Validação de Modelos RPOO Usando Simulação

Edna Dias Canedo
edna.canedo@gmail.com
Centro de Tecnologia da Informação do Centro Universitário Unieuro, Brasília, DF, Brasil.

Resumo

A integração da teoria de redes de Petri e dos conceitos da orientação objetos surgiu como uma solução para a decomposição e estruturação de modelos em redes de Petri. RPOO foi definida integrando de forma ortogonal estes dois formalismos, permitindo que o sistema modelado tenha duas visões: uma visão de redes de Petri e uma visão de Orientação a Objetos. Este artigo relata o resultado da simulação do experimento de modelagem do serviço Bouncer, desenvolvido utilizando a notação RPOO. O modelo é analisado usando o sistema para simulação de RPOO.

Abstract

The integration of Petri nets theory and object oriented concepts has emerged as a solution to decompose and structure Petri net models. RPOO was defined by integrating these two formalisms on an orthogonal perspective, allowing that the modeled system has two visions: one Petri net vision and one OO vision. This paper reports the result of the simulation of the experiment of modeling service Bouncer, developed using notation RPOO. The model is analyzed using the system for RPOO simulation.

1. Introdução

Redes de Petri [15,17] é uma ferramenta para modelagem e especificação de sistemas que permite simulação e verificação de determinadas propriedades dos modelos, através de análise formal. Apesar da facilidade de utilização da ferramenta e da amigável representação gráfica, especial atenção tem sido dedicada na busca de mecanismos mais adequados para a estruturação interna dos modelos [2, 3, 6, 10, 14, 9]. Uma abordagem para a questão é a integração das redes de Petri com os conceitos de orientação a objetos [13].

Uma proposta de integração entre as redes de Petri e orientação a objetos é conhecida como redes de Petri orientadas a objetos (RPOO) [11]. RPOO tem uma proposta de integração dos dois paradigmas que nos permite ter duas visões ortogonais dos modelos: uma do ponto de vista apenas das redes de Petri e outra do ponto de vista da orientação a objetos (Figura 1). Desse modo, a formalização de RPOO preserva as características originais tanto das redes de Petri como da orientação a objetos.

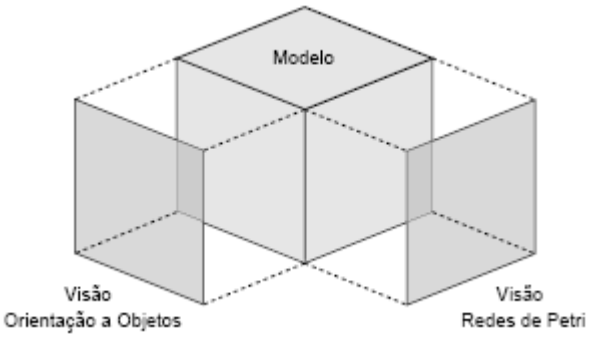

Figura 1 – Visões Ortogonais dos Aspectos de um Modelo RPOO

A idéia de RPOO é utilizar as construções da orientação a objetos para organizar os modelos. Logo, a construção chave da notação é a classe. Cada classe descreve um tipo de entidade do sistema. Neste caso, os objetos, que são as instâncias das classes, modelam entidades autônomas e concorrentes. O comportamento de cada classe de objetos é descrito através de uma rede de Petri colorida (CP-Net) [12]. Cada marcação pode ser vista como uma

representação do estado de uma rede de Petri da rede determina um possível estado das instâncias da classe. De forma análoga, as transições da rede modelam possíveis ações dos objetos. A comunicação entre os objetos do sistema completa a teoria de RPOO. Assim, a formalização propõe um conjunto de regras que definem como um sistema é representado a partir da comunicação entre as instâncias que estão executando. Este conjunto de regras e representações é chamado de sistema de objetos.

Neste artigo apresentamos o resultado da simulação de um experimento de modelagem, a fim de fornecer subsídios na avaliação do uso de RPOO para modelagem de sistemas reais. O sistema de simulação proposto integra ferramentas apropriadas para tópicos diferentes da formalização (ver Seção 5). Desse modo, o experimento também serviu para testar uma das ferramentas desenvolvidas para simulação. Para a leitura do restante deste documento, assume-se que o leitor tenha conhecimentos básicos de modelagem orientada a objetos e de redes de Petri coloridas.

O restante deste artigo está organizado como segue. Na próxima seção detalhamos alguns aspectos de RPOO para facilitar a compreensão do restante do documento. A seção 3 descreve a arquitetura do simulador de RPOO proposto. Na seção 4 descrevemos o problema que foi modelado em RPOO e submetido à simulação. A seção 5 descreve o processo de simulação e os resultados obtidos. Finalmente, na seção 6 apresentamos nossas conclusões.

2. Redes de Petri Orientadas a Objetos

Um modelo RPOO pode ser visto como um diagrama de classes, representando as entidades do sistema e suas associações; uma (ou mais) rede de Petri para modelar o comportamento de cada classe; e uma configuração inicial, que significa indicar instâncias das classes existentes no momento da inicialização do sistema modelado. Na Figura 2 apresentamos uma visão abstrata do modelo RPOO. Cada círculo dentro do Sistema de Objetos representa a instância de uma classe. As ligações entre os objetos indicam a possibilidade de envio/recepção de mensagens entre as instâncias. Como os objetos são entidades autônomas, o envio de uma mensagem de um objeto para outro não implica no consumo da mensagem. Assim, mensagens podem ficar pendentes e por isso são elementos considerados na formalização do sistema de objetos. O conjunto de objetos, ligações entre os objetos e mensagens pendentes é chamado de estrutura do sistema de objetos. As regras no sistema de objetos indicam qual o efeito do disparo da ação de um objeto sobre uma estrutura, ou seja, as regras definem como calcular a nova estrutura do sistema de objetos a partir do disparo de uma ação.

