Towards Reusable Colored Petri Nets

Nam-Hee Lee, Jang-Eui Hong, Sung-Deok Cha, and Doo-Hwan Bae
Department of Computer Science
Korea Advanced Institute of Science and Technology, Korea
{nhlee,jehong,cha,bae}@salmosa.kaist.ac.kr

Abstract

Reuse has long been recognized as a key technology that can bring about significant productivity gains in software development. Code-level reuse is well-understood and frequently practiced. However, reuse in software requirements, another phase where much benefit can be expected, remains inadequately addressed.

Hierarchical Colored Petri Net (HCPN) formalism has been successfully used on several large-scale industrial projects, and it includes features designed to enhance modularity and understandability of the model. Unfortunately, such features fall short of making HCPN reusable.

In this paper, we provide formal definitions of RCPN as a reuse extension to the HCPN. RCPN can reduce complexity and cost of modeling and analysis in requirements engineering phase and increase software development productivity. We demonstrate an application of RCPN by demonstrating how a RCPN component for water level monitoring system (WLMS) can be reused in a coffee vending machine (CVM) model.

1. Introduction

Potential benefits of reuse are well understood, and reuse is considered an established practice in some phase of software development life-cycle. For example, several libraries, written in Ada, C, and C++, are commercially available to facilitate code-level reuse. Requirement reuse is another area where significant benefits can be expected, and formal foundations are essential to make systematic and rigorous specification reuse a reality.

Although numerous formal specification languages have been proposed in literature, only a handful of them are winning acceptance in industry. Hierarchical Colored Petri Nets (HCPN) formalism is considered one of industrial-strength formal method, and industrial projects that have successfully used HCPN include a command control system, an electronic fund transfer system, and a large transaction processing system in distributed environment [1, 2, 3].

By extending the classical Place/Transition Petri Nets (P/T Nets) with notions of token types (called colors) and arc expressions, HCPN makes development of behaviorally equivalent but concise models possible. Additionally, constructs such as substitution transitions, fusion places, and fusion transitions assist in the modular development of models [3, 4]. However, they are primarily notational conveniences introduced to enhance understandability of the model, and a modular model is not necessarily a reusable model.

In this paper, we extend HCPN formalism to define a reusable HCPN (RCPN) formalism. Several issues arise in defining RCPN components and incorporating them into “global” models. For example, conflicts may occur among the names used to define color sets, variables, or functions. Type definitions may need to be modified to best fit the context within which RCPN components are reused. Since reusable components are most likely to be designed as general as possible, RCPN formalism must provide mechanisms to resolve such potential conflicts.

The rest of this paper is organized as follows. In section 2, we briefly review formal definitions of the HCPN. In sections 3, we introduce the RCPN formalism and describe procedures to convert a HCPN component into a RCPN component. Section 4 illustrates how a reasonably complex behavioral model of a coffee vending machine (CVM) can be developed using a couple of RCPN components. In section 5, we evaluate the RCPN formalism with respect to a reuse taxonomy proposed by Krueger and conclude the paper.

2. Hierarchical Colored Petri Nets

Definition 1 HCPN, proposed by Jensen [6, 7], is a tuple $HCPN = (S, SN, SA, PN, PT, PA, FS, FT,$
1. \( S \) is a finite set of pages. Each page \( i \in S \) is a non-hierarchical CPN: \((E, P_i, T_i, A_i, N_i, C_i, G_i, E_i, I_i)\).

(a) \( \Sigma, P_i, T_i, \) and \( A_i \) is a set of color sets, places, transitions, and arcs, respectively.

(b) \( N_i, C_i, G_i, E_i, \) and \( I_i \) is a node, color, guard, arc expression, and initialization function, respectively.

2. \( SN \subseteq T \) is a set of substitution nodes.

3. \( SA \) is a page assignment function.

4. \( PN \) is a set of port nodes.

5. \( PT \) is a port type function. It is defined from \( PN \) into \( \{ in, out, i/o \} \).

6. \( PA \) is a port assignment function.

7. \( FS \) is a finite set of fusion sets.

8. \( FT \) is a fusion type function.

