Uma Estratégia de Análise de Modelos em Redes de Petri e Orientação a Objetos

Érica de Lima Gallindo Ribeiro[1] and Angelo Perkusich[2]

[1] Laboratório de Redes de Petri,
erica@labpetri.dsc.ufpb.br,
http://www.labpetri.dsc.ufpb.br
[2] Departamento de Engenharia Elétrica,
Universidade Federal da Paraíba,
Caixa Postal 10105, CEP 58109-970,
Campina Grande - Paraíba,
perkusich@dee.ufpb.br,
http://www.dee.ufpb.br

Resumo RPOO - Redes de Petri e Orientação a Objetos – é uma notação para especificar sistemas distribuídos e concorrentes, baseada em Redes de Petri e Orientação a Objetos. Nesse trabalho, apresentamos uma primeira proposta para possibilitar a análise de modelos feitos utilizando esta notação. Estamos propondo a transformação de um modelo em RPOO para um modelo equivalente em Redes de Petri Coloridas – CP-Nets.

1 Introdução

A partir da especificação formal de um software é possível anteciparmos a presença ou ausência de características desejáveis ao software. Em [11], a autora enumera alguns objetivos que especificadores têm quando da construção de especificações formais de seus sistemas de software. Entre esses objetivos podemos citar: i)Mostrar que uma propriedade é válida para todo o sistema; ii)mostrar que um erro nunca ocorre; iii)mostrar que todos os casos foram cobertos, incluindo os casos de erros.

A escolha do método formal a ser utilizado vai depender das expectativas do modelador com a especificação, e das características do tipo de sistema a ser modelado. Existem inúmeros métodos formais adequados a diferentes classes de sistemas. Como exemplo de um método formal temos as Redes de Petri [7].

As redes de Petri têm sido extensivamente aplicadas à modelagem e análise de sistemas distribuídos e incorpora com naturalidade aspectos de concorrência e não-determinismo. Uma das principais vantagens das redes de Petri em relação a outras linguagens formais é o seu apelo gráfico. É relativamente fácil entender modelos em Redes de Petri mesmo para quem não está familiarizado com a notação[1]. Além disso, o comportamento de um modelo em Redes de Petri

[1] Uma explicação detalhada sobre a notação pode ser encontrada em [7].

pode ser analisado, seja através de simulação, ou através de métodos formais de análise.

Existem duas classes de redes de Petri, a saber: redes de Petri de baixo nível e redes de Petri de alto nível. O que diferencia as duas classes é o aumento no poder de descrição proporcionado pelas redes de alto nível. Fazendo uma analogia com as linguagens de programação, podemos dizer que as redes de baixo nível estão para as linguagens de máquinas (ex: *assembly*) assim com as de alto nível estão para as linguagens de programação de alto nível (ex: *C, pascal*). Como exemplo de redes de alto nível bastante conhecidas e utilizadas temos as redes de Petri Coloridas [5].

Entretanto, ainda existe muita relutância em se utilizar redes de Petri – ou qualquer outro método formal – por parte dos engenheiros de software. Um argumento muito utilizado é a dificuldade de se inserir essa atividade no ciclo de desenvolvimento usado em contextos práticos. Argumenta-se que os métodos exigem conhecimento especializado por parte da equipe e que as notações usadas são difíceis de casar com esquemas mais difundidos de documentação. Enfim, que o custo de usar métodos matemáticos excede os benefícios alcançados. Contudo, não se coloca em discussão o quão desejáveis são os benefícios advindos da sua utilização.

D. Parnas, em [8], defende que a deficiência e inadequação das notações são um dos maiores impedimentos para a disseminação de métodos matemáticos como parte do ferramental do dia-a-dia do engenheiro de software. É esta constatação que motiva uma busca por notações que aproximem os métodos formais da *praxis* convencional do engenheiro de software. Neste sentido, Guerrero em [4] desenvolveu uma notação, denominada Redes de Petri e Orientação a Objetos – RPOO, a ser utilizada na especificação de sistemas de software que possuam características de concorrência, distribuição e não-determinismo.

Em [6], apresentamos um experimento de modelagem utilizando a notação RPOO - Redes de Petri e Orientação a Objetos. O objetivo principal deste experimento foi o de iniciar um processo de validação da linguagem RPOO como uma linguagem adequada ao desenvolvimento de sistemas distribuídos e concorrentes. Esse trabalho nos permitiu confirmar nossas expectativas em relação a diversos aspectos de RPOO. Entretanto, além de outros experimentos ainda serem necessários para validar a notação, precisávamos também desenvolver mecanismos que possibilitassem a análise de sistemas modelados em RPOO.

O grande atrativo para o uso de métodos formais na fase de especificação de sistemas é a possibilidade de análise dos mesmos sem ser necessário ter o sistema implementado. Quando o sistema em questão é dito complexo, é preciso que mecanismos para análise automática de modelos sejam oferecidos. Não adianta se propor uma linguagem sem oferecer ferramentas que a suportem. É preciso oferecer aos projetistas de software algo a mais do que eles já possuem ao utilizar métodos informais de desenvolvimento.

