Da especificação à verificação de agentes móveis - Um ambiente gráfico

André G. Andrade*
Universidade de São Paulo, IME,
São Paulo, Brasil
oberon@ime.usp.br

and

Ana C. V. de Melo
Universidade de São Paulo, IME,
São Paulo, Brasil
acvm@ime.usp.br

and

Marcelo M. Amorim
Universidade de São Paulo, IME,
São Paulo, Brasil
mamorim@ime.usp.br

Abstract

Neste trabalho, apresentamos um ambiente de especificação e verificação para π-calculus. O ambiente proposto está inicialmente constituído de dois módulos: (i) especificação e representação gráfica; (ii) simulação e verificação de equivalências. O primeiro módulo permite a especificação de processos em π-calculus com recursos de visualização e representação gráfica da especificação. O seguinte torna possível a simulação e verificação de bi-simulações entre processos descritos em π-calculus utilizando uma nova abordagem através de técnicas de normalização e bi-simulação *up-to* em verificações baseadas em autômatos.

Keywords: Verificação Formal, Agentes Móveis, Pi-Calculus, Especificação Formal

1 Introdução

Fatores como o crescimento da comunicação celular, redes locais sem fio, e serviços via satélite, nos quais as informações e recursos podem ser acessados e utilizados em qualquer lugar e a qualquer momento, têm gerado grande impacto na área de computação concorrente. Programas executados em sistemas paralelos ou distribuídos são claramente mais difíceis de ter seus comportamentos previstos que sistemas seqüenciais, devido aos mecanismos de composição inerentes aos mesmos. Agentes móveis e colaboradores, além de possuírem as dificuldades apresentadas por sistemas concorrentes, carregam ainda problemas relacionados ao fato deles próprios, e o ambiente no qual estão inseridos, poderem ser reconfigurados dinamicamente (semântica de processos de alta-ordem [21, 18]).

O desenvolvimento formal (ou semi–formal) de agentes móveis requer um fundamento matemático que dê suporte à especificação e verificação tanto dos agentes individuais quanto do sistema como um todo. O π-calculus [13] é um modelo matemático de processos que torna possível descrever e analisar o comportamento das comunicações efetuadas entre agentes móveis. A representação de mobilidade em π-calculus pode ser expressada através do poder de reconfiguração em tempo real dos agentes, onde a transferência de portas de comunicação entre eles se faz possível. Ele tem sido amplamente utilizado nas especificações de sistemas de agentes móveis [15, 18, 23, 5] e pode ser considerado como base fundamental para as linguagens concorrentes

*Com auxílio à pesquisa do CNPq

com características de mobilidade, da mesma maneira que o λ-calculus é base fundamental para as linguagens funcionais.

Embora existam várias ferramentas de software disponíveis para trabalhar com π-calculus, a grande maioria delas exige do usuário uma infalibilidade em relação à sintaxe: tanto a entrada quanto a saída de dados são difíceis de serem visualizadas, constituindo-se de longas linhas de texto. A conseqüência da dificuldade de especificação de agentes móveis é a desistência do uso de ferramentas principlamente por novos usuários. Além disso, não há um padrão de símbolos adotado por todos os verificadores para π-calculus: cada verificador adota um conjunto próprio de símbolos, o que dificulta ainda mais o uso das várias ferramentas.

Neste trabalho, apresentamos um ambiente integrado para especificação e verificação de agentes em π-calculus constituído pelo editor gráfico PIG (um acrônimo para *Pi-calculus Gráfico*) [6], e pelo verificador VTUBAINA (*A Verification Tool for Up-to Bisimulation and Automata INtegration Automatization*) [1]. Este ambiente tem como objetivo facilitar a especificação de agentes móveis em π-calculus, dado o recurso gráfico de edição, e a verificação de agentes a partir do próprio ambiente gráfico.

2 π-Calculus e a Representação de Agentes Móveis

O π-calculus tem sido amplamente utilizado nas especificações de sistemas de agentes móveis [15, 18, 23, 5]. Ele pode ser considerado como base fundamental para as linguagens concorrentes da mesma maneira que o λ-calculus é base fundamental para as linguagens funcionais.