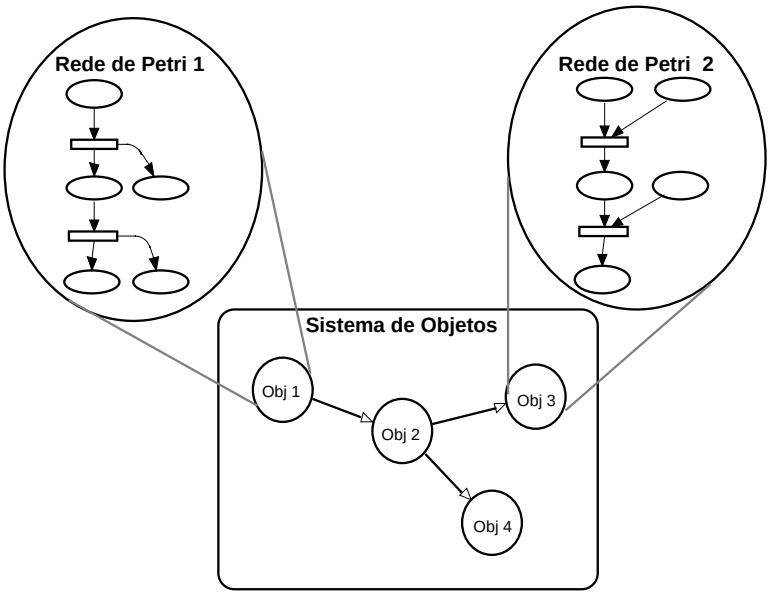

Figura 2: Visão Abstrata de um Modelo RP

Em um modelo RPOO, a troca de mensagens entre os objetos é definida através de inscrições de interação, que são associadas às transições das redes de Petri. As inscrições representam os tipos de ações que servem de

entrada para o sistema de objetos e que podem alterar a estrutura do sistema modelado. Pelas características da classe de sistemas para a qual RPOO foi projetada, ações podem ocorrer em conjunto, ou seja, mais de uma ação pode ser executada de forma atômica por objetos diferentes. Esse conjunto de ações é conhecido como Evento. As regras do sistema de objetos definem também qual o efeito do disparo de um evento sobre uma estrutura, a partir da composição do efeito das ações. Embora um evento possa conter uma quantidade indeterminada de ações, existem apenas sete tipos de ações definidos na notação RPOO, mais especificamente na formalização do sistema de objetos:

1. Ação interna - São ações ocorridas dentro de um objeto e que não modificam uma estrutura. Nos modelos das redes de Petri, as transições sem inscrição são interpretadas como uma ação interna.
2. Ação de criação - É uma ação na qual um objeto cria uma instância de outro. As regras definem que, quando um objeto cria outro, além do novo objeto passar a fazer parte da nova estrutura, uma ligação entre o objeto "criador" e o objeto criado também é inserida na estrutura.
3. Ação de saída assíncrona de dados - Uma ação em que um objeto envia uma mensagem para outro e continua seu processamento. As regras indicam que a mensagem só pode ser enviada se o objeto agente da ação possui uma ligação como objeto destino. O efeito dessa ação é a criação de uma mensagem pendente na estrutura.
4. Ação de saída síncrona de dados - Uma ação em que um objeto envia uma mensagem para outro e só continua seu processamento quando o objeto destino consome a mensagem. As regras indicam que a mensagem só pode ser enviada se o objeto agente da ação possui uma ligação como objeto destino. O efeito dessa ação é a criação e consumo da mensagem na estrutura.
5. Ação de entrada de dados - Uma ação em que um objeto consome uma mensagem pendente para ele. A mensagem consumida é excluída da estrutura resultante.
6. Ação de desligamento - Uma ação em que um objeto explicitamente se desliga de outro objeto.
7. Ação final - Uma ação em que um objeto se destrói. As regras definem que todas as ligações em que o objeto origem é o agente da ação sejam excluídas da estrutura.

A descrição formal das regras de disparo das ações e dos eventos sobre uma estrutura no sistema de objetos pode ser encontrada em [11].

3. Simulador RPOO

O sistema para simulação de RPOO é formado por um conjunto de ferramentas integradas que oferecem suporte computacional à simulação de modelos RPOO [16]. A arquitetura do sistema é composta basicamente por três elementos: um simulador para o sistema de objetos; um simulador para redes de Petri; e finalmente, um elemento responsável por executar a comunicação entre os simuladores do sistema de objetos e das redes de Petri. Na Figura 3 representamos a arquitetura do simulador de RPOO.

Para o sistema de simulação proposto, o módulo Simulador de Sistema de Objetos (Figura 3) é formado por uma ferramenta desenvolvida para oferecer suporte à formalização do sistema de objetos, que está sendo chamada de Simulador de Sistema de Objetos, ou SSO. O SSO é uma ferramenta que permite simular as alterações ocorridas em um sistema de objetos através da execução de ações. A ferramenta SSO pode ser considerada como um mediador, que trata as requisições de execução de ações de várias instâncias de objetos, validando ou não as ações requeridas. O comportamento dos objetos que se comunicam com o SSO é representado por uma CP-Net.

Figura 3 - Arquitetura do Simulador de RPOO

O SSO permite a criação de uma estrutura inicial para o sistema de objetos e a criação de eventos. Além disso, a ferramenta possui uma representação interna para as estruturas, os eventos e as regras que definem os efeitos dos

eventos sobre uma estrutura. A partir destes elementos, o SSO permite simular o efeito de qualquer evento sobre determinada estrutura, ou seja, é possível verificar qual a estrutura resultante em relação a um evento aplicado sobre uma estrutura inicial.

Para utilizar o SSO, foi definida uma gramática e uma linguagem para entrada de dados no sistema, que utilizam a representação algébrica proposta em RPOO [11]. Na expressão que representa uma estrutura, os objetos são idenficados por rótulos simples; as ligações são representadas por objetos contidos entre [ ] e separados por vírgula; as mensagens, por sua vez, são identificadas por rótulos seguidos de ( ), contendo os objetos origem e destino da mensagem. Como exemplo, temos a seguinte expressão: A[B] + B + mens(A, B).

O SSO interpreta esta expressão como sendo uma estrutura contendo os objetos A e B; o objeto A tem uma ligação para o objeto B, ou seja, o objeto A "conhece" o objeto B; e existe uma mensagem pendente com rótulo mens enviada de A para B.

A representação das ações é formada pelo objeto agente, seguido de ": ", mais o tipo da ação. Na Tabela 1 apresentamos os tipos de ações definidas na formalização do sistema de objetos, onde x é um objeto da estrutura.

Nome	**Ação**
#	Ação local (ou interna)
+x	Criação ou instanciação de objetos
x?m	Entrada de dados
x.m	Saída assíncrona de dados
x!m	Saída síncrona de dados
-x	Desligamento ou remoção de ligação
~	Ação final (ou auto-destruição)

Tabela 1 – Ações Elementares

Assim, a ação A: B.NovaMens é interpretada, pelo SSO, como uma ação de envio assíncrono de uma mensagem de A para B contendo o rótulo NovaMens. Como Simulador de Redes de Petri (Figura 3), utilizamos a ferramenta Design/CPN [7] ou, mais especificamente, o módulo de simulação do Design/CPN. O Design/CPN é uma ferramenta bastante utilizada na modelagem, análise e verificação das redes de Petri coloridas.

