Aspectos de Herança em uma Ferramenta de Modelagem de Sistemas Baseada em Redes de Petri

Sandro A. D. Costa[‡], Dalton D. S. Guerrero[†],
Jorge C.A. de Figueiredo[‡] and Angelo Perkusich[†*]
Laboratório de Redes de Petri
[‡]Departamento de Sistemas e Computação
[†]Departamento de Engenharia Elétrica
Universidade Federal da Paraíba
Caixa Postal 10105
58109-970 - Campina Grande - PB - Brazil
Fone: +55 83 310 1137 Fax: +55 83 310 1015
{sandro,abrantes}@dsc.ufpb.br and {dalton,perkusic}@dee.ufpb.br

Resumo

Métodos e técnicas matemáticas para o desenvolvimento rigoroso de sistemas são essenciais para o estabelecimento de uma disciplina de Engenharia de Software. A especificação formal de sistemas permite antecipar a detecção de erros e características indesejáveis, aumentando seu grau de confiabilidade. No caso de sistemas distribuídos, e por conseguinte concorrentes, o problema se agrava pela falta de arcabouços notacionais que capturem com simplicidade as propriedades inerentes desses sistemas. A teoria das redes de Petri é um dos mais bem sucedidos formalismos para lidar com concorrência. Entretanto, a falta de mecanismos para o desenvolvimento composicional impede sua utilização em sistemas complexos. Tais fatores são o principal motivo para a busca de unificação entre o mundo de redes de Petri e a Orientação a Objetos. Neste artigo discutimos especificamente questões pertinentes à integração de um dos mais importantes aspectos da Orientação a Objetos às redes de Petri: a herança.

Palavras-chave: Modelagem Orientada a Objetos, Redes de Petri, Herança, Métodos Formais, Sistemas Distribuídos.

Abstract

The mathematical techniques and methods for the rigorous development of systems are of upmost importance for the establishment of a software engineering discipline. The

*Os autores são parcialmente finaciados pels CAPES (Coordenação de Aperfeiçoamento de Pessoal de Ensino Superior) e pelo CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico).

formal specification of systems allows to anticipate the detection of errors and unexpected characteristics, thus increasing reliability. Considering distributed systems this problem is even worse due to the lack of notational frameworks that capture in a simple way the inherent properties of such systems. The Petri net theory is one of the well established formalisms to deal with concurrency. However, it suffers with the lack of compositional development mechanisms that is essential to cope with complex systems. In this paper we discuss aspects related to the integration of inheritance mechanisms and Petri nets. Two canonical forms of inheritance are conceptualized and integrated with Petri nets.
Keywords: Object-oriented Modeling, Petri Nets, Inheritance, Formal Methods, Distributed Systems.

1 Introdução

O desenvolvimento e o uso de métodos e técnicas matemáticas para o desenvolvimento rigoroso de sistemas é essencial para o real estabelecimento de uma disciplina de Engenharia de Software. Em especial, a especificação formal de sistemas permite antecipar a detecção de erros e características indesejáveis do sistema, aumentando o grau de confiabilidade e/ou eventualmente diminuindo custos de produção associados.

Em se tratando de sistemas distribuídos e orientados a eventos, o formalismo das redes de Petri tem se mostrado especialmente atrativo por incorporar com naturalidade aspectos de concorrência e não-determinismo, além de proporcionar uma extensa e fortemente consolidada teoria de provas que permite a verificação e análise dos modelos.

Recentes pesquisas têm aplicado com certo sucesso conceitos de Orientação a Objetos à teoria de redes de Petri [9], enriquecendo seu poder de expressão principalmente no tocante ao aspecto de modularidade e composição de sistemas complexos, notadamente uma das maiores limitações do formalismo. As abordagens propostas, entretanto, demonstram diversas limitações quanto à naturalidade com que capturam os conceitos da orientação a objetos.

Dentre os conceitos da Orientação a Objetos, a herança tem se mostrado o mais controverso e de difícil tratamento. As controvérsias entre os pesquisadores referem-se principalmente aos objetivos dos mecanismos de herança. Em essência os dois principais objetivos para considerar o mecanismo da herança em uma notação são:

1. permitir a representação da especialização conceitual;
2. permitir e facilitar o direto reaproveitamento de descrições (código, trechos de especificações, modelos, etc).

Em abordagens puramente teóricas, cujo objetivo é teorizar sobre limites impostos por um conjunto mínimo de construtos, é perfeitamente possível ignorar o segundo objetivo. Em ferramentas reais voltadas para o uso prático, entretanto, é comum a falta de comprometimento com os aspectos do primeiro objetivo. Percebemos, entretanto, que ambos têm relevada importância ao se considerar uma ferramenta cujo propósito é servir de notação para a modelagem/especificação formal de sistemas reais, muito embora, a literatura tenha mostrado diversas vezes que os objetivos citados são, de certa forma, antagônicos.

Assim, permitir a especialização conceitual, no seu mais pleno significado, é promover o desenvolvimento de especificações por refinamentos sucessivos, através dos quais a relação entre componentes especializado e genérico pode ser verificada rigorosamente. Por outro lado,

não queremos deixar de poder contar com a versão mais "fraca" de herança (do ponto de vista semântico), que permite o reaproveitamento de trechos de descrição entre objetos que não necessariamente tenham relacionamento conceitual.

Neste trabalho, portanto, abordamos algumas das mais relevantes questões relacionadas à inclusão de mecanismos de herança numa ferramenta de modelagem de sistemas distribuídos baseada em redes de Petri de alto nível. Isto implica em considerar os problemas comentados em um mundo não-seqüencial e não-determinístico em que a concorrência faz parte do próprio comportamento do sistema e seus componentes. Serão definidos dois mecanismos distintos de herança sobre classes cujos corpos são denotados por redes de Petri. Tais mecanismos serão tratados no restante do texto por *herança semântica* e *herança sintática*, e serão devidamente associados aos conceitos de subtipificação e subclassificação, respectivamente.