Suponha os processos S, P e C como um servidor, uma impressora, e um cliente respectivamente [1], onde o servidor S controla o acesso à impressora, a qual o cliente C gostaria de utilizar (conforme mostrado na Figura 1).

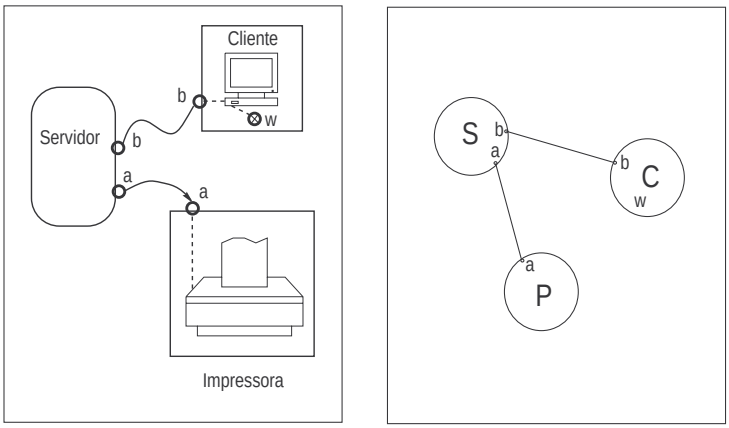

Figura 1: Caracterização dos processos antes da interação. Na direita são mostrados os processos S,P, e C, caracterizando respectivamente o Servidor, Impressora, e Cliente.

Os processos que caracterizam o servidor, cliente, e impressora podem ser especificados em π-calculus, respectivamente, da seguinte forma conforme 2:

$S \stackrel{\text{def}}{=} \overline{b}a.S'$ S tem a capacidade de enviar uma informação (nome a) através da porta b, e passa a se comportar como S'. Neste caso, a informação passada é uma porta de comunicação, isto é, o nome a representa um nome de uma porta de comunicação.

$C \stackrel{\text{def}}{=} b(w).\overline{w}z.C'$ C tem a capacidade de receber uma informação pela porta b, informação referenciada pelo nome w (que neste caso, receberá uma porta de comunicação). Após isto, C tem a capacidade de enviar por w um dado z, passando a se comportar como C'.

$P \stackrel{\text{def}}{=} a(d).P'$ P tem a capacidade de receber uma informação pela porta a que será referenciada por d, e passar a se comportar como P'.

Com o sistema rodando de forma concorrente $(S \mid C \mid P)$ temos:

$$\overbrace{\overline{b}a.S'}^{S} \mid \overbrace{b(w).\overline{w}z.C'}^{C} \mid \overbrace{a(d).P'}^{P}$$

Os processos S e C podem realizar uma comunicação interna entre eles através da porta b, a qual é representada por $\stackrel{\tau}{\longrightarrow}$. Se essa comunicação interna for realizada, então teremos o processo S passando a se

[1] Exemplo retirado de textos introdutórios sobre π-calculus [13, 17]

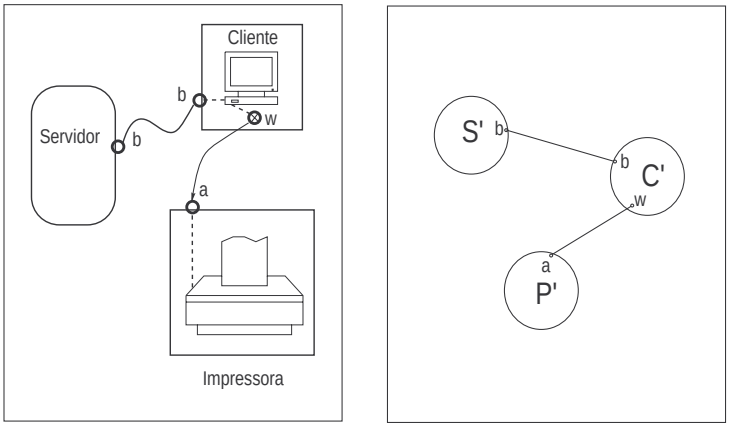

Figura 2: Caracterização dos processos depois da interação.

comportar como S', e em C ocorre a substituição $\{a/w\}$ (substitui-se todos os nomes w por a) [2].