9. \( PP \in SN_{MS} \) is a multi-set of prime pages.

Figure 1 is a partial HCPN model for CVM. Once powered on, represented by firing of the transition Power On, the machine waits until coins are inserted and coffee buttons are pressed. These events are represented by having a token of types \( b \) (button) in the place Button Pressed and one or more tokens of the type \( n \) (nickel) or \( d \) (dime), respectively, in the place Coin Inserted. Should a user demand the coins be returned, the transition Coin-Return is fired and the coin deposit counters are cleared.

Although detailed description of all the HCPN features used in the CVM model cannot be provided due to space limitations, the followings are worth noting:

- HCPN modeling languages are based on a functional programming language ML, and HCPN models can be compiled and simulated. Reachability or invariants analysis can be applied on HCPN models. Furthermore, HCPN models can be mechanically transformed into a behaviorally equivalent non-hierarchical CPN and P/T net models.

- HCPN features such as transition guard conditions or conditional arc expressions\(^1\) often simplify the model significantly.

\(^1\)In order not to make the CVM model unnecessarily complex, we assumed that each cup of coffee costs only a nickel and that the CVM accepts only nickels and dimes. Inserting a dime and purchasing a cup of coffee result in decreasing the dime counter by one and increasing the nickel counter by one at the same time. Modeling such function in P/T Net would be quite complex.

**Figure 1. Coffee Vending Machine**

- Temperature and Ready, denoted by “FG”, are fusion places. They appear on other pages under the same name, and all of them are considered identical. In our example, fusion places are used both in the CVM and the heater controller models (see Figure 6). They are essentially notational conveniences (or syntactic sugars) designed to improve understandability and modularity of HCPN models.

- Transitions marked with “HS” (e.g., Coin-Return) indicate the substitution transitions. Places that are connected to the substitution transitions are given the port assignments (e.g., input, output, or input/output), and detailed behavioral model is drawn separately (see Figure 2). Substitution transitions are similar, in concept, to the procedure calls, and leaf level models are executed repeatedly when the corresponding substitution transitions are enabled.

While various HCPN constructs explained above are useful in developing modular and understandable behavioral models, they fall short of meeting reuse requirements such as abstraction or specialization.

### 3. Reusable Colored Petri Nets

Our extension to the HCPN to support reuse principles, RCPN, is defined as follows:

**Definition 2** A component of the reusable HCPN is defined as a tuple, \( RC_i = (HCPN_i, IG_i, OGT_i, OG_i, \sigma_i, G_i, N_i, E_i, A_i, F_i) \), where
Figure 2. A coin-return component

- $HCPN_i$ is a hierarchical CPN.
- $IG_i \in P_i$ is the in-gate place.
- $OG_i \in P_i$ is the out-gate place.
- $IGT_i \in T_i$ is the in-gate transition.
- $OGT_i \in T_i$ is the out-gate transition.
- $GA_i \subseteq A_i$ is a finite set of arcs.
- $IGN_i$ is an in-gate node function. 
  It is defined from $GA_i$ into $IG \times IGT \cup IGT \times FIG_i$, where $FIG_i \subseteq P_i$ is a set of formal input places like the formal parameters in programming languages.
- $OGN_i$ is an out-gate node function. 
  It is defined from $GA_i$ into $FOF \times OGT \cup OGT \times OGF_i$, where $FOF_i \subseteq P_i$ is a set of formal output places.
- $GE_i$ is a gate arc expression function.
- $NA_i$ is a set of name aliases.
  
  $$NA_i : \Lambda \rightarrow \Lambda$$

  , where $\Lambda$ is color set, variable, or ML function names.
- $MF_i$ is a set of mapping functions.
  
  $$MF_i : D_\Sigma \rightarrow D'_\Sigma$$

  , where $D_\Sigma$ and $D'_\Sigma$ are the domain of color sets.

It should be first noted that RCPN extends the HCPN with in-gate and out-gate places to define external interfaces. They are essentially wrappers placed around RCPN components to enforce single-entry and single-exit constraints. Even though RCPN may actually require multiple and distinct inputs or outputs, such aspects can be shielded by abstracting them in a token. RCPN, when necessary, can be augmented with arc expressions to combine or extract “component” tokens.