O restante deste artigo está estruturado como segue: na Seção 2 introduzimos em detalhes o conceito de herança; Na Seção 3 são formalizadas as redes de Petri de Alto Nível e sua relação com Orietação a Objetos; na Seção 4 discutimos como os conceitos de herança podem ser considerados no contexto de redes de Petri orientadas a objetos; por fim na Seção 5 apresentamos conclusões e perspectivas futuras.

2 Herança

Como um dos conceitos chaves em Orientação a Objetos, a herança foi introduzida inicialmente como um mecanismo de representação da *especialização conceitual* presente no domínio dos problemas, também conhecida como relacionamento "é-um". Porém, a forma como a herança foi mais largamente introduzida às ferramentas orientadas a objeto tornou possível um emprego fortuito do conceito: a reutilização de descrições (implementações, especificações, etc.) entre as classes relacionadas (uma subclasse herda a descrição da sua superclasse) [13].

A reutilização tem sido, na verdade, a grande promotora da Orientação a Objetos e tem impulsionado a sua divulgação, devido às vantagens práticas oferecidas. Dessa forma, o conceito de herança tem sido amplamente utilizado, difundido e suportado como uma forma eficiente de reaproveitamento. Contudo, assim torna-se possível que o relacionamento resultante entre as classes tenha pouco ou nada a ver com a especialização conceitual.

Podemos identificar como origem dessa desarmonia o fato do termo herança ser muitas vezes utilizado para representar diferentes tipos de relacionamentos. De maneira geral, a idéia de herança permite estruturar objetos (ou classes) em hierarquias, nas quais os descendentes compartilham ou reutilizam propriedades dos seus ancestrais [15]. Diferentes tipos de propriedades compartilhadas determinam diferentes hierarquias [15]. Logo, devem ser representadas por mecanismos distintos. Dois tipos de propriedades possíveis de serem herdadas estão em questão aqui:

1. propriedades comportamentais ou conceituais, cujo compartilhamento é mais adequado à noção de especialização conceitual;

2. propriedades estruturais, associadas à reutilização de descrições.

A idéia inicial, de ser possível representar o compartilhamento de propriedades estruturais e comportamentais através de um mesmo mecanismo de herança, talvez tenha sido motivada pela utilização, na documentação dos primeiros sistemas orientados a objetos, de analogias

com exemplos retirados da biologia, principalmente a classificação dos seres vivos em espécies [4, 15]. Nestas hierarquias, ambos os tipos de propriedades podem ser compartilhadas entre ancestrais e descendentes. Identificar por exemplo que um macaco "é-um" mamífero (conceitual) implica determinado nível de compatibilidade genética (estrutural). Porém, no universo de elementos computacionais que se pretende classificar, essas propriedades não possuem necessariamente qualquer relação. Somente sob condições específicas (que serão vistas mais à frente), compatibilidades estruturais podem garantir algum nível de compatibilidade de comportamento.

Uma vez esclarecida a distinção entre os dois tipos de hierarquias de objetos para compartilhamento de propriedades, vale ressaltar que os teóricos denominam como herança somente os mecanismos de relacionamento que preservam características de comportamento [5, 15]. Ao outro tipo de relacionamento, são atribuídas as mais variadas denominações: herança não estrita [15], subclassificação, etc.

Alguns conceitos aparecem sempre na literatura relacionados a herança, porém nem sempre com significados convergentes. Considerando importante a tarefa de deixar bem claro o significado de cada termo utilizado, vamos partir de definições mais genéricas até especializá-las e constituir uma terminologia precisa e satisfatória para nossos objetivos. É importante ainda notar que como os conceitos mais amplamente denominados como herança são subclassificação e subtipificação, fica claro que as definições mais básicas que devemos esclarecer são para os termos classe e tipo.

2.1 Classes e Subclassificação

O conceito de classe em Orientação a Objeto é bem mais consensual que o de herança. Podemos entender uma classe como sendo a descrição da estrutura comum a todas as suas instâncias (objetos daquela classe). A classe molda a estrutura que os objetos terão no momento da sua criação (instanciação). Logo, uma classe define uma coleção de objetos com a mesma estrutura interna [3]. Este mecanismo de instanciação/classificação reproduz uma importante abstração no entendimento de domínios de problemas. Muito mais do que possuir compatibilidade de qualquer natureza, objetos de uma mesma classe são idênticos (embora distingüíveis) nos momentos de suas criações, divergindo a partir daí durante as suas existências (caracterizado por seus estados).

Subclassificação pode ser entendida como sendo um mecanismo de reaproveitamento da descrição de uma classe na descrição de outras (subclasses) [3]. Desta forma, constitui-se em um instrumento poderoso para um desenvolvimento incremental de sistemas, através do qual somente as diferenças entre as classes precisam ser expostas, havendo reutilização do que já está definido. Este mecanismo é largamente denominado de herança por aqueles que têm como principal foco a reutilização. Porém, abrange aspectos puramente sintáticos.

Uma das questões mais relevantes com relação à subclassificação diz respeito às ações permitidas (cancelamento, redefinição, adição) sobre as propriedades na operação de geração de uma nova classe. Decidir quais destas operações estarão disponíveis definirá o caráter do mecanismo. Operações diferentes definem mecanismos diferentes. O que está em discussão aqui é o poder de expressão contra a clareza das estruturas hierárquicas resultantes. Implementações muito permissivas (que permitem cancelar, redefinir e adicionar propriedades livremente) com relação à subclassificação aumentam o poder de expressão da linguagem ao mesmo tempo em que reduzem a possibilidade de garantir algum nível de compatibilidade entre as classes associadas [15].