Por ser uma linguagem nova, e ainda em desenvolvimento, não existe nenhuma ferramenta para a edição, nem para a análise de modelos RPOO. Assim, foi preciso optar por uma ferramenta que pudesse ser ajustada de forma a permitir

a edição e análise de modelos RPOO. A ferramenta escolhida foi o Design/CPN [3], uma ferramenta desenvolvida para a edição, simulação e análise de redes de Petri coloridas, permitindo, entre outras coisas, a geração do espaço de estados do modelo.

Uma das razões para a escolha do Design/CPN é que o comportamento de uma classe em RPOO é dado por uma rede de Petri colorida acrescida de inscrições de interação que promovem a a comunicação entre os objetos. Além disso, como o Design/CPN oferece um conjunto de funções para manipular com as redes, tornou-se possível desenvolvermos um método que possibilite analisar modelos em RPOO utilizando a ferramenta.

Nesse trabalho, apresentamos uma primeira proposta para a análise de modelos em RPOO. Estamos propondo a transformação de um modelo em RPOO para um modelo equivalente em CP-Net. O trabalho consiste em possibilitar o uso da ferramenta Design/CPN – ferramenta para edição, simulação e análise de redes de Petri coloridas – como auxiliar na geração do espaço de estados para o modelo RPOO. Assim poderemos utilizar as técnicas de análise do espaço de estados conhecidas para redes de Petri.

Este trabalho está organizado da seguinte forma: Na seção seguinte apresentamos um exemplo de modelos em RPOO. Em seguida, apresentamos a estratégia de análise utilizada. Na seção 4 apresentamos as diretrizes para a edição de modelos RPOO no Design/CPN, seguido dos aspectos a se considerar para o procedimento de conversão dos modelos. Finalmente, na seção 6, apresentamos nossas conclusões sobre este trabalho.

2 Redes de Petri e Orientação a Objetos

Nesta seção apresentamos um exemplo de modelagem de uma solução para o clássico problema dos jantar dos filósofos. Inicialmente, definimos o diagrama de classes, e em seguida detalhamos cada uma das classes separadamente. Os conceitos de RPOO necessários ao entendimento do modelo são introduzidos ao longo do detalhamento.

O problema dos jantar dos filósofos é um exemplo clássico da área de programação concorrente que tem por objetivo demonstrar como os recursos são alocados quando se tem mais de um processo concorrendo por eles.

Neste jantar cinco filósofos estão sentados em uma mesa redonda na qual cinco garfos são colocados. Os filósofos são entidades ativas que podem estar em dois estados: *comendo* ou *pensando*. Se um filósofo está no estado *pensando* e quer passar para o estado *comendo*, ele tenta pegar dois garfos - um garfo localizado em seu lado direito e outro localizado em seu lado esquerdo. Depois de comer, o filósofo recoloca os dois garfos de volta na mesa e volta para o estado *pensando*.

2.1 Diagrama de Classes

Na Figura 1 é apresentado o diagrama de classes para o modelo dos filósofos. Existem apenas duas classes: a classe `FILÓSOFO` descrevendo os filósofos e a classe `GARFO` descrevendo os garfos do sistema.

Figura 1. Jantar dos Filósofos – Diagrama de Classes

Assim como em outros métodos, o diagrama de classes apresenta uma visão abstrata dos objetos do sistema e de seus relacionamentos [1,9]. Nós do diagrama representam classes e arcos representam associações, composições ou generalizações. A notação usada segue em grande parte o estilo UML.

As inscrições nas extremidades dos arcos definem seletores[2] para a associação. Cada seletor pode ser usado no detalhamento da classe da extremidade oposta da associação para efeitos de navegação—envio de mensagens.

2.2 Detalhamento das Classes

Nesta seção apresentamos o detalhamento de cada uma das classes identificadas na seção anterior. O detalhamento de uma classe em RPOO consiste em dois elementos: uma rede de Petri colorida, denominada o *corpo* da classe; e inscrições de interação associadas a alguns elementos do corpo da classe, que promovem a interação entre objetos.

O corpo descreve o funcionamento dos objetos da referida classe. Ações são representadas por transições, e a representação dos estados é distribuída nos lugares da rede. As inscrições de interação RPOO são associadas exclusivamente às transições. Desta forma, cada inscrição determina uma relação entre a ação local e ações em outros objetos do sistema.

Existem três tipos de inscrições de interação. Inscrições de entrada, indicam um ponto de entrada de um dado externo no objeto, por exemplo na transição Liberar do modelo da Figura 3, a inscrição `fil?lib` indica que a mensagem `lib` é recebida do objeto `fil`.

Inscrições de saídas síncronas, indicam que dois objetos compartilham uma única ação de forma indivisível, por exemplo na transição Comer do modelo da

[2] Chamados *rolenames* em UML.

Figura 2, as inscrições `ge.lib` e `gd.lib` indicam que a mensagem `lib` é enviada de maneira síncrona aos objetos `ge` e `gd`.

Por último, inscrições de saídas assíncronas descrevem ações de saídas assíncronas de dados do objeto. Por exemplo, as inscrições `ge!req` e `gd!req` indicam que a mensagem `req` é enviada aos objetos `ge` e `gd`.