RCPN is designed to maintain maximum compatibility with the HCPN. A system is modeled as a HCPN at the top level, and its subpages can be either a set of HCPN pages or RCPN components. RCPN components can be separately tested for properties such as deadlock, boundedness, or fairness. Therefore, such behavioral analysis need not be repeated when the RCPN components are reused.

3.1. Name Aliases and Type Mapping Functions

It is a prudent practice to use descriptive names when developing models as is the case in programming. Unfortunately, name conflicts may occur, and effective reuse program cannot be established without providing systematic mechanisms to resolve such conflicts smoothly. Since the HCPN supports fusion constructs, it is unlikely that name conflicts will be resolved automatically. Moreover, it would be foolhardy to require the modeler to examine potential name conflicts and resolve them manually as (and if) they are detected.

Much cleaner solution to the name conflict problem is to allow name aliases be declared explicitly. In RCPN, name aliases can be included as an attribute of the in-gate place as shown below:

$$NA_i = \{
  \text{alias var } m : \text{n; }
  \text{alias color money : current; }
  \text{alias fun add_sum(a1, a2) : }
  \text{sum(a1, a2);} \\
\}$$

The above example specifies that all the occurrences of $m$, money, add_sum in the reusable CPN component are to be renamed as $n$, current, sum, respectively, in the global model.

Likewise, slight differences (e.g., range of valid values) in type definition may need to be adjusted. This step is often called specialization in reuse literature. In RCPN, specialization is provided by the mapping functions ($MF_i$). For example, assume that the token colors Value and Status are declared to be of types integer and enumerated types of either Active or Inactive, respectively. If this component is reused in
an environment where value is known to be between 1 and 100 and the status is either on or off, the following mapping functions are needed:

\[
NA_i = \{
\begin{align*}
\text{alias color Value : Limit =} \\
\quad \text{int with 1..100;}
\end{align*}
\begin{align*}
\text{alias color Status : OnOff =} \\
\quad \text{with On | Off;}
\end{align*}
\}
\]

Name aliases and mapping functions need not always be of one-to-one relationship. Since reusable components are likely to be designed as general as possible, only a subset of token types may be needed when the component is reused. For example, a RCPN model for coin management subsystem may have been designed to support all types of U.S. coins. If one were to reuse this RCPN component in an environment where only two types of Korean coins, 50 wons and 100 wons, are used the following specialization functions are needed:

\[
NA_i = \{
\begin{align*}
\text{alias color dollars : wons;}
\end{align*}
\}
\]

\[
MF_i = \{
\begin{align*}
\text{mfun(dollars : wons):}
\end{align*}
\begin{align*}
\text{case w of Nickel => Fifty | Dime => One_Hundred;}
\end{align*}
\}
\]

The name aliases and mapping functions do not change the behavior of the HCPN because they are merely mechanisms designed to resolve name conflicts.

One final note on RCPN is that processing of each reusable component is assumed to be atomic. This assumption is needed to guarantee the absence of side-effects between the global model and the reused RCPN component. That is, when a token is placed into the in-gate place \(IG_i\), “internal processing” of the \(RC_i\) is assumed to take place without interruption by other RCPN components until a token capturing the results is placed into the out-gate place \(OG_i\).

Since external interfaces of RCPN components are limited to the in-gate and out-gate places, the behavior of RCPN components can be abstracted as instantaneous firing of a transition in the global model. Therefore, other transitions of the global model can be executed concurrently while the reused RCPN is processed.

### 3.2. Converting HCPN to RCPN

A HCPN component can be converted into a RCPN component in steps described below:

1. **In-gate Definition:** All input places, the ones that have only outgoing arcs, are first identified. In case HCPN components contain port assignments (as is the case on pages expanding substitution transitions), all the places with input or input/output ports are included.

   In order to satisfy the single entry constraints, a new token color is defined as a product of all the input tokens. For example, in the example of coin management subsystem model, \(Coin\ Inserted\) and \(Return\ Requested\) are input places, and they accept tokens of the type \(MONEY\) and \(RETURN\), respectively (see Figure 3). Thus, a new token type \(MONEY \ast RETURN\) is declared and passed from the in-gate place to the in-gate transition.