$$(\overline{b}a.S' \mid b(w).\overline{w}z.C' \mid a(d).P') \xrightarrow{\tau} (S' \mid \overline{a}z.C') \mid a(d).P'$$

Após esta comunicação, os processos C e P podem se comunicar através da porta a, ou seja, C pode enviar um dado z para P que irá receber a informação que será referenciada por d em P.

$$(S' \mid \overline{a}z.C' \mid a(d).P') \xrightarrow{\tau} (S' \mid C' \mid P')$$

Em π-calculus, todos os elementos são representados por nomes. Dentro de um conjunto infinito de nomes $\mathcal{N}$, as portas de comunicação, as variáveis e os dados são representados por a,b,c,...,z, os agentes são representados por P,Q,R, e os processos são definidos com a seguinte sintaxe:

$$P ::= \mathbf{O} \mid \alpha.P \mid (\boldsymbol{\nu}x)P \mid (P_1 \mid P_2) \mid (P_1 + P_2) \mid !P$$

Os prefixos $\alpha.P$ podem representar respectivamente uma entrada, uma saída, ou uma ação silenciosa τ: $\alpha ::= a(b) \mid \overline{a}b \mid \tau$

No operador de *Restrição* ($(\boldsymbol{\nu}x)P$) o processo se comporta como P mas o nome x é local, e não pode ser visto pelo ambiente (por outros processos). Como em outras algebras de processos, a restrição $(\boldsymbol{\nu}x)P$ define x sendo um nome local em P (em CCS isto é escrito como $\backslash x$), e desta forma é chamado de *bound name*. Um segundo caso onde um nome pode ser *bound* é quando temos $y(x).P$, onde x é uma variável para qualquer nome que possa ser recebido através do link y. Um nome é chamado de *nome livre* quando ele não é um *nome bound*. Por exemplo, em $\overline{x}z.P$ e em $y(a).Q$, temos que x, z, e y são *livres* pois não dependem de ninguém. O conjunto de nomes livres (*free names*) em α é dado por $\mathbf{fn}(\alpha)$, e $\mathbf{n}(\alpha) = \mathbf{bn}(\alpha) \cup \mathbf{fn}(\alpha)$.

Os operadores $(P_1 \mid P_2)$ e $(P_1 + P_2)$ representam, respectivamente, a composição paralela e a escolha entre os processos. $!P$ representa a replicação do processo P (existem tantas cópias do processo P quanto for necessário - um número ilimitado).

3 Ambiente de Especificação e Verificação

O ambiente de especificação e verificação proposto para π-calculus utiliza, até o momento, um módulo para especifcação gráfica de agentes móveis e um módulo para a verificação de equivalências entre os agentes. Este dois módulos foram inicialmente produzidos como duas ferramentas distintas, PiG e VTubaina, as quais foram posteriormente integradas ao ambiente. Dessa forma, mantivemos os dois módulos com os nomes das ferramentas originais. A idéia é utilizar a PiG para realizar as especificações dos agentes, bem como a visualização gráfica de cada um deles. A verificação é então realizada pela VTubaina, a qual importa o arquivo de especificação gerado a partir de agentes descritos na PiG. A Figura 3 descreve o diagrama de funcionamento do ambiente.

[2] O nome w no processo C se comporta como um nome que servirá como referência ao valor recebido pela porta b. Em CCS [12], somente é permitido que sejam passados valores na comunicação entre os processos, o que não acontece em π-calculus onde é permitia a passagem de portas de comunicação, e processos durante as comunicações.