A camada Interface de Comunicação da Figura 3, por sua vez, representa a comunicação entre os módulos Simulador de Sistema de objetos e Simulador de Redes de Petri. No sistema atual, a integração entre as ferramentas Design/CPN e SSO não foi automatizada ainda. Assim, desempenhamos o papel do elemento responsável por executar a comunicação entre os simuladores do sistema de objetos e das redes de Petri. Os recursos da ferramenta Design/CPN [7] foram utilizados para recuperar as informações sobre os passos da simulação e estes resultados serviram de entrada para o SSO. O módulo de simulação do Design/CPN gera relatórios indicando quais transições foram disparadas em uma simulação. A partir destes relatórios, verificamos cada transição disparada, identificamos as inscrições nas transições relacionadas ao sistema de objetos (transições sem inscrição foram consideradas como ações internas) e através da forma de representação algébrica reconhecida fizemos a entrada de dados no SSO.

O objetivo no futuro é automatizar esta tarefa com a utilização do COMMS/CPN [8], uma biblioteca que possui primitivas para comunicação entre o Design/CPN e algum processo externo (que no caso, será o SSO). Assim, o COMMS/CPN vai representar a maior parte da Interface de Comunicação. Entretanto, parte desse módulo será implementado no Design/CPN e no SSO. No Design/CPN será executada a tarefa de traduzir as inscrições associadas às transições dos modelos para a linguagem reconhecida pelo SSO. No SSO serão implementadas primitivas para responder ao Design/CPN se os eventos solicitados podem ou não ser disparados, de acordo com as regras do sistema de objetos.

4. O Experimento de Modelagem

O serviço Bouncer [5,4] é um sistema distribuído para controlar o uso de licenças de software em ambientes de redes locais, oferecendo um conjunto de primitivas a serem usadas pelas aplicações que irão se beneficiar de seus serviços.

Na Figura 4 apresentamos um possível cenário do serviço Bouncer. Neste cenário, existem 02 máquinas HOST1 e HOST2, conectadas através de um canal de comunicação. Em cada máquina em que existem processos clientes executando, existe um único servidor Bouncer. Este é o caso, por exemplo, da máquina HOST2 que possui dois processos clientes PC1 e PC2, mas apenas o servidor Bouncer SB2.

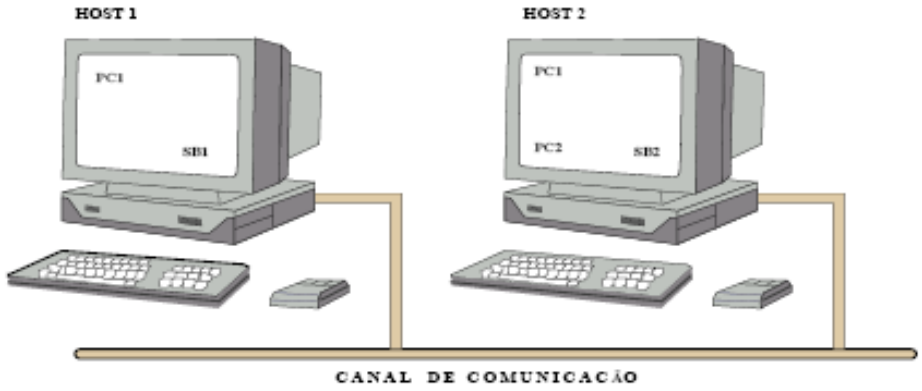

Figura 4 - Cenário do Serviço Bouncer

O Serviço Bouncer é constituído dos seguintes componentes: **Host** - em uma máquina da rede local, pode existir um ou mais processos clientes executando, mas apenas um servidor Bouncer deverá estar ativo. **Processo Cliente** - quando um processo cliente é executado, ele é responsável por iniciar o servidor Bouncer caso ele não esteja ativo, solicitar uma licença e após o uso desta liberá-la. **Servidor Bouncer** - ao ser iniciado pelo processo cliente a primeira tarefa do servidor Bouncer é ingressar no grupo bouncer, caso o grupo não exista o servidor irá criá-lo. Ao ingressar no grupo, o servidor Bouncer receberá do líder do grupo uma tabela de licenças. Quando o servidor Bouncer receber um pedido de licença de um processo cliente, ele verificará a disponibilidade de licenças para aquele processo em sua tabela. Se não existir licença disponível em sua tabela local, o pedido será negado. Caso exista, o pedido será repassado ao demais integrantes do grupo bouncer. **Serviço de Comunicação em Grupo** – o serviço de comunicação em grupo é utilizado para prover mecanismos de comunicação entre os servidores Bouncers ativos nas máquinas da rede local.

4.1 Modelos RPOO

Nesta seção apresentamos os resultados da modelagem do serviço Bouncer utilizando a linguagem RPOO.

4.1.1 Diagrama de Classes

Na Figura 5, é apresentado o diagrama de classes do modelo. Os nós do diagrama representam as classes e os arcos representam as associações. As inscrições nas extremidades dos arcos representam os seletores para a associação, os quais são ligados ao identificador do objeto. Os seletores host, clients, bouncer e service GC serão utilizados no detalhamento das classes para efetuar a troca de mensagens entre os objetos.

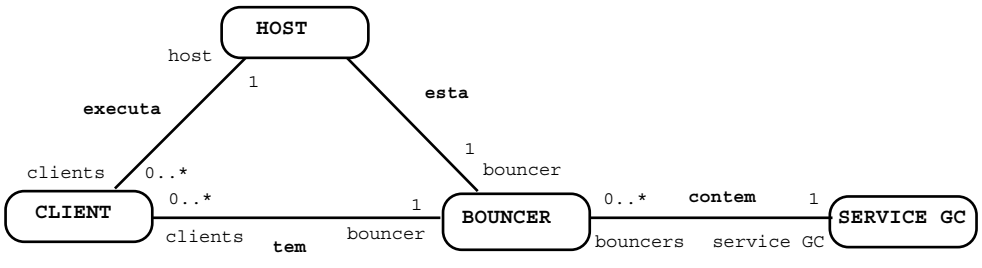

Figura 5 – Diagrama de Classes

4.1.2 Classe Host

Na Figura 6, é apresentado o corpo da classe HOST. O corpo da classe consiste em uma rede com dois lugares (process client e server bouncer) e cinco transições request bouncer, init_bouncer, request_bouncer, request end e request unjoin). A classe HOST atende as requisições dos objetos das classes CLIENT e BOUNCER. A

transição request bouncer tem as seguintes inscrições de interação client?req bouncer, que é uma inscrição de interação de entrada, e client!bouncer Inative, que é uma inscrição de interação de saída síncrona. A ocorrência desta transição indica que o host recebeu uma mensagem do client solicitando um servidor Bouncer para que ele possa executar. O host enviará como resposta uma mensagem para o client informando que o servidor Bouncer está inativo.