Na Figura 2 modelamos o comportamento dos objetos criados a partir da classe `FILÓSOFO`. Quando um objeto da classe `FILÓSOFO` é criado, o seu estado inicial é `1‘PENSANDO`. A partir deste estado, um objeto dessa classe pode enviar as mensagens `ge!req` e `gd!req`. Isto significa que a transição comer só pode ocorrer se for possível sincronizá-la com os objetos referenciados por `ge` e `gd`. Nesse caso, a sincronização expressa que um filósofo só pode comer se, e quando, pegar os dois garfos.

Em seguida o objeto vai para o estado comendo. A partir daí, quando a transição Pensar disparar, as mensagens `ge.lib` e `gd.lib` são enviadas assincronamente aos objetos referenciados por `ge` e `gd` indicando a liberação dos garfos pelo filósofo que, por sua vez, volta ao estado `PENSANDO`.

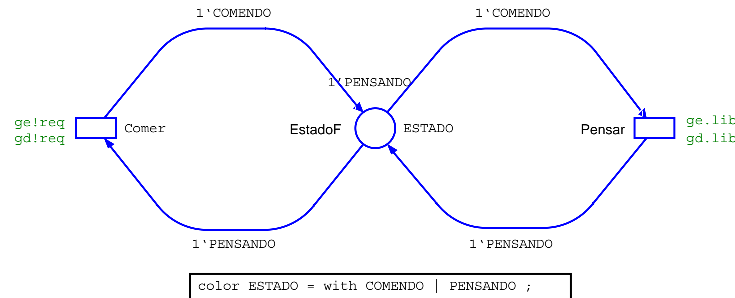


Figura 2. Jantar dos Filósofos – Classe `FILÓSOFO`

Na Figura 3 modelamos o comportamento da classe `GARFO`. Inicialmente, um objeto da classe `GARFO` está no estado `LIVRE`. Se a transição Pegar ocorrer, o objeto passa para o estado `OCUPADO`. A transição Pegar tem as seguintes inscrições de interação: `fil?req`. Essa transição só pode ocorrer mediante a recepção de um estímulo externo do objeto `fil`, que contenha a primitiva `req`.

Quando o estado do objeto for `OCUPADO`, a transição Liberar pode ocorrer se, e quando, receber um estímulo de um objeto externo indicando que este garfo foi liberado.

3 Estratégia de Análise

Um dos principais objetivos da construção de um modelo formal para um software é a possibilidade de se predicar sobre certos aspectos do software sem precisar chegar até sua implementação. É preciso que existam mecanismos que

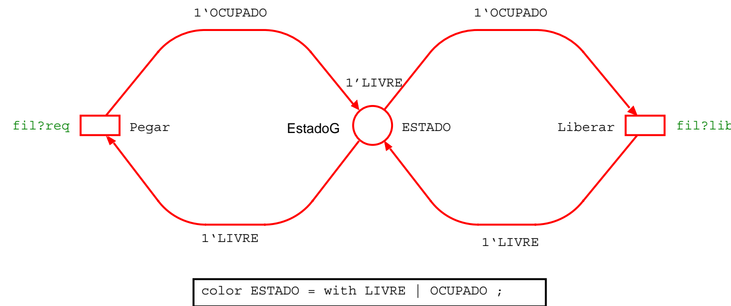


Figura 3. Jantar dos Filósofos – Classe `GARFO`

possibilitem analisar e verificar modelos em RPOO. Uma vez que analisar e verificar propriedades em especificações formais é uma tarefa praticamente impossível de ser realizada sem ajuda de poderosas ferramentas de software, tais mecanismos devem possibilitar uma análise automática dos modelos.

Na Figura 4 apresentamos a estratégia que estamos adotando para a análise dos modelos em RPOO. Primeiramente, é preciso que os modelos em RPOO – o diagrama e o detalhamento das classes – sejam desenhados no Design/CPN, utilizando os recursos disponíveis para edição de CP-Nets e criando recursos que possibilitem desenhar elementos não previstos na ferramenta, como as inscrições de interação.

Com os modelos desenhados, o passo seguinte é a obtenção do espaço de estados para o modelo – também chamado de *grafo de ocorrência* ou *árvore de alcançabilidade* na nomenclatura da ferramenta. Entretanto, como a ferramenta é para modelos em redes de Petri coloridas, antes de submeter o modelo à geração do grafo de ocorrência, precisamos transformar um modelo em RPOO para um modelo equivalente em redes de Petri colorida. Depois de gerada a rede de Petri equivalente, as funcionalidades providas pela ferramenta podem ser utilizadas para a geração do espaço de estados.

Com o espaço de estados gerados, é possível então que proceder com a verificação das propriedades desejadas para o sistema. Quando da construção de um modelo para um dado sistema, precisamos ter em mente quais os aspectos que temos interesse em analisar e, ainda, que perguntas desejamos responder. Em seguida, tais questões precisam ser mapeadas em elementos do modelo em RPOO – objetos, ligações, lugares, transições – resultando nas propriedades desejadas para o sistema.

O Design/CPN possibilita que propriedades sejam expressas utilizando-se CPN/ML [10] – linguagem de suporte à modelagem dos dados em redes de Petri coloridas. Uma alternativa à CPN/ML é utilizar uma biblioteca – denominada ASK/CTL [2] – que permite que propriedades sejam expressas em termos de lógica temporal.