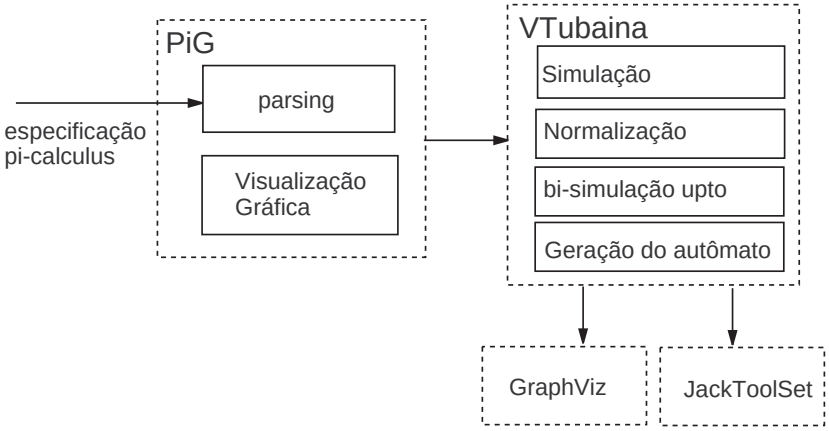

Figura 3: Diagrama do ambiente de especificação e verificação.

3.1 PiG

Apesar de ser um cálculo de referência para agentes móveis e para outros cálculos que tratam mobilidade, os estudos sobre π-calculus se concentram em aspectos teóricos relacionados à expressividade e relações de equivalência computáveis. Desta forma, grande parte das ferramentas desenvolvidas para π-calculus são protótipos que visam a certificação de aspectos teóricos desenvolvidos, sem vislumbrar o usuário final que deseja especificar os seus problemas reais. Por essa razão, alguns problemas são evidentes: a edição texto dos agentes é um processo longo e muitas vezes não intuitivo (principalmente para os novatos); e como a edição é em modo texto, cada ferramenta adota um conjunto de símbolos para representar os operadores π-calculus, e o usuário precisa aprender "uma nova linguagem" para o uso de cada ferramenta.

O módulo PiG - um acrônimo para *π-calculus Gráfico* - tem como objetivo prover um ambiente amigável para especificação de agentes móveis utilizando π-calculus. Neste ambiente, a entrada e a saída de dados podem ser feitas graficamente, agilizando o trabalho do usuário. Como uma árvore sintática é usada internamente, este módulo foi projetado para ser usado juntamente com ferramentas de verificação, funcionando como um *front-end* para as mesmas. Assim, a PiG poderá ser integrado a verificadores existentes para dar suporte à edição gráfica dos agentes em π-calculus. O ambiente apresentado neste artigo é o primeiro passo dessa integração com a VTubaina.

A PiG foi desenvolvida com o objetivo de ser um ambiente altamente configurável e adaptável, podendo lidar ao mesmo tempo com várias representações gráficas do mesmo modelo. Atualmente encontram-se implementadas as Pi-Nets e a linguagem gráfica padrão, que descreveremos e exemplificaremos no decorrer do artigo. Entretanto outras formas de representação gráfica podem ser acopladas à PiG, e a cada modificação da especificação graficamente ou textualmente causa a atualização de todas as outras representações gráficas. Isso permite adotar a representação gráfica que for mais adequada para o tipo de trabalho a ser efetuado.

Existem basicamente duas formas de edição: a introdução direta dos agentes através de uma linguagem textual e a forma gráfica através de uma barra de ferramentas. Essas formas de edição podem ser usadas em conjunto também, sendo que a edição gráfica atualiza a representação textual do processo, e o mesmo ocorre com a edição textual.

Como a PiG é configurável, pode-se criar módulos que se acoplem a ela para permitir ainda novas formas de edição, como por exemplo, editar diretamente a árvore sintática do processo.

A linguagem gráfica utilizada é composta por alguns elementos básicos: ações, processos e operadores. As ações são representadas através de uma caixa e um cone, exceto a ação silenciosa, representada por um pequeno círculo. A representação de um processo é dada pela definição de um processo. O **O** é representado através de um círculo vermelho. Os outros processos através de um quadrado de bordas arredondadas. À exceção da prefixação, todos os outros operadores são representados através de símbolos parecidos com suas representações textuais usuais

O exemplo apresentado, Servidor-Impressora-Cliente, pode ser representado no módulo gráfico como segue:

Como podemos ver na Figura 4, temos a definição de um processo *Press* que é a composição dos processos Servidor, Impressora e Cliente: as duplas barras verticais representam a composição, as ações de entrada e saída levando aos processos *Sl, Cl, Pl*.