Ao receber a mensagem do host informando que o servidor bouncer está inativo, o client enviará ao host a mensagem Init bouncer. Quando o host receber esta mensagem a transição init_bouncer irá ocorrer. Esta transição tem as seguintes inscrições de interação client?Init bouncer, indicando que o host recebeu uma mensagem do client solicitando a inicialização de um servidor Bouncer, bouncer = new BOUNCER, que é uma inscrição de interação de instanciação, ou seja, criação do objeto bouncer e client.bouncer Active,bouncer, indicando que o host está enviando uma mensagem para o client informando que o servidor Bouncer foi iniciado e qual é o seu identificador.

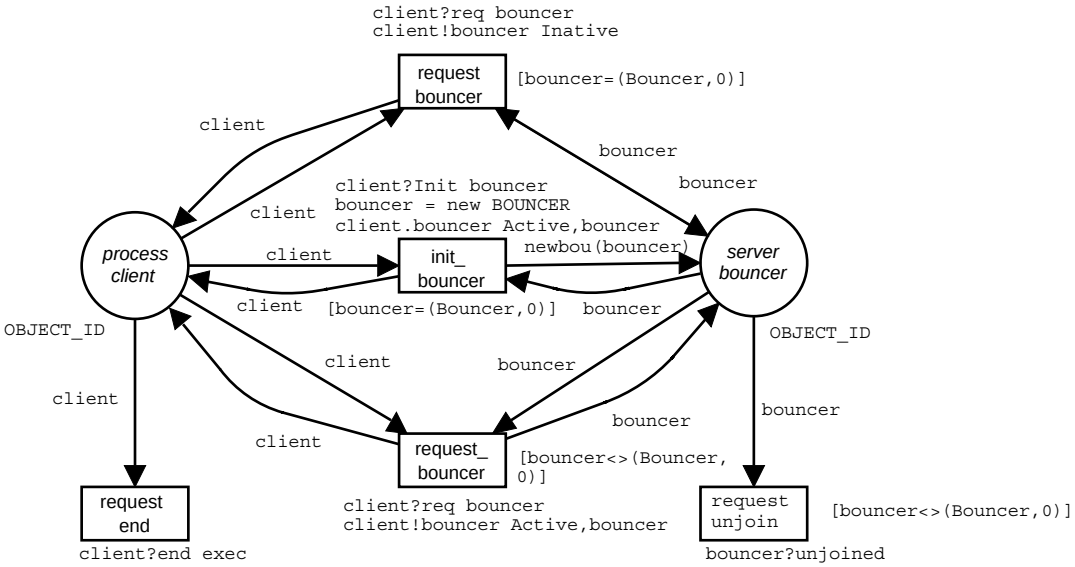

Figura 6 – Classe Host

A transição request unjoin tem a inscrição de interação de entrada bouncer?unjoined. A ocorrência desta transição indica que o host recebeu uma mensagem do bouncer, informando a finalização da sua execução. A transição request end modela a situação na qual o host recebe a mensagem client?end exec do client informando que ele finalizou a sua execução.

4.1.3 Classe CLIENT

Na Figura 7, é apresentado o corpo da classe CLIENT. O corpo da classe consiste em uma rede com seis lugares (process client exec, initializing sb, process client with sb, waiting requested lic, process client with lic e waiting release) e oito transições (request bouncer, request_bouncer, get bouncer, request license, get lic denied, get lic ok, request release e get released). A marcação da rede determina o estado inicial dos objetos instanciados desta classe. Assim, um client tem como estado inicial a marcação **client** no lugar **process client exec**, indicando que inicialmente o processo cliente está pronto para requisitar por um bouncer.

A partir do estado process client exec, podem ocorrer às transições request bouncer e request_bouncer. A ocorrência destas transições indica a situação na qual um client solicita ao host um servidor Bouncer para que ele possa executar. Caso exista um servidor Bouncer ativo no host, o seu identificador será informado ao client. Caso o servidor Bouncer esteja inativo, o client irá solicitar ao host a sua inicialização, como podemos observar nas inscrições de interação das referidas transições.

Quando o processo cliente conhecer o servidor Bouncer de sua máquina (estado em que existe uma ficha no lugar process client with sb), ele estará apto a requisitar uma licença ao bouncer para continuar a sua execução, enviando a mensagem bouncer.req license (transição request license).