2. **Token Extraction Arcs:** A number of arcs, equal to the number of input places identified above, are drawn from the in-gate transition to each of the input places. Arc expressions are designed to extract the required type of token component from the composite token.

3. **Out-gate Definition and Result Token Generation:** This step is analogous to the steps 1 and 2 explained earlier. Output places are first identified, and arcs are connected from all the output places to the out-gate transition. Finally, another arc is placed from the out-gate transition to the out-gate place, and role of arc expression is to generate a token which is a product of all the output tokens.

![Figure 3. A reusable coin-return component](image-url)
It should be noted that additional transitions, called connection-transitions, and arc expressions may need to be introduced when RCPN components are included in the global model. They are needed to keep the tokens stored in either the in-gate or out-gate places compatible to the tokens in other places of the global model (see Figure 4).

4. An Example: CVM with Embedded Heater Controller

In this section, we demonstrate an application of RCPN formalism. Figure 5 shows a RCPN model for the water-level monitoring system (WLMS), based on description provided in [9], with the following color sets and variable definitions:

```plaintext
color Power = with P_on | P_off;
color WaterLevel =
    integer with WL_LOW .. WL_HIGH;
var power : Power;
var waterlevel : WaterLevel;
```

We assume that a coffee vending machine (CVM) is equipped with a heater controller that continuously monitors water temperature and turns the heater on or off as needed provided that the tank contains “adequate” amount of water. Therefore, it is natural to include RCPN model for WLMS as an abstract and atomic component in the heater controller model, and the following name aliases and mapping functions are needed:

```plaintext
NA_WLMS = {
    alias color Power : POWER;
    alias var power : p;
    alias color WaterLevel : LEVEL;
    alias var waterlevel : level;
}
```

```plaintext
MF_WLMS = {
    mfun(Power : POWER) :
    case Power of P_on => ON |
                       P_off => OFF;
}
```

Finally, figures 4 and 6 illustrate how CVM and embedded heater controller interact using fusion places\(^2\) Temperature and Ready.

\(^2\)Although one might be tempted to model heater controller as another reusable component within the CVM model, we’ve decided against such a possibility since two subsystems are designed to operate concurrently.
5. Conclusions

In this paper, we described formal definitions of RCPN which is a reuse extension to the industrial-strength formalism HCPN. Effectiveness of RCPN is demonstrated using a model for coffee vending machine with embedded heater controller which includes a water-level monitoring system as a reused component. RCPN provides advantages such as conceptual simplicity and compatibility to HCPN.

However, RCPN needs to be further developed to provide better reuse support. This need becomes apparent if one were to evaluate RCPN according to a four-dimension reuse taxonomy proposed by Krueger[8]:

- **Abstraction**
  All approaches to software reuse need to provide abstraction capability for software artifacts. Without abstraction, software developers would have no option but to manually sift through a collection of reusable artifacts trying to figure out what each artifact do, when it could be reused, and how it can be reused.

- **Selection**
  Reuse approaches, to be effective, need to provide mechanism to help software developers locate, compare, and select reusable software artifacts.

- **Specialization**
  Software artifacts designed with reuse in mind tend to be general or generic. Therefore, reuse approaches must provide mechanisms to tailor reusable components as needed.

- **Integration**
  Reuse technologies typically have an integration framework. A software developer uses this framework to combine a collection of selected and specialized artifacts into a complete software system.
RCPN formalism proposed in this paper satisfies the abstraction, specialization and integration criteria well. Using in-gate and out-gate constructs, one can easily insulate potentially complex interfaces to reusable components. Treating reused RCPN component as an atomic and black-box entry is another feature designed to enhance abstraction capability. Name aliases, specified in the in-gate place, provides effective mechanism to accomplish conflict-free introduction of reusable components. Furthermore, mapping functions are introduced to support the specialization criteria to reuse.

Unfortunately, selection is an area for which RCPN failed to provide adequate support. It is necessary to further enhance RCPN to provide better support for the selection requirement.

References