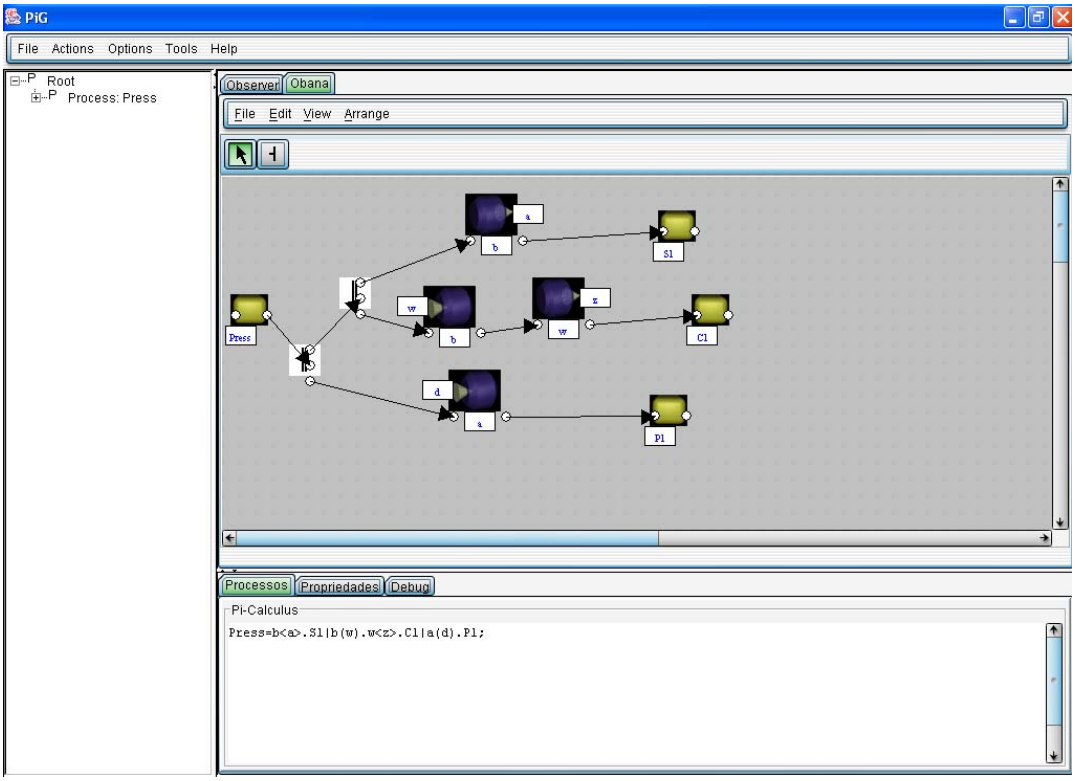

Figura 4: Especificação dos agentes Impressora-Servidor-Cliente

3.1.1 Da PiG para VTubaina

A PiG tem como representação interna uma árvore sintática que agrega a definição de todos os agentes. Através dela, toda a representação gráfica pode ser gerada, bem como toda a representação texto. É a essa árvore que todos os módulos desenvolvidos para a PiG têm acesso e podem modificar.

Para gerar um ambiente capaz de acoplar a PiG à VTubaina, foi necessário apenas fazer com que esse pudesse importar as árvores provenientes da PiG e então gerar os agentes correspondentes descritos na linguagem aceitável pela VTubaina. Assim, em um ambiente gráfico, o usuário pode requisitar a verificação de agentes editados na PiG, e o ambiente gera, de forma transparente ao usuário, os agentes na linguagem da VTubaina e procede a verificação solicitada.

3.2 VTubaina

Para que se possa formalizar como os agentes se comportam, e de alguma forma saber que nível de semelhança existe entre eles, é necessário definir relações de equivalências comportamentais sobre agentes. Na maioria das álgebras de processos [12, 9, 13], a bi-simulação é a idéia matemática mais comum utilizada para estabelecer as equivalências, sob o ponto de vista comportamental, entre processos concorrentes. Em π-calculus, a instanciação de nomes promove um aumento de expressividade no cálculo, mas afeta dramaticamente a teoria, de forma que novas técnicas de verificação precisaram ser desenvolvidas. No presente trabalho, além de implementar algumas técnicas de verificação existentes, implementamos também uma nova técnica de verificação desenvolvida em [1]. VTubaina (*A Verification Tool for Up-to Bisimulation and Automata INtegration Automatization*) é a ferramenta de verificação desenvolvida em [1].