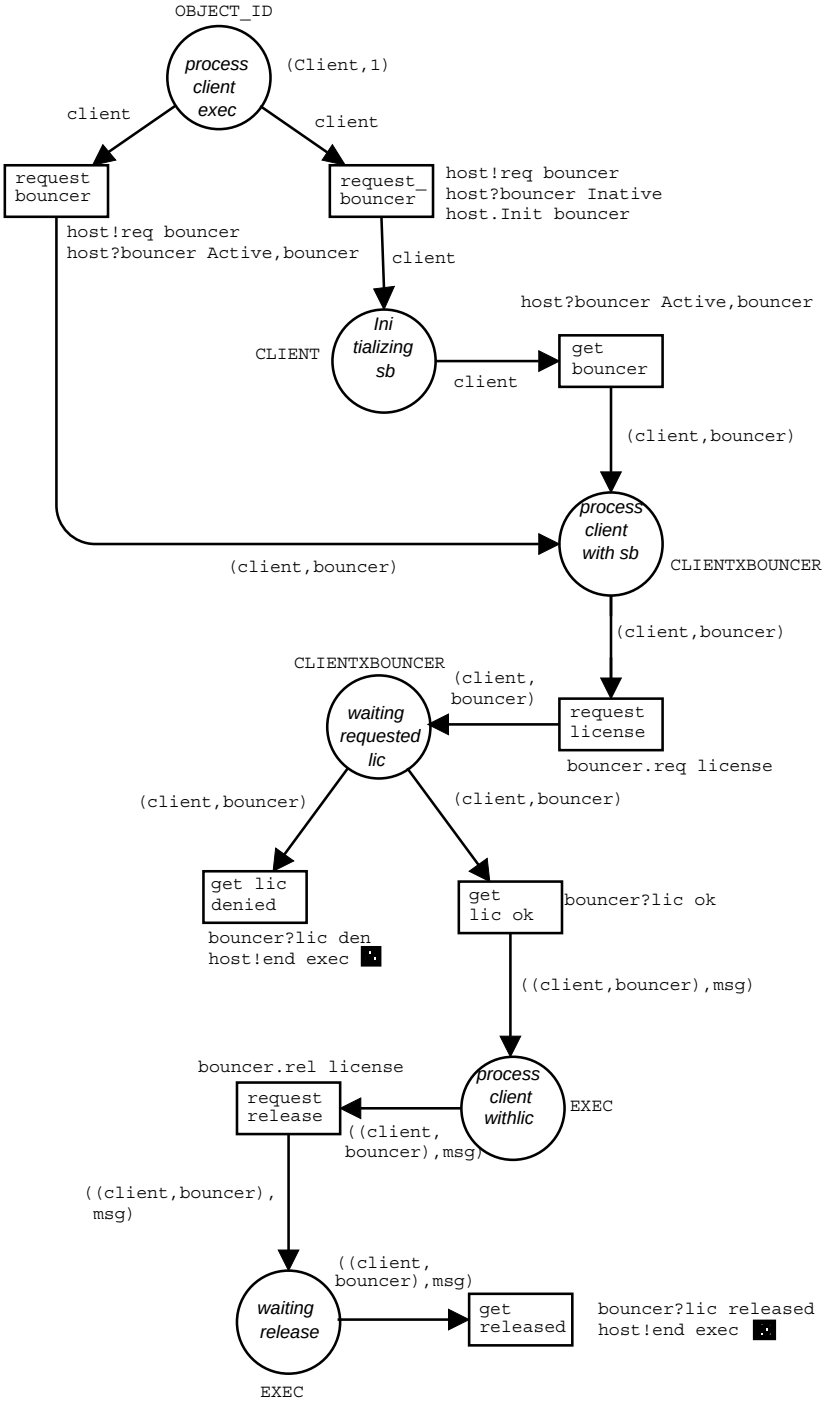

Figura 7 – Classe Cliente

A partir do estado waiting requested lic podem ocorrer duas transições. Ocorrendo a transição get lic ok, o client receberá a mensagem bouncer?lic ok, indicando que poderá continuar a sua execução e uma ficha será depositada no lugar process client with lic. Ocorrendo a transição get lic denied, o client receberá a mensagem bouncer?lic den do bouncer informando que a licença para a sua execução foi negada.

Quando o processo cliente terminar a sua execução, uma mensagem requisitando a liberação da licença que ele detém será enviada ao bouncer (transição request release). Ao receber a mensagem bouncer?lic released confirmando a liberação da licença, o client enviará uma mensagem de auto-destruição para o host, informando que está encerrando a sua execução (transição get released).

4.1.4 Classe BOUNCER

Na Figura 8, é apresentado o corpo da classe BOUNCER. O corpo da classe consiste em uma rede com dois lugares (Server bouncer e sb with table) e sete transições (request join, request license, release license, get_lic ok, get_lic release, get_lic denied e request unjoin).

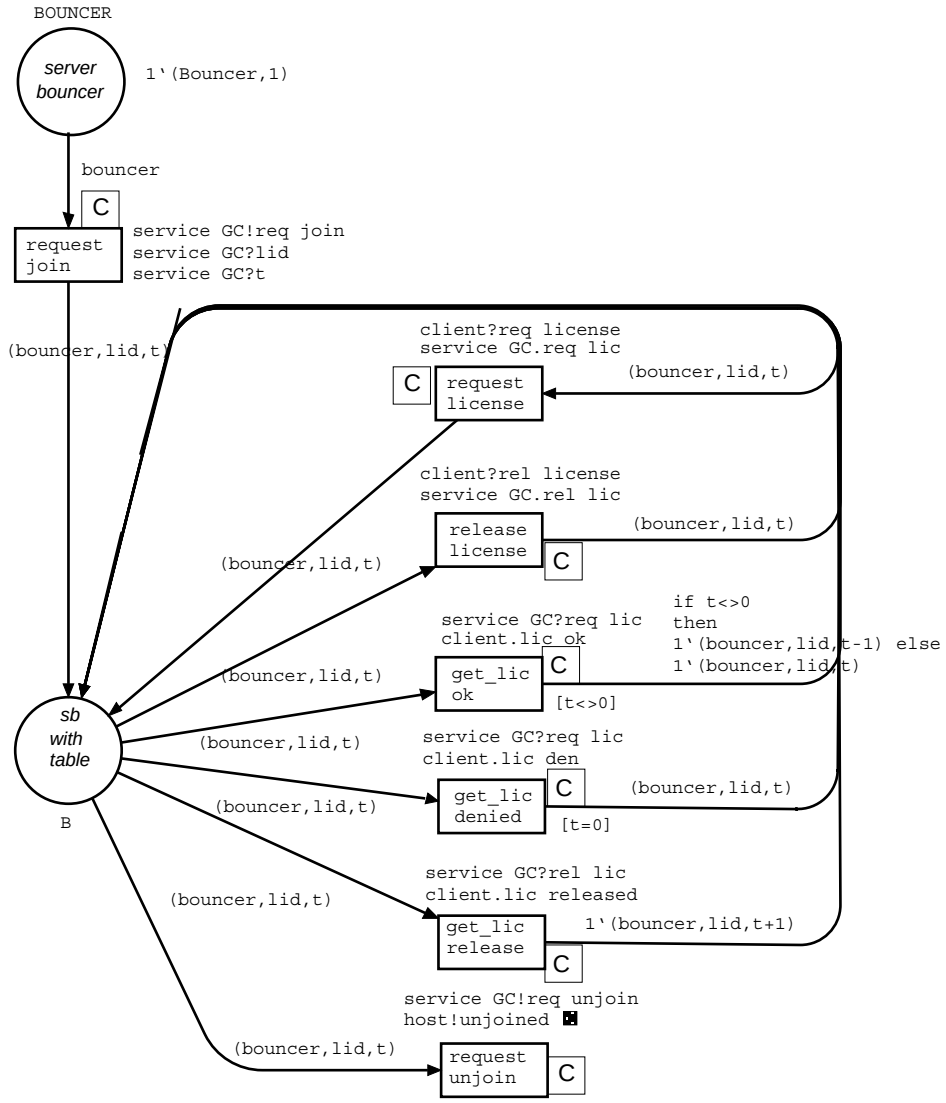

Figura 8 – Classe Bouncer

Quando o servidor Bouncer é iniciado pelo processo cliente, ele segue um processo de iniciação para atualizar a sua base de dados de acordo com a base de dados dos demais bouncers do grupo bouncer. Este processo é modelado pela transição request join. Ao disparar esta transição o bouncer envia a mensagem service GC!req join para o service GC (que modela o serviço de comunicação em grupo), requisitando a sua entrada no grupo bouncer. Ao enviar esta mensagem o bouncer receberá como resposta do service GC a mensagem service GC?leader,leader(lb), informando a identificação do líder do grupo bouncer e a mensagem service GC?table, newb (cli,bouncer,lb) atualizando a sua base de dados. Após, o bouncer estará apto a atender aos pedidos de requisição e liberação de licenças do client, estado em que existe uma ficha no lugar sb with table. Quando as transições request license e release license ocorrerem, as requisições de serviços recebidas através das mensagens client?req license e client?rel license serão repassadas para o grupo bouncer (service GC), através das mensagens service GC.req lic e service GC.rel lic.