A biblioteca ASK-CTL possui duas partes: uma que implementa a linguagem da lógica, e outra que implementa um verificador de modelos (*model checker*). O

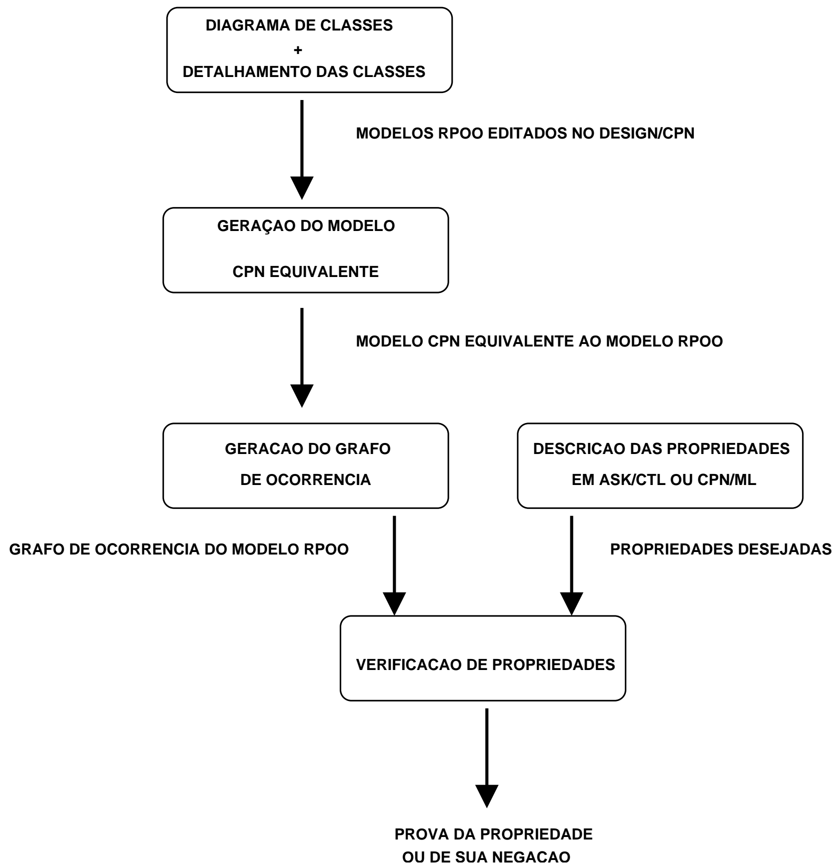


Figura 4. Estratégia de Análise de Modelos em RPOO

verificador de modelos recebe como argumento uma fórmula ASK-CTL, verifica esta fórmula no espaço de estados e retorna o valor verdade da fórmula dada.

O projetista então escolhe a forma de descrever as propriedades a serem verificadas em modelos RPOO. Cabe a ele decidir sobre qual das duas formas é mais apropriada ao seu propósito.

4 Edição de Modelos RPOO no Design/CPN

Para que o modelo em RPOO seja convertido num modelo CPN equivalente, se faz necessário que os modelos sejam feitos utilizando a ferramenta Design/CPN. Como a ferramenta é para redes de Petri coloridas, nem todos os elementos de modelagem em RPOO estão disponíveis. Sendo assim, é preciso adaptar a ferramenta de forma a contemplar a edição de tais elementos. Como exemplo de tais elementos podemos citar: os elementos de modelagem do diagrama de classes,

além das inscrições de interação associadas às transições do detalhamento das classes.

O Design/CPN faz distinção entre objetos *CPN* e objetos *auxiliares*. Os objetos CPN são aqueles utilizados na construção do modelo propriamente dito. Estes possuem um significado formal e influenciam no comportamento da rede. Como exemplo de objetos CPN podemos citar: lugares, transições, arcos, expressões de inicialização, guardas e expressões de arco. Os objetos auxiliares, por sua vez, não possuem nenhum significado formal. Estes desempenham um papel similar aos comentários em linguagens de programação, sendo utilizados apenas para ajudar a tornar o modelo mais legível, É possível criar objetos auxiliares de vários formatos diferentes como elipses e retângulos. Os objetos auxiliares podem estar relacionados a objetos CPN, não existe nenhum tipo de restrição quanto a isso. Por exemplo, é possível criar um conector auxiliar entre um par de lugares, sem influenciar no comportamento da rede. Utilizamos então, os objetos auxiliares do Design/CPN para contemplar a edição dos elementos RPOO.

5 Criação de um modelo CPN Equivalente

Nesta seção enumeramos e discutimos os aspectos a serem considerados em direção à obtenção de um procedimento automático de conversão de um modelo em Redes de Petri Orientadas a Objetos para um modelo em Redes de Petri Coloridas. Os aspectos aqui relacionados não seguem nenhuma ordem em particular. Os temas são apresentados na ordem em que foram identificados durante o experimento de análise.

5.1 União das Classes

Ao se construir modelos de sistemas complexos, são necessários mecanismos que nos permitam trabalhar com partes específicas dos modelos. Em termos de CPNs, a solução para decomposição e estruturação de modelos, resultou nas redes de Petri hierárquicas (HCPNs).