Alguns trabalhos têm sido desenvolvidos para verificação automática em π-calculus [22, 3, 8, 4], e a descoberta de melhorias e novos métodos têm despertado interesse na área de verificação formal em computação móvel. As técnicas atuais para verificação de bi-simulações em π-calculus podem ser subdivididas nas abordagens: semântica e sintática. A abordagem semântica utiliza basicamente os algoritmos de particionamento (*partiton refinement algorithm*) [16, 9], os quais constroem os desdobramentos dos processos (geram todas as possíveis transições), ou os algoritmos baseadas na construção dos sistemas de transição dos agentes à medida que a verificação é realizada (*on-the-fly*) [2]. A abordagem sintática se baseia em métodos de prova *up-to bi-simulação* [19, 20, 8] e é implementada em sistemas de reescritura.

Cada uma das abordagens acima citadas tem suas vantagens: a sintática, na eficiência, e a semântica, na abrangência de agentes com comportamentos semelhantes. Tais diferenças despertou o interesse pelo estudo de uma nova técnica de verificação para π-calculus realizada em [1], a qual utiliza os algoritmos de normal-

ização e de bi-simulação *up-to* (provenientes da abordagem sintática) no auxílio da construção dos autômatos de transição que serão utilizados para verificação (o algoritmo de particionamento - abordagem semântica). Esta nova técnica, bem como as técnicas sintática e semântica, estão implementadas na VTubaina.

3.2.1 Uma nova técnica

Em π-calculus, as duas abordagens de verificação citadas acima têm sido alvo de estudos. A sintática usa a comparação sintática das estruturas dos agentes (expressões), sistemas de normalização e técnicas de prova por bi-simulações *up-to* [19, 20, 8]. Na abordagem semântica, Montanari e Pistore [14] sugerem uma técnica caracterizada pela construção de autômatos de transição dos agentes utilizando algoritmos de refinamento de partições para a verificação de equivalências.

O interesse despertado por essas duas técnicas nos levou a uma proposta de utilização de técnicas de verificação sintática no momento da construção dos desdobramentos dos processos. Devido à ação de entrada poder receber qualquer nome e levar a uma explosão exponencial do número de estados, existe uma preocupação em encontrar processos sintaticamente iguais durante a construção dos autômatos de transição dos agentes. Montanari e Pistore utilizam, para isso, buscas por mapeamento de nomes (*função bijetiva*) e, ao detectar processos semelhantes em estados diferentes, o autômato é reduzido.

Com a utilização apenas de uma busca por processos semelhantes através de um mapeamento de nomes, não conseguimos identificar processos equivalentes e que poderiam estar relacionados em um mesmo estado do autômato. Ao aplicarmos uma verificação sintática baseada em sistemas de normalização e técnicas de prova de bi-simulações *up-to* [19, 20, 8], podemos compactar ainda mais o autômato final. Ao aplicarmos esta técnica durante a verificação entre um processo que está sendo atingido e outros que já foram atingidos no momento da construção do desdobramento das transições do processo, conseguimos detectar um número maior de processos que podem ser agrupados em um mesmo estado do autômato. A diferença de compactação usando a técnica de Montanari e Pistore e a técnica proposta pelo autores [3] será ilustrada no Exemplo 1.

3.2.2 VTubaina: Serviços Fornecidos

A ferramenta VTubaina é o resultado dos estudos sobre técnicas de verificação para π-calculuse, por isso, implementa tanto as técnicas de verificação por refinamento de partições, sistemas de normalização e prova por bi-simulações *up-to*, quanto a nova técnica proposta. A ferramenta efetua as seguintes tarefas:

- Simula as transições de um processo;
- Detecta e visualiza *nomes livres* e *nomes bound* de um processo;
- Aplica apenas o sistema de normalização em um processo, visualizando passo a passo, as regras aplicadas e os resultados gerados na expressão;
- Verifica congruência estrutural entre dois processos;
- Verifica bi-simulações *up-to* entre dois processos;
- Gera o autômato final de transição do processo com desdobramento utilizando apenas os *nomes ativos*;
- Gera o autômato final utilizando a verificação sintática durante os desdobramentos;
- Imprime o autômato final em formato .fc2 (Jack tool Set[7]);
- Imprime o autômato final em formato .dot (dotty - Graph Editor[10, 11]).