Ao receber as mensagens service GC?req lic e service GC?rel lic do service GC, o bouncer irá responder as requisições do client de acordo com a sua base de dados. Caso exista licença disponível, a transição get_lic ok irá ocorrer e a mensagem client.lic ok será enviada ao client. Se não houver licença disponível, a transição get_lic

denied irá ocorrer e a mensagem client.lic den será enviada ao client, informando que o pedido de licença foi negado.

4.1.5 Classe SERVICE GC

Na Figura 9, é apresentado o corpo da classe SERVICE GC. O corpo da classe consiste em uma rede com um lugar lan e quatro transições (request_join, request license, req_release lic e request_unjoin). Quando o service GC receber do bouncer a mensagem bouncer?req join (transição request_join), o service GC enviará as mensagens bouncer!leader,leader(lb) e bouncer!table,newb(cli,bouncer,lb) para o bouncer informando o seu líder de grupo e a sua base de dados.

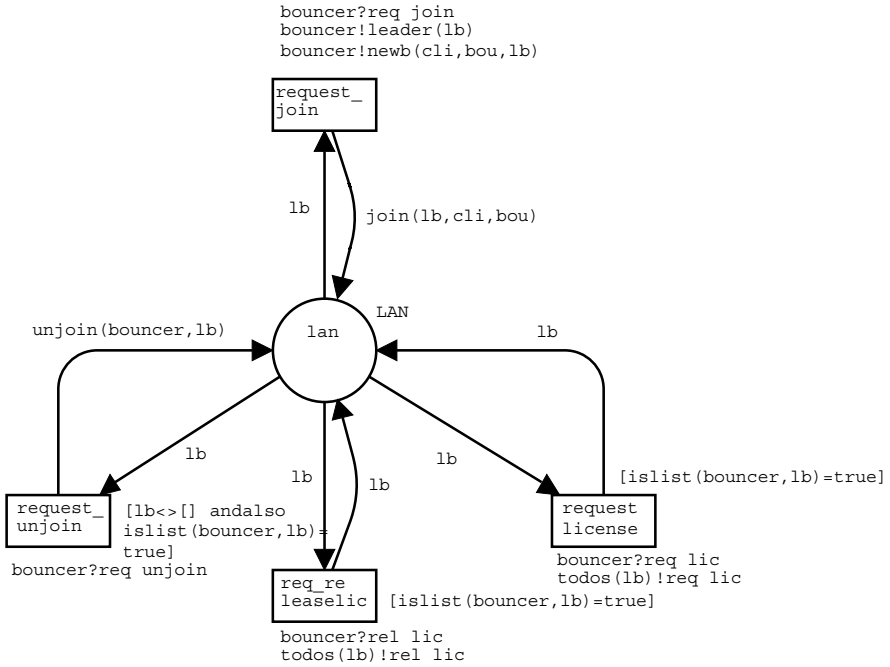

Figura 9 - Classe SERVICE GC – Serviço de Comunicação em Grupo

As transições request license e req_release lic modelam o recebimento das mensagens de requisição e liberação de licenças de um bouncer. Ao receber estas mensagens o service GC simplesmente as repassará para todos os membros do grupo bouncer, através das mensagens todos(lb)!req lic e todos(lb)!rel lic. A função todos(lb) ao ser avaliada retorna todos os bouncers pertencentes ao grupo bouncer. Quando o service GC receber a mensagem bouncer?req unjoin de um bouncer (transição request_unjoin), ele removerá o referido do grupo bouncer.

5. Simulação

Para efetuar a simulação do modelo consideramos diversos cenários. A Figura 10 mostra um dos cenários utilizados. Este cenário consiste em um host, um bouncer, um service GC e dois processos client. A simulação de cada classe do modelo RPOO foi efetuada isoladamente usando a ferramenta Design/CPN e a simulação do sistema de objetos utilizando a ferramenta SSO.

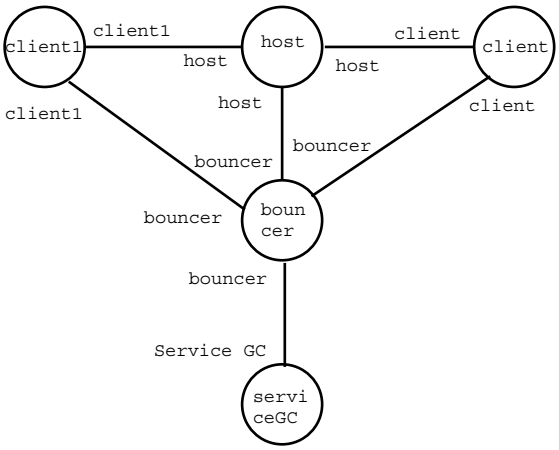

Figura 10 – Configuração Inicial do Serviço Bouncer

Os resultados da simulação do serviço Bouncer foram analisados a partir dos relatórios gerados pelo Design/CPN após a simulação de cada objeto do sistema e dos diagramas de seqüência de mensagens (MSCs) [1], gerados automaticamente pela ferramenta. Cada passo registrado durante a simulação das redes de Petri no Design/CPN foi convertido manualmente em uma mensagem para o SSO. A validação das ações pelo SSO foi utilizada para indicar qual o conjunto de ações, ou transições, poderia/deveria ser disparado pelas redes de Petri. Embora não seja uma verificação formal, a simulação aumentou o grau de confiabilidade na modelagem do problema e ofereceu uma importante contribuição para a avaliação da notação, porque permitiu a análise do comportamento dos modelos que utilizam a formalização de RPOO. A etapa de simulação contribuiu na identificação de alguns erros nas funções utilizadas no modelo, os quais foram corrigidos durante a simulação. Além disso, o experimento serviu também como uma etapa de teste da ferramenta SSO. A representação interna para as estruturas, os eventos e as regras da ferramenta foi testada. Testamos também a gramática definida para a utilização do SSO. Com o uso da ferramenta SSO, verificamos que ela atende aos propósitos para os quais foi projetada.