A ferramenta Design/CPN suporta a edição de redes de Petri coloridas (CPNs) e de redes de Petri coloridas hierárquicas (HCPNs). Um modelo em HCPN é comumente organizado em *páginas* que geralmente estão relacionadas através dos mecanismos disponibilizados. Entretanto, como o nosso objetivo é obter uma CPN equivalente ao modelo RPOO, e não uma HCPN, as páginas existentes no modelo não devem estar relacionadas. Cada página contém o detalhamento de uma classe presente no modelo bem como o nó de declarações local à classe, se esse existir.

Logo, um modelo RPOO no Design/CPN consiste de, no mínimo, uma página com o nó de declarações global contendo todas declarações de tipos comuns a todas as classes, e de uma página para cada classe do sistema modelado. Dessa forma, uma das primeiras atividades desenvolvidas em direção à conversão dos modelos RPOO, foi estudar as possibilidades de automatizar o processo de

junção destas páginas numa única página do Design/CPN. Assim, esta página resultante seria utilizada como entrada para a geração do espaço de estados. Acabamos por implementar esse processo de junção das páginas utilizando as bibliotecas de manipulação das CP-Nets disponibilizadas na ferramenta.

5.2 Conflito de Nomes

Em RPOO, declarações de tipos de dados são feitas em nós de declarações - de maneira semelhante às CP-Nets. Entretanto, em RPOO as declarações possuem escopo o que determina o subconjunto de classes em que um tipo é reconhecido. Ao utilizarmos escopo de tipos em RPOO, possibilitamos que tipos usados em mais de uma classe sejam declarados apenas uma vez e utilizados por várias classes. Os tipos utilizados numa única classe devem ser declarados no nó de declarações locais à classe.

A definição de escopo de tipos em RPOO permite que a cada nó de declaração seja associada uma inscrição de escopo, definindo o conjunto dos nomes de classes em que os tipos ali declarados devem ser válidos. Entretanto, para esse trabalho nós assumimos que só existem dois tipos de escopo: ou as declarações são locais à uma única classe, ou elas são compartilhadas por todas.

Tipos com definições diferentes em nós de declarações diferentes podem ter nomes iguais. Dessa forma, é bem provável que tenhamos nomes iguais para diferentes declarações de tipos quando as classes forem unidas. Para evitar este problema prefixamos todos os nomes de tipos de dados, de variáveis, e de constantes com o nome da classe. Tomemos como exemplo o modelo da solução para os problemas dos Filósofos, apresentados nas Figuras 2 e 3. Os nós de declarações das classes `FILÓSOFO` e `GARFO` possuem uma definição diferente com o mesmo nome, como mostrado na Figura 5.

```
color ESTADO = with COMENDO | PENSANDO ;
```

```
color ESTADO = with LIVRE | OCUPADO ;
```

Figura 5. Nós de declaração com nomes repetidos

Neste caso, o nome ESTADO está com duas definições diferentes - uma para a classe `FILOSOFO` e outra para a classe `GARFO`. Dessa forma, é necessário que antes da junção dos nós de declarações, os nomes sejam prefixados com o nome da classe ao qual ele pertence. Como resultado desse processo, temos o nó de declaração apresentado na Figura 6.

```
color Filosofo_ESTADO = with COMENDO | PENSANDO ;
```

```
color Garfo_ESTADO = with LIVRE | OCUPADO ;
```

Figura 6. Nó de declaração resultante

5.3 Simulação de Contexto

O detalhamento de uma classe em RPOO é uma descrição da estrutura de um objeto em particular. Entretanto, utilizamos classes com o objetivo de descrever objetos. Dessa forma precisamos de algum meio para simularmos instâncias de classes no sistema, ou seja, objetos. É preciso introduzir algum tipo de informação que nos permita saber a que objeto em particular uma dada marcação é referente. Por exemplo, a marcação da Figura 7 é uma marcação válida para o modelo da classe `FILÓSOFO`. Entretanto, como podemos observar, esta marcação só nos informa qual o estado no qual um filósofo se encontra num dado momento, não sabemos qual é a identificação do filósofo. Neste caso o filósofo está no estado **pensando** - representado pela marcação `1‘PENSANDO` no lugar EstadoF.

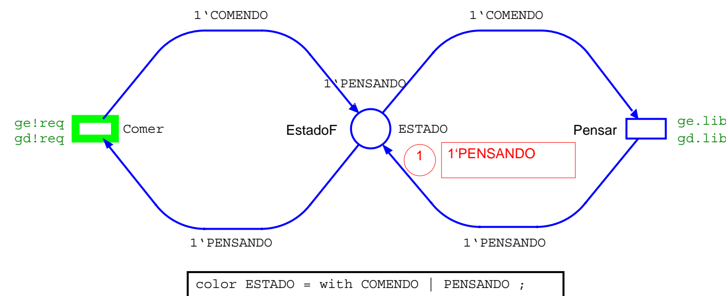


Figura 7. Marcação Inicial da Classe `FILÓSOFO`

É preciso então encontrar alguma maneira de representar a identificação dos objetos no sistema. Adotamos a seguinte estratégia: adicionamos um identificador de objetos às fichas dos lugares dos modelos. Isso significa que à cada cor do nó de declarações será acrescida de um identificador de objetos. Como exemplo tomemos a cor ESTADO do modelo da Figura 7. Sendo assim, o nó de declarações ficaria como mostrado na Figura 8.