A funcionalidade principal deste módulo é de possibilitar a descrição e utilização simultânea das duas abordagens de verificação, tanto semântica (particionamento), quanto sintática (congruência estrutural).

3.2.3 Exemplo de Uso: Geração do Autômato e Verificação

Para ilustrarmos o funcionamento de verificação entre processos descritos em π-calculus no ambiente proposto, especificamos os agentes descritos no Exemplo 1 abaixo no módulo gráfico do ambiente, ilustrado na Figura 5.

Exemplo 1

$$T_1 \stackrel{\text{def}}{=} (\boldsymbol{\nu} a)(a(x).a(x) \mid a(x).\overline{x}b \mid \overline{a}c.a(x) \mid \overline{a}c)$$

$$T_2 \stackrel{\text{def}}{=} (\boldsymbol{\nu} a)(a(x).a(x) \mid a(x).\overline{x}b \mid \overline{a}c.a(x) \mid \overline{a}c.a(x))$$

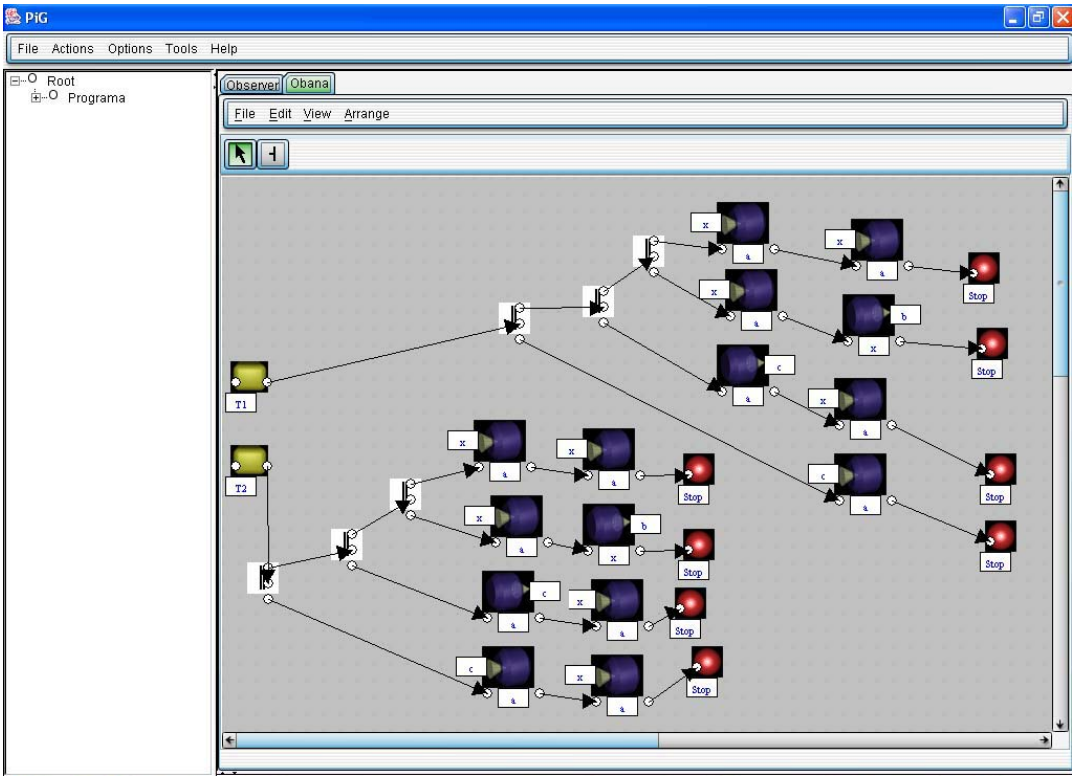

Figura 5: Agentes T_1 e T_2 especificados na PIG

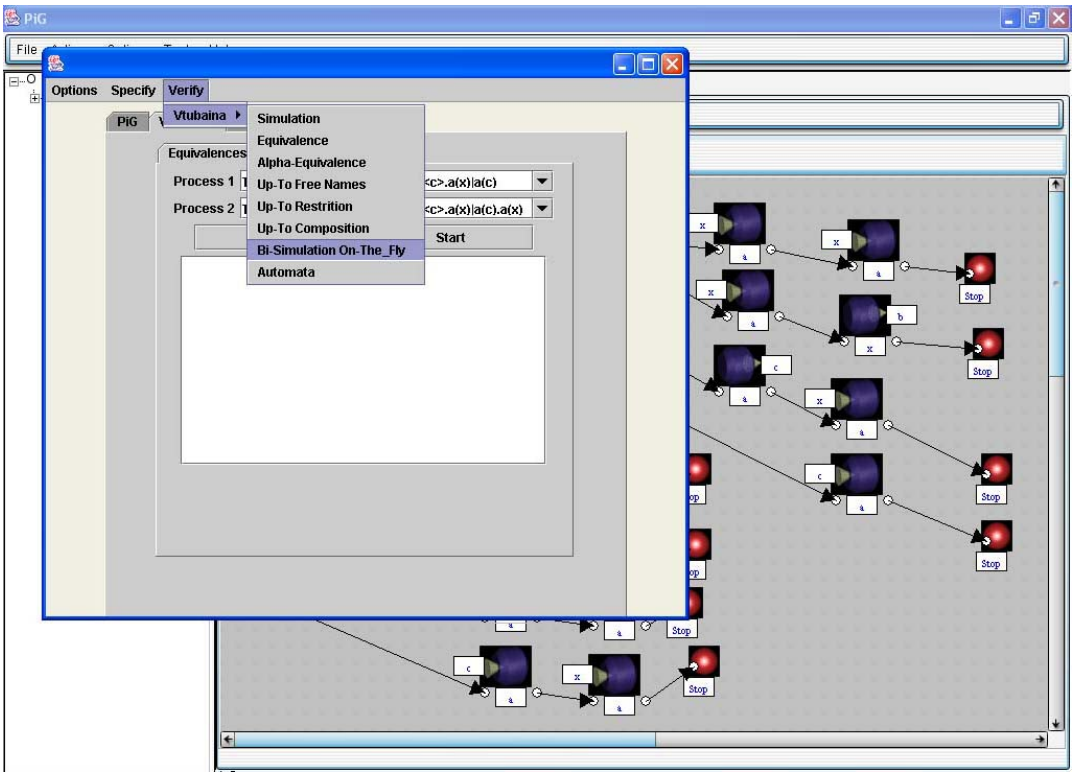

Figura 6: Escolha de Tarefas de Verificação