Pretendendo estabelecer os níveis de compatibilidade possíveis entre classes, Wegner propõe a seguinte classificação (aqui apresentada do nível mais básico ao mais restritivo) [16]:

cancelamento: uma (sub)classe não possui obrigatoriamente qualquer nível de compatibilidade com sua superclasse, uma vez que existe total liberdade no processo de herança de propriedades (cancelamento, redefinição e adição);

compatibilidade de nomes: uma (sub)classe preserva os nomes dos atributos e métodos presentes na sua superclasse, que podem contudo ser redefinidos;

compatibilidade de assinatura: uma (sub)classe preserva completa compatibilidade sintática com relação à sua superclasse, sendo considerados, além dos nomes, os tipos relativos à assinatura;

compatibilidade de comportamento: uma (sub)classe preserva o comportamento de sua superclasse (de acordo com uma definição de equivalência).

A compatibilidade de comportamento necessita de certas considerações semânticas sobre as classes e não será associada neste trabalho ao conceito de subclassificação e sim ao conceito de subtipificação, abordado na próxima seção. Contudo, em modelos seqüenciais de computação, a compatibilidade de assinatura (sintática) pode ser suficiente para garantir compatibilidade semântica, uma vez que o comportamento dos elementos pode ser entendido de maneira funcional. Desta forma, torna-se suficiente conhecer os nomes e os tipos dos elementos de entrada (parâmetros) e o tipo do elemento de saída para determinar o comportamento de uma entidade computacional. Esta é a sua assinatura.

Devido ao fato do termo subclassificação aparecer algumas vezes na literatura para designar outras idéias que não as apresentadas nesta seção, de agora em diante utilizaremos o termo herança sintática para denominar o mecanismo de herança utilizado com fins de descrição.

2.2 Tipos e Subtipificação

Com o objetivo de obter um mecanismo que garanta compatibilidade semântica entre classes relacionadas, um caminho que tem sido adotado é o suporte teórico dos tipos abstratos de dados [3, 10, 11, 15]. Neste sentido, o conceito de subtipificação (*subtyping*) é a tradução do mecanismo desejado, que será tratado neste trabalho como herança semântica.

Palsberg nos dá uma definição bastante genérica para tipo [11]. Para ele, um tipo determina simplesmente um predicado sobre um objeto. Deste modo, um objeto x pertence a um tipo T se satisfaz o predicado associado. Da mesma forma, um objeto de um tipo T_1, *subtipo* de um tipo T_2, é também do tipo T_2. O contrário porém não é garantido. Assim, um objeto pode pertencer a vários tipos, que necessariamente não possuem nenhuma relação entre si, desde que satisfaça aos respectivos predicados.

Embora caracterize uma visão bastante genérica, algumas categorias de predicados vêm sendo propostas/utilizadas na definição de tipos com objetivos específicos. Deste modo, surge o conceito de tipos de dados presente nas linguagens de programação, cujo principal objetivo é possibilitar a verificação de determinadas propriedades em tempo de compilação, evitando certos comportamentos não desejados. Para tanto, os predicados associados aos tipos devem estar relacionados à preservação de propriedades sintáticas e semânticas. Objetos de um mesmo tipo devem compartilhar o mesmo comportamento observável externamente [3]. Além disso,

um objeto de um subtipo pode ser utilizado (aparecer) em um contexto no qual é esperado um objeto do seu supertipo. Deve ser garantido, portanto, um nível de compatibilidade de comportamento entre um tipo e seus respectivos subtipos, que está relacionado com a noção de equivalência semântica adotada.

Podemos perceber a relação entre a noção de subtipificação discutida e a especialização conceitual que pretendemos capturar. O relacionamento "é-um" indica uma forte ligação entre as formas como elementos das categorias relacionadas podem ser manipulados (visão externa), e não necessariamente as suas semelhanças estruturais (visão interna), muito embora em outras formas de classificação, como a já citada classificação dos seres vivos em espécies, estas duas características venham juntas.

3 Redes de Petri Orientadas a Objetos

Uma rede de Petri pode ser representada graficamente por um grafo em que há duas classes de nós: lugares e transições. Os arcos da rede devem necessariamente conectar um lugar a uma transição ou o inverso (grafo bipartido). Estados de uma rede de Petri são denominados *marcações*, por serem representados por distribuições de marcas (ou fichas) nos lugares da rede. Graficamente, lugares são representados por círculos e transições por retângulos. é usual diferenciar entre a estrutura da rede de Petri e uma rede de Petri propriamente dita. Esta última é caracterizada por uma estrutura e uma marcação, denominada marcação inicial.

Definição 1 (Estrutura de Rede) *Uma estrutura de rede de Petri é uma tripla* $\langle P, T, F \rangle$, *em que:*

- P *é um conjunto finito de lugares;*
- T *é um conjunto finito de transições;*
- F *é uma relação de fluxo;*
- *desde que* $P \cap T = \emptyset$ *e* $F \subseteq P \times T \cup T \times P$

Além da estrutura definida acima, redes de Petri de alto nível têm alguns elementos sintáticos adicionais. Logo, para ser realmente preciso em relação à definição de redes de alto nível é necessário especificar a linguagem usada para inscrevê-la. Por limitações de espaço, entretanto, neste artigo faremos apenas algumas considerações sobre a linguagem.

Para simplificar e manter sob controle a natureza das expressões usadas na rede, nos utilizaremos de uma abordagem algébrica para descrições de dados. Assim, podemos utilizar a álgebra de termos resultante da especificação de tipos abstratos de dados (uma assinatura, um conjunto de axiomas e os termos definidos a partir dos operacionais e de variáveis). Quatro classes de inscrições podem ser identificadas e representadas:

Domínios dos Lugares determinam o tipo dos elementos comportados por um lugar e por conseqüência o tipo das expressões usadas nos arcos que se ligam àquele lugar. Serão denotados graficamente por *sorts* (nomes de conjuntos).

Pesos dos Arcos indicam os elementos que devem ser retirados ou adicionados aos lugares na ocorrência das transições. Serão representados por expressões que resultam em multiconjuntos do tipo associado ao lugar que se liga ao arco.