Por enquanto, esse processo não foi automatizado. As modificações necessárias - alterações nos nós de declarações, alterações nos arcos, marcações iniciais e cores dos lugares - devem ser feitas manualmente. Para que isso seja feito de forma automática, um procedimento para tal deve contemplar :os seguintes pontos:

```
color ESTADO = with PENSANDO | COMENDO;
color INT = int;
color CLASSNAME = with Filosofo | Garfo;
color OBJ_ID = product  CLASSNAME * INT;
var id: OBJ_ID;
```

Figura 8. Nó de Declarações com Identificação dos Objetos

- Todas as cores dos lugares do modelo devem ser mudadas pelas novas cores criadas. Por exemplo, para a cor ESTADO será criada uma nova cor ESTADOxID.
- Cada cor criada obedecerá à seguinte forma: `<cor antiga> * OBJ_ID`. Por exemplo, a nova cor ESTADOxID com a definição `color ESTADOxID = product ESTADO * OBJ_ID` será criada.
- Todas as inscrições de arco devem ser alteradas para refletirem as alterações nas cores dos lugares de entrada e de saída dos arcos. Por exemplo, a inscrição do arco de saída da transição Comer deve ser substituída de `1‘PENSANDO` por `(1‘PENSANDO, id)`, onde `id` é uma variável que representa o identificador do objeto em questão.

5.4 Manipulação das Inscrições de Interação

Como mencionado anteriormente, o detalhamento de uma classe em RPOO é - excetuando as inscrições de interação - exatamente uma rede de Petri colorida. Como as inscrições de interação são os elementos ”estranhos”a uma rede de Petri colorida e o nosso objetivo é obter uma rede de Petri colorida, precisamos transformar as inscrições de interação em elementos de redes de Petri coloridas. Nesta seção discutimos os aspectos relevantes para essa transformação; por questões de espaço, discutiremos apenas o tipo mais complexo de inscrição de interação - inscrições de saída síncrona. O termo complexo diz respeito à grande quantidade de casos especiais a serem tratadas quando da conversão deste tipo de inscrição em elementos de CP-Nets.

No diagrama da Figura 9 mostramos graficamente os casos especiais das inscrições de interação síncronas que nos estamos referindo. No caso 1, uma transição de um objeto está associada a uma única inscrição de interação síncrona que modela o envio de uma mensagem a um outro objeto. No segundo caso, dois objetos diferentes enviam mensagem síncronas para um mesmo objeto e o recebimento destas é feito através de uma mesma transição. O terceiro caso representa a situação na qual um mesmo objeto envia mensagem síncronas para dois objetos diferentes através de uma mesma transição. O quarto, e último caso, representa a situação na qual um objeto envia uma mensagem síncrona para um outro objeto, que por sua vez, através da mesma transição que recebe a mensagem, envia uma outra mensagem síncrona para um outro objeto de uma terceira classe.

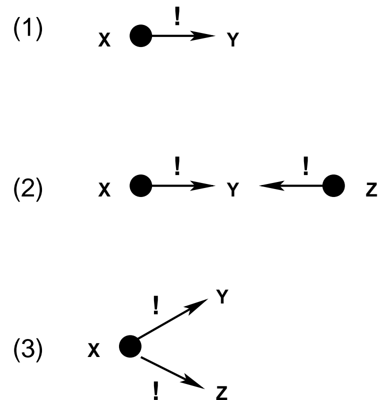
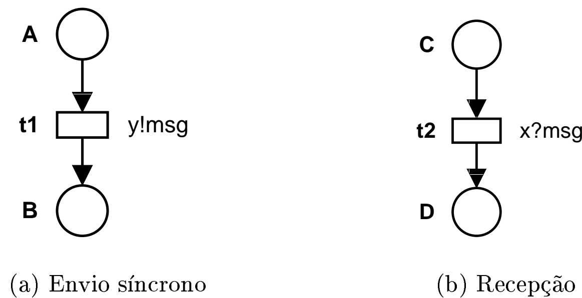


Figura 9. Casos especiais de uso de inscrições síncronas

A seguir apresentamos o caso geral (Caso 1) das inscrições de interação síncronas, seguido dos casos especiais que requerem tratamento diferenciado.

Caso 1: $(\mathtt{x} \rightarrow \mathtt{y})$ Para entendermos como é feita a conversão das inscrições de interação para elementos de redes de Petri coloridas, considere as classes do modelo da Figura 10.



(a) Envio síncrono (b) Recepção

Figura 10. Caso Geral: Envio de mensagem síncrona

O modelo destas classes capturam a forma mais simples de envio e recepção de um inscrição de interação síncrona. Um objeto da classe `X`, ao ser instanciado, será capaz de enviar a mensagem `msg` a objetos da classe `Y`. Como o envio é feito de forma síncrona - representado pelo caractere `!` - isto significa que a transição `t1`, da classe `X` só vai disparar se a transição `t2` também puder disparar. Dizemos então que a transição `t1` e a transição `t2` estão sincronizadas.