As Figuras 11 (a), 11 (b), 12 (a), 12 (b) e 13, representam graficamente o MSC gerado após a simulação do modelo. As colunas do MSC Client e Client2 representam os objetos client; a coluna Host representa o objeto host; a coluna Bouncer representa o objeto bouncer e a coluna ServiceGC representa o objeto serviceGC. As mensagens trocadas entre os objetos são representadas no MSC por um arco direcionado, definindo o objeto emissor e receptor. As setas são rotuladas com o tipo da mensagem enviada/recebida. Um processamento interno no objeto é representado no MSC com um processo marcado ■ (quadrado preto) na coluna correspondente ao objeto.

Para cada classe do modelo RPOO foi gerado um gráfico MSC com as ações ocorridas nos objetos instanciados. As ações estão rotuladas com as mensagens trocadas entre os objetos, que são as mesmas das inscrições de interação associadas às transições das redes de Petri que modelam os corpos das classes. Em cada gráfico representamos somente os objetos que se comunicam com a instância em questão.

Os rótulos com inscrição Evento indicam o evento que é executado no SSO. Assim, o Evento 1, das Figuras 11(a) e 11(b), representa o evento Client1:host!req bouncer @ host:Client1?req bouncer @ host:Client1!bouncer Inactive @ Client1:host?bouncer Inactive @ Client1:host.Init bouncer, executado pelo SSO. O símbolo @ é utilizado para representar a composição de ações em um evento.

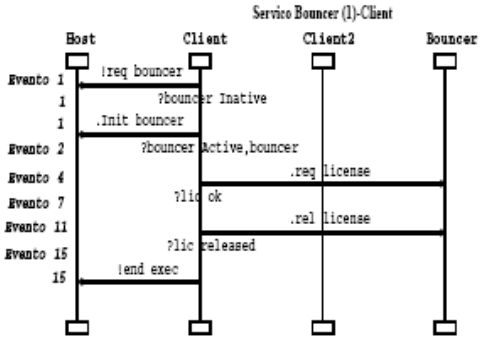

Figura 11 (a) Classe Cliente

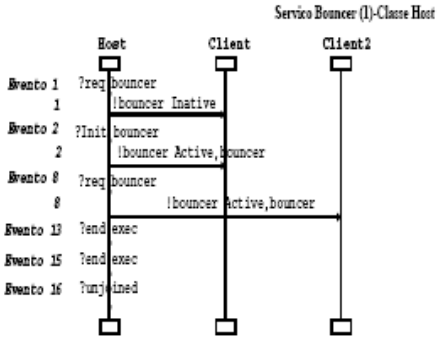

Figura 11 (b) Classe Host

Figura 11: MSC – Classe CLIENT e Classe HOST

A inscrição de interação!req bouncer (coluna Client da Figura 11 (a)), indica que o objeto client (origem) está enviando a mensagem req bouncer de maneira síncrona para o objeto host (destino). A inscrição?req bouncer (coluna Host da Figura 11 (b) indica que o objeto host recebeu (executou uma ação ■) a mensagem req bouncer enviada pelo objeto client. Como podemos observar na coluna Host do MSC, quando o objeto host executa o processamento desta mensagem, ele envia uma mensagem de volta ao objeto client informando se o Servidor Bouncer está ativo ou não. Neste caso, a resposta foi negativa, mensagem !bouncer Inative. Como podemos observar na coluna Client da figura 11 (a), quando o objeto client recebe a mensagem !bouncer Inative enviada pelo objeto host, ele solicita a inicialização do servidor Bouncer ao referido objeto, através da mensagem .Init bouncer. De acordo com a especificação do serviço Bouncer, o objeto client é responsável por iniciar o Bouncer, caso este esteja inativo.

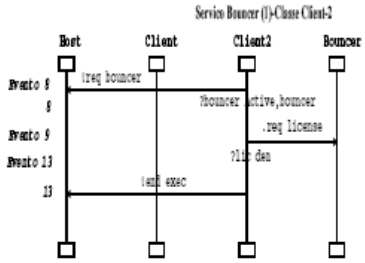

Figura 12 (a) Classe Cliente – Objeto 2

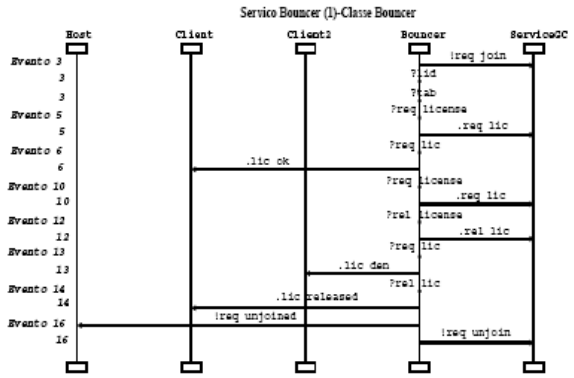

Figura 12 (b) Classe Host

Figura 12: MSC – Classe CLIENT e Classe BOUNCER

Quando o objeto bouncer recebe a mensagem?req license do objeto client (coluna Bouncer da Figura 12 (b), ele repassa a mensagem (.req lic) ao objeto ServiceGC. O objeto bouncer somente responderá a uma requisição de licença do objeto client após receber a mensagem .req lic do objeto ServiceGC. As mensagens !req unjoined e !req unjoin representam a destruição do servidor Bouncer, pois neste momento não existe nenhum objeto client executando. Quando o objeto ServiceGC receber a mensagem !req unjoin ele irá remover o objeto bouncer do grupo bouncer (coluna ServiceGC da Figura 13).

O processo de validação do modelo utilizando a ferramenta RPOO nos dá confiança de que o modelo está correto. As trocas de mensagens entre os objetos foram efetuadas da maneira como esperávamos (os resultados da simulação usando o SSO foram armazenados no arquivo simulação.log, onde verificamos que após cada ação sobre a estrutura vigente, a estrutura resultante era a esperada). Ou seja, as mensagens foram enviadas e recebidas corretamente pelos objetos do modelo. Além disso, os resultados das simulações efetuadas de cada classe isoladamente pelo Design/CPN, verificando o comportamento dos objetos em diferentes situações nos dão confiança na corretude dos modelos das classes, pois durante a simulação o comportamento dos objetos instanciados foram exatamente os definidos pela especificação do serviço Bouncer.