- *N é uma estrutura de rede de Petri;*
- *i é uma inscrição para a estrutura N baseada em $\mathcal{D}$;*

Definição 5 (Rede de Petri de Alto Nível) *Uma rede de Petri de alto nível é uma tripla $\langle \Sigma, \mathcal{N}, m_0 \rangle$, em que:*

- *Σ é um conjunto de tipos de dados;*
- *$\mathcal{N} = \langle N, i \rangle$ é uma estrutura de rede de Petri de alto nível baseada nos domínios de Σ;*
- *m_0 é uma marcação inicial para N, baseada nos domínios de Σ.*

Formalmente, é preciso ser mais rigoroso quanto às expressões e à linguagem subjacente utilizada para definir os tipos de dados[2]. A relação entre a rede e os tipos se dá através dos termos gerados a partir de uma assinatura e um conjunto de variáveis. Entretanto, como já foi comentado, devido às restrições de espaço tais detalhes foram omitidos neste trabalho, o que justifica a simplificação das definições.

Podemos definir o comportamento de uma rede de Petri em termos de uma regra de disparo para as transições. Assim dizemos que uma transição está habilitada e pode ocorrer se há fichas suficientes (de acordo com as expressões dos arcos) em todos os seus lugares de entrada. Se ocorre, a rede muda de marcação. A nova marcação é calculada retirando-se as fichas necessárias à habilitação e incluindo-se fichas nos lugares de saída, indicadas pelos respectivos arcos[3].

A regra dada acima, entretanto, indica que em um dado instante apenas uma transição pode ocorrer. Neste caso, diz-se que a semântica não é de concorrência real, mas de intercalação de eventos[4]. Para resolver isso, é necessário introduzir a idéia de passo. Informalmente, um passo pode ser definido como um conjunto de transições que disparam simultaneamente. Assim, para que um passo possa ocorrer é necessário que cada uma das transições possa ocorrer "usando" conjuntos disjuntos de fichas.

Por razões de limites de espaço, não definiremos formalmente a semântica das redes de Petri de alto nível aqui apresentadas.

Antes de apresentarmos as definições formais, é interessante caracterizarmos o modelo conceitual do mundo de objetos considerado neste trabalho. Cada objeto é visto como um componente de software autônomo que encapsula estado, comportamento e linhas de controle próprias. Assim definido, um objeto tem total independência em relação aos demais existentes, tornando a concorrência uma propriedade natural destes sistemas. O mecanismo básico de comunicação assumido é a passagem assíncrona de mensagens, que se justifica por concebermos o sistema como uma coleção de objetos conceitualmente distribuídos no espaço. Finalmente, é necessário dizer que os objetos e a sua topologia de interconexão é potencialmente dinâmica. Isto implica que os objetos podem ter ciclo de vida limitado e que podem conhecer seus "parceiros" de comunicação durante o funcionamento do sistema.

[2]Em nossa abordagem utilizamos a teoria das Σ-álgebras poli-sortidas para especificar os tipos de dados.

[3]Para determinar as fichas que devem ser retiradas e incluídas nos lugares da vizinhança da transição é necessário avaliar as expressões para uma dada substituição de variáveis. Além disso, é preciso verificar se tal substituição valida a guarda da transição.

[4]Expressão escolhida para traduzir *Interleaving*.

Guardas são condições sobre os elementos envolvidos que devem ser satisfeitas para que a transição ocorra. Podem ser representadas por equações ou expressões booleanas associadas às transições.

Marcações Iniciais determinam as fichas que devem estar presentes inicialmente em cada lugar. Serão representadas por expressões fechadas (sem variáveis) cujas avaliações resultam em multi-conjuntos de elementos do domínio do lugar relacionado.

A definição a seguir estabelece um conjunto de funções que permitem associar tais elementos sintáticos a uma estrutura de rede.

Definição 2 (Inscrições Estruturais) *Sejam $\langle P, T, F\rangle$ uma estrutura de rede e $\mathcal{D}$ um conjunto de domínios. Uma inscrição para a estrutura é uma tripla de funções totais $\langle c, p, g\rangle$, em que:*

- $c : P \rightarrow \mathcal{D}$ *é a função de domínios, denominada função de cores;*
- $p : F \rightarrow \mathcal{E}$ *é a função de pesos (ou de expressões) dos arcos, em que $\mathcal{E}$ é o conjunto de todas as expressões definíveis sobre os domínios de $\mathcal{D}$ (toda expressão de arco $p(f)$ é necessariamente do mesmo tipo associado ao lugar conectado ao arco);*
- $g : T \rightarrow \mathcal{E}_B$ *é a função de guardas das transições, na qual $\mathcal{E}_B$ é o conjunto de expressões booleanas.*

Uma das características mais importantes de redes de alto nível é o fato de que suas fichas transportam valores (historicamente denominados cores). Assim, uma marca ou uma ficha em redes de alto nível é um elemento de algum domínio. Uma marcação de um lugar é um multi-conjunto de marcas. Finalmente uma marcação de uma rede de alto nível é a associação de uma marcação a cada lugar. Naturalmente, exige-se que as marcações dos lugares "obedeçam" ao domínio associado a cada lugar. Estes conceitos são formalizados a seguir.

Definição 3 (Marcação) *Seja $\mathcal{D} = \{D_1, D_2, \ldots, D_n\}$ um conjunto finito de domínios contáveis, $\langle P, T, F\rangle$ uma estrutura de rede de Petri e $c : P \rightarrow \mathcal{D}$ uma função total que associa um domínio a cada lugar de P. Uma função*[1]

$$m : P \rightarrow \bigcup_{i=1}^{n} D_i^{mc}$$

é uma marcação se e somente se é total e respeita a função de domínios, ou seja:

$$\forall p \in P : m(p) \in c(p)^{mc}$$

Finalmente definimos uma rede de Petri de alto nível como sendo uma estrutura inscrita à qual associamos uma marcação inicial.

Definição 4 (Estrutura de Alto Nível) *Seja $\mathcal{D}$ um conjunto finito de tipos de dados. Uma estrutura de rede de Petri de alto nível (ou uma estrutura inscrita) baseada em D é um par $\langle N, i\rangle$, em que:*

[1] A família de funções $[X \rightarrow \mathbb{N}]$, chamada de multi-conjuntos de X, na qual $\mathbb{N}$ é o conjunto dos números naturais, será notada por X^{mc}.