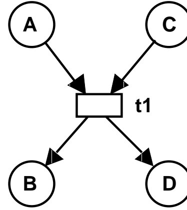
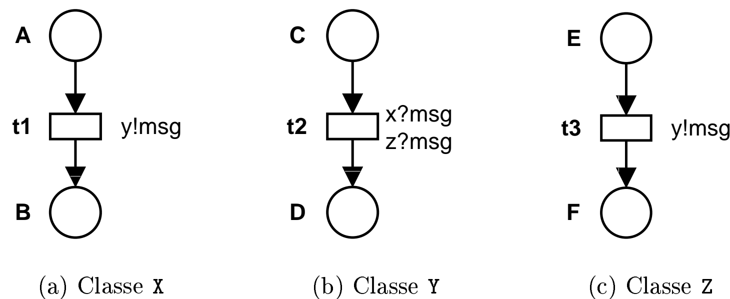


Figura 11. Modelo sem as Inscrições de Interação Síncronas

Este é o caso mais geral de uso da inscrição de interação síncrona, um objeto - dito emissor - envia uma mensagem de forma síncrona a um outro objeto - dito receptor. Dessa forma, tudo o que se precisa fazer é fundir as respectivas transições das inscrições.
Fundir `t1` com `t2` significa, transferir todos os lugares de entrada e de saída da transição `t2` – com seus respectivos arcos – da `t2` para a `t1` e em seguida remover `t2`. Fazendo isso no modelo da Figura 10, obtemos o modelo mostrado na Figura 11 sem inscrições de interação, que é resultado da fusão das transições `t1` e `t2`. Neste modelo, a transição `t2` foi eliminada e os seus lugares de entrada (C) e de saída (D) são transferidos para a transição `t1`. Como resultado, a transição `t1` passa a ter os lugares A e C como lugares de entrada e os lugares B e D como lugares de saída.

Caso 2: ($x \rightarrow y \leftarrow z$) Considere o modelo apresentado na Figura 12, contendo as classes `X`, `Y` e `Z`.

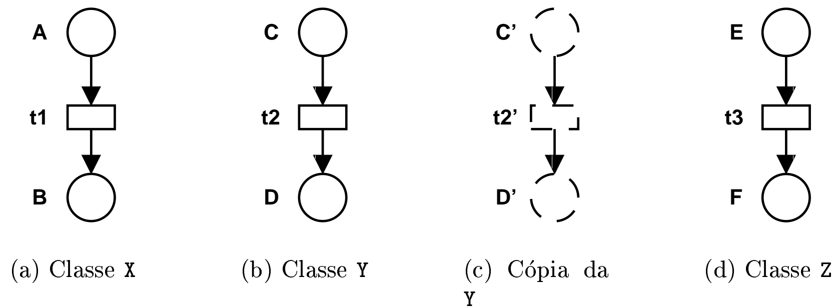


(a) Classe `X` (b) Classe `Y` (c) Classe `Z`

Figura 12. Modelos de Classes em RPOO

Sejam `x`, `y` e `z` objetos das classes `X`, `Y` e `Z` respectivamente. Neste modelo, objetos da classe `X` estão sincronizados com objetos da classe `Y` através da inscrição de interação `y!msg` que representa o envio síncrono da mensagem `msg` ao objeto `y`. A classe `Y`, por sua vez, modela numa mesma transição, o recebimento da mensagem `msg` do objeto `x` – inscrição `x?msg` – e o recebi-

mento da mensagem `msg` do objeto `z` – inscrição `z?msg`. Por fim, o objeto `z` está sincronizado com o objeto `x` através da inscrição `x!msg` que significa que o objeto `z` está sincronizado com o objeto `y`. Este corresponde ao caso 2 do diagrama da Figura 9, na qual um mesmo objeto (`y`) está sincronizado, num mesmo ponto, com outros dois objetos (`x` e `z`).



(a) Classe `X` (b) Classe `Y` (c) Cópia da `Y` (d) Classe `Z`

Figura 13. Modelo Intermediário

O tratamento desta situação é um pouco mais complicado do que o da situação anterior pois não requer apenas que as transições envolvidas na troca de mensagens sejam fundidas. Note que se apenas fundíssemos as três transições envolvidas `t1`, `t2` e `t3`, estaríamos introduzindo no modelo uma situação de conflito que não existia anteriormente. Dessa forma, é preciso criar uma cópia de partes do modelo, antes de efetuarmos a fusão das transições.

Então, o que deve ser feito quando esta situação é identificada no modelo pode ser dividida em duas etapas. Primeiro, a transição que contém as inscrições de interação de entrada – no caso, a transição `t2` da classe `Y` – e seus respectivos lugares de entrada e de saída, devem ser replicados. A execução deste passo resulta num modelo intermediário apresentado na Figura 13.

Em seguida, num segundo passo, o que se faz é fundir a transição `t1` com a transição `t2` e a transição `t2'` (transição replicada) com a transição `t3`, obtendo-se o modelo apresentado na Figura 14.

Note que desta forma, a situação de conflito é evitada, pois tanto pode ocorrer o envio da mensagem por `x` e o recebimento pelo `y` (modelado pela transição `t1-t2`), quanto pode ocorrer o envio da mensagem por `z` e o recebimento por `y` (modelado pela transição `t2'-t3`). Ainda, se todas as transições fossem fundidas, isso significaria que obrigatoriamente uma mensagem só poderia ser enviada por `x` para `y` se pudesse ser enviada por `z` ao mesmo tempo, que não correspondia ao que estava modelado através das inscrições de interação.