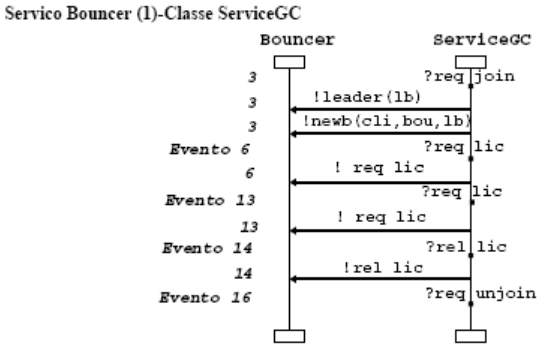

Figura 13: MSC Classe Service GC

6. Conclusão

Neste trabalho apresentamos os resultados de uma análise efetuada a partir da simulação do modelo do serviço Bouncer desenvolvido utilizando a notação RPOO. A simulação utilizou as ferramentas Design/CPN e SSO, e a análise foi baseada nos gráficos e relatórios gerados. Os resultados podem ser visualizados através dos diagramas MSCs, também gerados automaticamente. A integração entre as ferramentas não foi automática, ou seja, os resultados da simulação do Design/CPN foram utilizados como entrada no SSO manualmente.

A simulação possibilitou uma análise do comportamento do modelo e dessa forma nos permitiu avançar na avaliação da viabilidade da notação RPOO. Além disso, os resultados aumentaram nossa confiança na corretude do modelo. O experimento serviu também como etapa importante na fase de teste da ferramenta SSO.

A construção de ferramentas para suportar a modelagem, análise e validação de modelos RPOO são de extrema importância para a avaliação e utilização da notação. Como citado anteriormente, o SSO não está integrado a um simulador de redes de Petri. Neste sentido, estamos trabalhando na integração do SSO com a ferramenta Design/CPN. A idéia é utilizar a biblioteca COMMS/CPN [8] para fazer a comunicação entre os dois simuladores. Esta integração deve fornecer subsídios na avaliação de um projeto para construção de uma ferramenta completa para suportar a formalização de RPOO.

7. Referências

[1] Design/CPN message sequence charts library. Url: http://www.daimi.au.dk/designcpn/libs/mscharts.

[2] E. Battiston, A. Chizzoni, and F. De Cindio. CLOWN as a Testbed for Concurrent Object-Oriented Concepts. In F.DeCindio, G.A.Agha, and G.Rozenberg, editors, *Concurrent Object-Oriented Programming and Petri Nets*, volume 2001 of *Lecture Notes in Computer Science*. Springer, Berlin, 2001.

[3] E. Battiston and F. de Cindio. OBJSA Nets: A Class of High-level Nets having Objects as Domains Advances in Petri Nets. In G. Rozenberg, editor, *Lecture Notes in Computer Science 340*. SpringerVerlag, 1988.

[4] Tárcio Rodrigues Bezerra, Francisco Vilar Brasileiro, and Walfredo Costa Cirne Filho. Bouncer: Um serviço distribuído e tolerante a faltas para controle de licenças de software. In Marcos Borges, editor, *VII Simpósio de Computadores Tolerantes a Falhas*, pages 221–235, Campina Grande, PB – Brasil, July 1997. Sociedade Brasileira de Computação.

[5] Tárcio Rodrigues Bezerra. Bouncer- uma solução distribuída para controle de licenças de software. Dissertação de mestrado, Universidade Federal da Paraíba, 1996.

[6] O. Biberstein, D. Buchs, and N. Guelfi. Object-Oriented Nets with Algebraic Specifications: The CO-OPN/2 formalism. In F.DeCindio, G.A.Agha, and G.Rozenberg, editors, *Concurrent Object-Oriented Programming and Petri Nets*, volume 2001 of *Lecture Notes in Computer Science*. Springer, Berlin, 2001.

[7] S. Christensen, J. B. Joergensen, and L. M. Kristensen. Design/CPN — A computer tool for coloured Petri nets. *Lecture Notes in Computer Science*, 1217:209, 1997.

[8] Guy Gallasch and Lars Michael Kristensen. Comms/cpn: A communication infrastructure for external communication with design/cpn. In *Proceedings of CPN'01*, Aarhus – Denmark, August 2001.

[9] Dalton D. S. Guerrero. Orientação a objetos e modelos de redes de petri. Technical report, Coordenação de Pós-graduação em Engenharia Elétrica-COPELE/UFPB, Campina Grande, PB, April 1998.

[10] Dalton Dario Serey Guerrero. Sistemas de redes de petri modulares baseadas em objetos. Dissertação de mestrado, COPIN - Universidade Federal da Paraíba, 1997.

[11] Dalton Dario Serey Guerrero. Redes de petri orientadas a objetos. Tese de doutorado, COPELE-Universidade Federal da Paraíba, 2002.

[12] Kurt Jensen. *Coloured Petri Nets: Basic Concepts, Analysis, Methods and Practical Use, Volume 1*. EACTS – Monographs on Theoretical Computer Science. Springer-Verlag, 1992.

[13] Charles Lakos. From Coloured Petri nets to object Petri nets. In *Proceedings of the 15th International Conference on the Application and Theory of Petri Nets*, Lecture Notes in Computer Science, pages 278–297. Springer Verlag, Turin, Italy, 1995.

[14] J. Lilius. OB(PN) 2 : An Object Based Petri Net Programming Notation. In F.DeCindio, G.A.Agha, and G.Rozenberg, editors, *Concurrent Object-Oriented Programming and Petri Nets*, volume 2001 of *Lecture Notes in Computer Science*. Springer, Berlin, 2001.

[15] Tadao Murata. Petri nets: Properties, analysis and applications. *Proc. of the IEEE*, 77(4):541–580, April 1989.

[16] José Amâncio Macedo Santos. Ferramentas de suporte á simulação de RPOO. Proposta de dissertação de mestrado, COPIN – Universidade Federal da Paraíba, 2001.

[17] Canedo, Edna Dias. Validação de uma Linguagem de Modelagem de Sistemas Baseada em Redes de Petri e Orientação a Objetos. Dissertação de Mestrado - COPIN – Universidade Federal da Paraíba, 2002.

EDNA DIAS CANEDO

Possui graduação em Análise de Sistemas pela Universidade Salgado de Oliveira Goiás (1999) , mestrado em Ciência da Computação pela Universidade Federal da Paraíba (2002) e ensino-medio-segundo-grau pela Escola Técnica Federal de Goiás (1992) . Atualmente é professor titular do União Educacional de Brasília, professor titular da Faculdade Michelangelo, Professor Horista do Centro Universitário Unieuro e Consultora do Poliedro Informática Consultoria e Serviços Ltda. Tem experiência na área de Ciência da Computação , com ênfase em Sistemas de Software. Atuando principalmente nos seguintes temas: Redes de Petri, Orientação a Objetos.