Para descrever objetos é preciso definir um conjunto de atributos, um conjunto de métodos e um corpo. Os valores dos atributos são o estado do objeto; os métodos são um conjunto de eventos ou operações que podem ocorrer no objeto e alterar seu estado; o corpo determina em que circunstâncias os métodos são ativados e como alteram o valor dos atributos.

Como todo objeto é instância de uma classe, para modelarmos um sistema basta descrever todas as classes e uma configuração inicial de objetos (com a respectiva topologia de comunicação). Para descrever a classe, dois elementos são essênciais: sua interface e sua estrutura interna. A interface é definida como um conjunto de nomes de atributos e métodos, definidos através da linguagem utilizada para a descrição de tipos de dados. A estrutura interna é modelada através de uma rede de Petri de alto nível para a qual os elementos da interface são mapeados. Atributos são associados a lugares e métodos a transições, respeitando os tipos dos atributos e as assinaturas dos métodos.

Nesta abordagem de redes de Petri orientadas a objeto, definimos um modelo como sendo determinado por um conjunto de tipos de dados, um conjunto de classes e uma configuração inicial. Uma configuração é um possível estado do sistema. Assim, definimos uma configuração como um conjunto de instâncias de classes e suas respectivas marcações.

As definições a seguir caracterizam a categoria de redes de Petri orientadas a objetos aqui descritas. No restante deste artigo esta caracterização será referenciada por RPOO.

Definição 6 (Interface de uma Classe) *Seja Σ um conjunto de tipos de dados, cujo conjunto de domínios é $\{D_1, D_2, \ldots, D_n\}$. Uma interface é uma quádrupla $\langle A, \tau, M, a \rangle$, em que:*

- *A é um conjunto finito de nomes de atributos;*
- *$\tau : A \rightarrow \Sigma$ é uma função de tipos de atributos;*
- *M é um conjunto finito de nomes de métodos;*
- *$a : M \rightarrow \Sigma^* \cup (\Sigma^* \times \Sigma)$ é uma função de assinatura para os métodos;*
- *observando-se que $A \cap M = \emptyset$.*

Definição 7 (Classe) *Seja Σ um conjunto de tipos de dados. Uma classe é uma dupla $\langle I, \mathcal{N} \rangle$, em que:*

- *$I = \langle A, \tau, M, a \rangle$ é uma interface baseada em Σ;*
- *$\mathcal{N}$ é uma rede de Petri de alto nível baseada em Σ, denominada corpo da classe;*

Definição 8 (Configuração) *Seja C um conjunto finito de classes, $\vec{O}$ um conjunto infinito de nomes de objetos e $\mathcal{M} = \bigcup_{c \in C} \mathcal{M}_c$ o conjunto de todas as marcações possíveis para os corpos das classes de C. Uma configuração é uma tupla $\langle O, M \rangle$*

- *$O \in \vec{O} \times C$ é um conjunto finito de objetos;*
- *$M : O \rightarrow \mathcal{M}$*
- *tal que M respeita o corpo da classe do objeto, ou seja,*

$$(\forall \langle \vec{o}, c \rangle \in O) m(\langle \vec{o}, c \rangle) \in \mathcal{M}_c$$

Definição 9 (Modelo) *Um modelo é uma tripla* $\langle \Sigma, C, c_0 \rangle$, *em que:*

- Σ *é um conjunto de tipos de dados;*
- C *é um conjunto finito de classes que têm* Σ *como seus tipos de dados;*
- c_0 *é uma configuração inicial baseada em* C.

4 Herança e Redes de Petri Orientadas a Objeto

Diante das abordagens já tratadas sobre as peculiaridades relativas ao conceito de herança, nesta seção discutiremos tais mecanismos em termos de uma notação baseada em redes de Petri para a modelagem de sistemas distribuídos e concorrentes. Deve estar claro que tais conceitos devem ser suportados por mecanismos diferenciados, por se tratarem de visões distintas sobre o relacionamento de objetos e/ou classes.

O mecanismo que denominamos de herança semântica, possui um destacado interesse em relação ao contexto da notação aqui apresentada. Isto se deve, ao fato de servir de representação para o que pode ser considerado o mais importante mecanismo de abstração da orientação a objetos, e ao fato de que permite estabelecer uma compatibilidade semântica entre os componentes relacionados (classes de objetos, neste caso) cuja verificação pode se dar automaticamente.

A herança sintática, colocada aqui como um segundo mecanismo de herança, é tratada com o objetivo prático de favorecer a eficiência de modelagem de sistemas. Isto é conseguido permitindo o reaproveitamento parcial dos modelos de classes previamente descritas. Neste caso, entretanto, nenhum tipo de compatibilidade é assegurado entre as classes relacionadas.

Segundo América, os dois mecanismos aqui descritos podem ser considerados como diferentes formas de *observar* os objetos. No primeiro caso, os objetos são observados externamente e interessa sob que circunstâncias podem ser utilizados e em que contextos podem aparecer – uma questão de tipos. No segundo, observa-se o interior dos objetos, interessando quais estruturas internas podem ser reutilizadas – uma questão de classes [3].

Nesta seção, detalhamos e exemplificamos a introdução de herança sintática e semântica no contexto do modelo RPOO introduzido.

4.1 Herança Sintática

A introdução do conceito de herança sintática no modelo RPOO apresentado visa aumentar o poder de descrição ou expressão do modelo. Este aumento de poder de expressão intencional uma economia na descrição das classes. Assim, torna-se possível reaproveitar partes de descrições, estabelecendo uma hierarquia entre as classes, na qual uma classe descendente herda a estrutura da sua classe ancestral.

Considerando como classes básicas são descritas como estabelecido para o modelo RPOO (ver Seção 3), vamos definir o mecanismo pelo qual é possível descrever uma nova classe a partir de outra. As questões relativas a este tipo de herança [13, 15] devem ser agora mapeadas para o universo de RPOO, no qual as particularidades das redes de Petri marcam as diferenças.