Caso 3: (x → y e x → z) Este caso está ilustrado no modelo da Figura 15. Um mesmo objeto envia de maneira síncrona, através do disparo de uma mesma

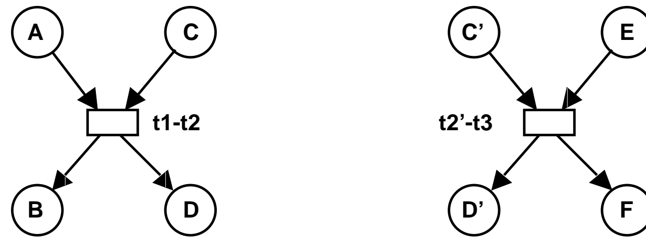
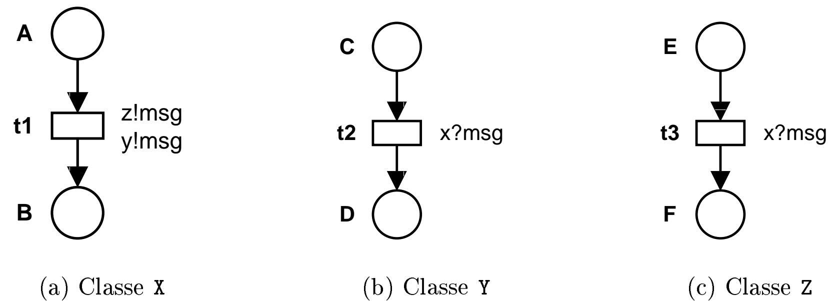


Figura 14. Situações envolvendo inscrições síncronas

transição, uma mensagem a dois objetos diferentes. O projetista deste modelo especificou que neste ponto, o objeto `x` só poderia estar sincronizado com o objeto `y` se também estivesse sincronizado com `z`. Observe que o objetivo do projetista foi bem claro para este modelo, ou seja, ele pretendia sincronizar todos os objetos envolvidos. Para este caso a solução é simples, basta fundir todas as transições envolvidas. O modelo resultante é apresentado na Figura 16.



(a) Classe `X` (b) Classe `Y` (c) Classe `Z`

Figura 15. Modelo RPOO com três classes `X`, `Y` e `Z`

6 Conclusões

Neste trabalho apresentamos uma estratégia para a análise de modelos em Redes de Petri e Orientação a Objetos. A idéia é converter um modelo em RPOO para um modelo equivalente em CP-Nets, para que com isso possamos utilizar a ferramenta para a edição, simulação e análise de CP-Nets – Design/CPN – a fim de analisarmos os modelos em RPOO. Uma vez que obtida a CP-Net equivalente, pudemos então utilizar a ferramenta para gerar o espaço de estados e assim proceder com a verificação de propriedades desejadas para o modelo.

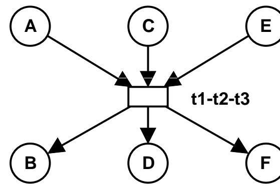


Figura 16. Modelo sem as Inscrições de Interação Síncronas

Optamos por esta estratégia por acreditar que os esforços desprendidos para "adaptar"o Design/CPN para a edição e análise de modelos em RPOO seriam muito menores do que os necessários para a construção de uma ferramenta do zero.

Elaboramos algumas diretrizes para a edição de modelos em RPOO no Design/CPN e com isso já foi possível desenhar os modelos na ferramenta. Em seguida, automatizamos uma série de procedimentos para a conversão em uma CP-Net equivalente. Com isso, não precisamos nos preocupar com os mecanismos de análise, usando os já disponíveis na ferramenta. Todos os procedimentos foram implementados em CPN/ML utilizando as bibliotecas de manipulação das CP-Nets disponíveis na ferramenta.

Referências

1. Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide*. Object Technology Series. Addison-Wesley, 1999.
2. Soren Christensen and Kjeld H. Mortensen. *Design/CPN ASK-CTL Manual*. University of Aarhus, 0.9 edition, 1996.
3. Meta Software Corpotarion. *Design/CPN Reference Manual for X-Windows*. Cambridge, MA, 2.0 edition, 1993.
4. D.D.S. Guerrero. *Redes de Petri e Orientação a Objetos*. PhD thesis, Universidade Federal da Paraíba, 2002.
5. Kurt Jensen. *Coloured Petri Nets: Basic Concepts, Analysis, Methods and Practical Use. Volume 1*. Springer-Verlag, 1992.
6. Ana Melo, editor. *Aplicação de uma Notação baseada em Redes de Petri e Orientação a Objetos: Um Experimento de Modelagem*, Rio de Janeiro, RJ, oct 2001. IME - Instituto Militar de Engenharia.
7. T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.
8. David Lorge Parnas. Mathematical methods: what we need and don't need. *IEE Computer*, pages 28–29, April 1996. In An Invitation to Formal Methods.
9. J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modelling and Design*. Prentice-Hall, Englewood Cliffs, New Jersey, 1991.
10. University of Aarhus, Aarhus, Denmark. *Design CPN - Overview of CPN ML Syntax, version 3.0*, 1996.
11. Jeannette M. Wing. Hints to specifiers. Technical Report Report CMU-CS-95-118R, School of Computer Science, Carnegie Mellon University, Pittsburgh, May 1995.