae Kwon** received the BS and MS degrees in physics from Seoul National University, Korea, in 1969 and 1971, respectively, and the PhD degree in physics from the University of Pittsburgh in 1978. He taught at the Korea Military Academy from 1971-1974. He was on the technical staff of Computer Science Corporation from 1978-1983, working on the ground support software systems for NASA's satellite projects. He joined the faculty of the Department of Computer Science of the Korea Advanced Institute of Science and Technology 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. He is a member of the IEEE and the IEEE Computer Society.