A questão mais relevante neste ponto diz respeito às formas como as descrições podem ser herdadas na constituição de uma nova classe. A operação de criação de uma classe a partir de outra pode comportar as seguintes ações:

1. cancelar propriedades (descrições de atributos ou métodos) presentes na classe ancestral;
2. redefinir algumas delas;
3. acrescentar outras novas.

Definir quais dessas ações estarão disponíveis irá caracterizar o mecanismo de herança sintática. Tendo-se em conta que o objetivo de tal mecanismo é de natureza prática, as três ações possíveis no processo de herança são justificáveis. O cancelamento de propriedades é útil quando se pretende que uma classe seja declarada como subclasse de outra simplesmente para aproveitar uma parte das suas descrições (principalmente de métodos), enquanto que as demais não interessam de modo algum, sendo portanto desabilitadas. A redefinição de métodos é aplicada quando há o desejo de alterar a descrição de um determinado método, porém preservando o seu nome. A possibilidade de uma classe possuir propriedades não declaradas em sua classe ancestral é garantida através do acréscimo de novas propriedades.

Restrições podem ser feitas com relação às ações disponíveis num processo de herança, com o objetivo de estabelecer níveis de compatibilidade entre as classes relacionadas (ex. compatibilidade de assinatura). Porém estas restrições relacionam-se a preocupações quanto ao comportamento dos objetos das classes e já estão sendo tratadas neste trabalho no contexto de herança semântica. Logo, o que chamamos aqui de herança sintática é uma operação de criação de uma nova classe a partir de outra já existente, sendo possível cancelar, redefinir e adicionar propriedades livremente. Dessa forma, nenhum tipo de compatibilidade pode ser garantida entre as classes.

Uma das características mais interessantes deste tipo de herança nas linguagens de programação orientadas a objetos é que o código herdado é na verdade compartilhado entre as classes, havendo apenas uma única implementação para ele. Desse modo, futuras manutenções no sistema são simplificadas, pois uma parte do código, embora compartilhada por várias classes, é conservada em um só local. Devido ao fato de que os atributos, os métodos e suas interações estão definidos na estrutura interna das classes do modelo RPOO, o mecanismo aqui tratado corresponde simplesmente a *copiar e colar* as descrições. A formalização de tal mecanismo é portanto óbvia e não será detalhada neste trabalho.

4.2 Herança Semântica

Por se tratar de uma ferramenta para especificação formal de sistemas, é importante que RPOO suporte principalmente uma representação adequada da especialização conceitual. Um mecanismo de herança que represente o relacionamento "é-um" deve considerar aspectos comportamentais dos objetos. Isso porque pretende-se garantir que objetos de uma (sub)classe possam ser manipulados também como objetos da sua super classe. Desse modo, é apropriada a associação de tal mecanismo aos conceitos sobre tipos, discutidos na Seção 2.2.

Em uma computação seqüencial, na qual as operações podem ser vistas como seqüências finitas de ações sem efeitos colaterais, podemos simplificar o entendimento do comportamento das entidades através de conjuntos de entrada e saída (modelo funcional). Assim, considerando o comportamento de um sistema orientado a objetos sob uma perspectiva puramente seqüencial, é possível caracterizar o comportamento dos objetos simplesmente através de restrições nas suas assinaturas—nomes e tipos dos atributos, métodos e de seus parâmetros. Algumas linguagens de programação e ferramentas para a especificação formal utilizam esta simplificação para

caracterizar compatibilidades comportamentais entre classes relacionadas por herança [8, 9]. Porém, ao considerarmos as características interativas da computação orientada a objetos concorrente, a compreensão das operações sob um ponto de vista puramente funcional não é adequada [12]. Os métodos dos objetos, além de poderem provocar a alteração dos seus estados, podem ter o seu fluxo de execução interrompido por interações com outros objetos no sistema, ou eventualmente por linhas paralelas de controle existentes no próprio objeto. Desta forma, uma noção de comportamento mais apropriada precisa ser definida.

O que pretendemos com a herança semântica é a associação de tipos às classes. Desta forma fazemos coincidir o conceito de herança às definições de subtipificação, discutidas na Seção 2.2. Assim, este mecanismo estabelece categorias de compatibilidade de comportamento relacionando definições alternativas de equivalência entre as redes que compõem a estrutura interna dos objetos. Relações de equivalência distintas levam a hierarquias de herança também distintas.

Com relação às classes definidas na Seção 3, os principais aspectos que nos interessam estudar, para caracterizar noções adequadas de equivalência, são:

Dinâmica da Interface: a comparação entre classes busca relacionar como o conjunto de métodos habilitados varia durante o ciclo de vida dos objetos;

Espaço de Estados: consiste em encontrar semelhanças entre as classes através da preservação de certas propriedades associadas aos respectivos espaços de estados;

Seqüências de Eventos: abordagem semelhante à anterior, sendo que a ênfase é dada às possíveis seqüências de eventos que podem ocorrer nas classes;

Para cada aspecto descrito, é possível definir inúmeras variações sobre quais elementos (ou combinações deles) serão considerados na definição de uma relação. A questão reside, portanto, em determinar tais elementos de forma a obter relações que melhor se aproximem da especialização conceitual.

O objetivo principal deste mecanismo é disciplinar a interação entre objetos em sistemas distribuídos abertos [1, 2], nos quais a topologia de interconexão é naturalmente dinâmica. é preciso garantir que objetos de uma (sub)classe possam substituir objetos da sua superclasse em interações com outros objetos (princípio de substituição[5] em termos de objetos concorrentes). Mesmo sendo possível encontrar uma definição que atenda completamente ao princípio citado, nem sempre é esta a relação de comportamento desejada entre superclasse e subclasse. Tal princípio restringe fortemente as possíveis relações hierárquicas que preservam certas propriedades comportamentais.

Os exemplos apresentados a seguir têm o propósito de discutir possíveis relações de equivalência. Eles se relacionam ao conhecido problema do *jantar dos filósofos*.

Na Figura 1 apresentamos uma descrição de um possível conjunto de classes para construir um modelo para solucionar este problema. Utilizamos a descrição em rede de Petri Colorida (Coloured Petri Nets - CPN) [6, 7] com o objetivo de usar a ferramenta Design/CPN [14] para simulação e análise dos modelos. A rede que representa a classe dos filósofos é composta por quatro lugares (*pensando*, *espera_garfo*, *pronto_para_comer* e *comendo*) e quatro transições. De acordo com o conjunto de declarações da rede, os lugares *pensando* e *comendo* possuem cor do tipo E (ficha sem informação). A presença de ficha em um dos dois lugares indica o estado do filósofo.

[5]Tradução para *substitutability principle*.

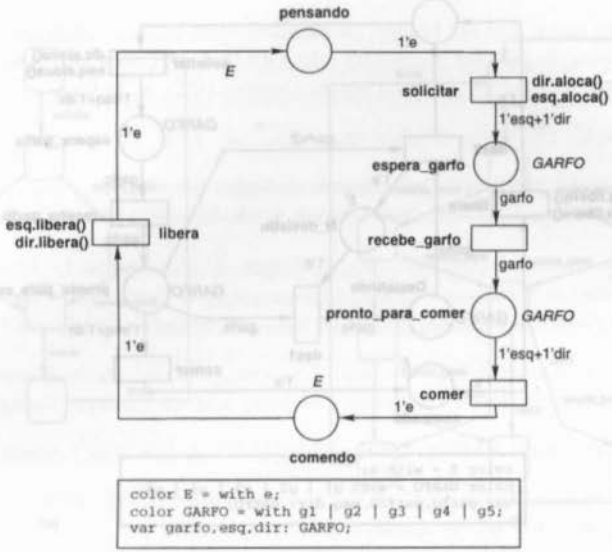

Figura 1: Classe represetando o comportanto de um filósofo

Um filósofo está relacionado a dois garfos: um à sua direita e outro à sua esquerda. Uma vez pensando, um filósofo pode solicitar os garfos da direita e da esquerda para que possa comer. Para tanto, ele envia mensagens solicitando alocação para os dois garfos, representada no modelo pelas expressões *esq.aloca()* e *dir.aloca()* associadas à transição *solicitar*. Após o disparo da transição *solicitar* fichas são colocadas no lugar *espera_garfo* indicando a espera da disponibilização dos garfos. Após a confirmação da disponibilização dos garfos (fichas no lugar *pronto_para_comer*) a transição que pode levar o filósofo a comer torna-se habilitada. Para retornar a pensar, é necessário que ele libere os garfos utilizados, para que possam servir a outros filósofos (expressões *esq.libera()* e *dir.libera()*).

Os garfos por sua vez podem estar em dois estados, *alocado* ou *livre*, que podem ser alterados em resposta a solicitações (envio de mensagens) dos filósofos da vizinhança.

Um possível sistema é determinado por uma configuração composta por n filósofos e n garfos, na qual cada filósofo "conhece" dois garfos (esquerda e direita), dispostos de forma circular. Como não é possível garantir que os garfos da direita e da esquerda receberão as mensagens de alocação ao mesmo tempo, o sistema em questão apresenta estados de impasse (*deadlock*), relativos à alocação dos garfos da direita ou dos garfos da esquerda por todos os filósofos.

De acordo com o princípio da substituição definido em relação à herança semântica, podemos desejar definir uma nova classe de filósofos que em substituição a filósofos da primeira classe (na configuração acima citada) possa prover algum tipo de melhoria no modelo. Porém se considerarmos a herança semântica em termos muito restritos, com objetos de uma sub(classe) comportando-se *exatamente* como objetos da superclasse, dificilmente introduziremos melhorias nos modelos dos sistemas.

A classe descrita na Figura 2 representa filósofos que além de comer e pensar podem

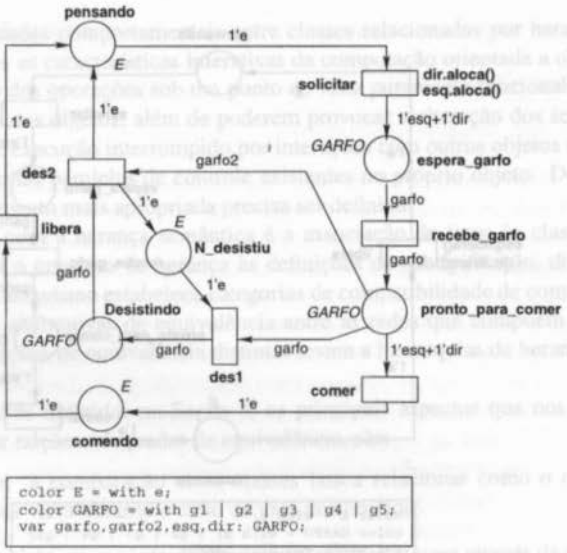

Figura 2: Classe represetando o comportanto modificado de um filósofo

também, uma vez conseguido algum garfo (ou os dois), desistir de comer (disparo das transições *des1* e *des2*). Filósofos desta classe, dentro de configurações como a já descrita, eliminam a possibilidade de estados de impasse, embora introduza situações de inanição (*starvation*). Pode-se agora sair de estados que eram mortos em configurações nas quais apareciam filósofos da primeira classe.

Comparando os espaços de estado dos filósofos das primeira e segunda classe gerados pelo Design/CPN (Figura 3), observamos que o primeiro é um sub-grafo do segundo. Em outras palavras, existe uma seqüência de transições no espaço de estados dos filósofos da segunda classe que é semelhante a todas as transições presentes no espaço de estados de filósofos da primeira classe. Porém uma série de outras transições podem alterar o estado interno dos filósofos da segunda classe de maneira diferente em relação aos da primeira classe, o que pode alterar a dinâmica da interface. Com isso, podemos verificar que mesmo classes que não preservam estritamente o comportamento podem ser úteis e preservar a substituição. Dessa forma as relações entre espaços de estado podem constituir uma interessante equivalência entre classes.

5 Conclusões

Neste trabalho discutiu-se de forma detalhada as questões relacionadas com o conceito de herança e modelos de rede de Petri orientados a objetos. As principais motivações que justificam tal investimento estão embasadas na utilidade da inclusão de algum mecanismo de herança em uma notação orientada a objetos permitindo tanto a representação de especialização conceitual como o reaproveitamento (reuso) de código.

Além disto a necessidade de introduzir uma metodologia formal para lidar com sistemas

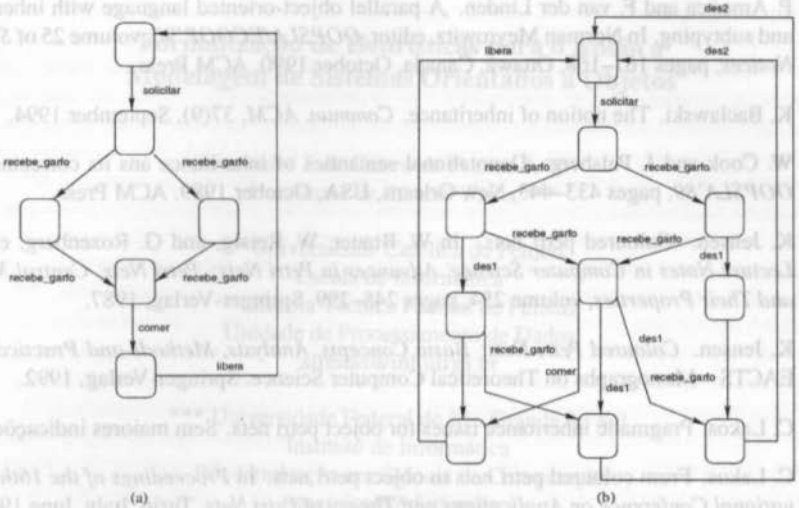

Figura 3: Comportamentos para os filósofos da primeira classe (a) e segunda classe (b)

distribuídos e orientados a eventos, com o objetivo de disponibilizar mecanismos formais de análise e verificação motivou o investimento num modelo capaz de lidar com concorrência de fato e não determinismo. Deste modo, adotamos as redes de Petri e redes de Petri de alto-nível, além dos conceitos de orientação a objetos, como base teórica e conceitual para a introdução de um modelo de redes de Petri orientadas a objeto.

No contexto do modelo de rede de Petri Orientada a Objeto, a introdução de mecanismos de herança tem sido motivada pela aplicação do formalismo em problemas reais. O exemplo introduzido neste artigo com base no clássico problema dos filósofos serviu para demonstrar como os mecanismos de herança podem ser introduzidos no modelo de rede de Petri Orientada a Objeto considerado.

O desenvolvimento tanto do formalismo aqui introduzido bem como métodos de análise e verificação de modelos vêm sendo considerados. Além disto a implementação de uma ferramenta de suporte a modelagem está sendo implementada.

Referências

[1] G. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems*. MIT Press, Cambridge, Massachusetts, 1986.

[2] G. Agha, S. Frolund, W. Kim, R. Panwar, A. Patterson, and D. Sturman. Abstraction and modularity mechanisms for concurrent computing. In G. Agha, Peter W., and A. Yonezawa, editors, *Research Directions in Concurrent Object-Oriented Programming*. MIT Press, Cambridge, Massachussetts, 1993.

[3] P. America and F. van der Linden. A parallel object-oriented language with inheritance and subtyping. In Norman Meyrowitz, editor, *OOPSLA/ECOOP'90*, volume 25 of *Sigplan Notices*, pages 161–168, Ottawa, Canada, October 1990. ACM Press.

[4] K. Baclawski. The notion of inheritance. *Commun. ACM*, 37(9), September 1994.

[5] W. Cook and J. Palsberg. Denotational semantics of inheritance ans its correctnes. In *OOPSLA'89*, pages 433–443, New Orleans, USA, October 1989. ACM Press.

[6] K. Jensen. Coloured petri nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Lecture Notes in Computer Science, Advances in Petri Nets: Petri Nets, Central Models and Their Properties*, volume 254, pages 248–299. Springer-Verlag, 1987.

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

[8] C. Lakos. Pragmatic inheritance issues for object petri nets. Sem maiores indicações.

[9] C. Lakos. From coloured petri nets to object petri nets. In *Proceedings of the 16th International Conference on Applications and Theory of Petri Nets*, Turin, Italy, June 1995.

[10] J. Palsberg and M. I. Schwartzbach. Three discussions on object-oriented typing. *ACM SIGPLAN OOPS*, 3(2):31–38, 1992.

[11] J. Palsberg and M. I. Schwartzbach. Static typing for object-oriented programming. *Science of Computer Programming*, 23(1):19–53, 1994.

[12] L. Pomello, G. Rozenberg, and C. Simone. A survey of equivalence notions for net based systems. In G. Rozenberg, editor, *Advances in Petri Nets 1992*, volume 609 of *Lecture Notes in Computer Science*, pages 410–472. Springer-Verlag, 1992.

[13] A. Taivalsaari. On the notion of inheritance. *ACM Computing Surveys*, 28(3), September 1996.

[14] University of Aarhus, Aarhus, Denmark. *Design CPN - Overview of CPN ML Syntax, version 3.0*, 1996.

[15] P. Wegner. The object-oriented classification paradigm. In B. Shriver and P. Wegner, editors, *Research Directions in Object-Oriented Programming*, pages 479–560. MIT Press, 1987.

[16] P. Wegner and S. B. Zdonik. Inheritance as an incremental modification mechanism or what like is and isn't like. In *ECOOP'88: European Conference on Object-oriented Programming*, volume 276 of *Lecture Notes in Computer Science*, pages 55–77, Oslo, Norway, August 1988. Springer-Verlag.