Após a especificação, exportamos, do módulo PIG, os arquivos `proc_t1.tuba` e `proc_t2.tuba` correspondente à especificação dos processos T_1 e T_2 a serem então carregadas pelo protótipo VTUBAINA. A Figura 6 ilustra os principais serviços fornecidos pelo ambiente para a verificação de agentes móveis.

[3] A nova técnica de verificação está fora do escopo do presente trabalho, detalhes podem ser vistos em [1].

Após especificar os agentes, quando solicitamos a verificação *OnTheFly*, importamos a árvore sintática da PiG, oferecendo os agentes já definidos como opções para a verificação. Após a escolha dos processos, um arquivo no formato aceito pela VTubaina é gerado, contendo os agentes e os comandos solicitados. A VTubaina é então invocada com este arquivo como parâmetro, executando assim a verificação.

No trecho abaixo, temos as linhas de comandos executadas pelo módulo VTubaina, com suas respectivas mensagens.

```
VTUBAINA> loadspec proc_t1.tuba
Agent T1 = (^#0)(#0(#3).#0(#4) | #0(#2).#2<b> | #0<c>.#0(#1) | #0<c>)
Free Names  T1: c,b
Bound Names T1: a,x,x,x,x
Agent T1 was built successfully!

VTUBAINA> loadspec proc_t2.tuba
Agent T2 = (^#0)(#0(#4).#0(#5) | #0(#3).#3<b> | #0<c>.#0(#2) | #0<c>.#0(#1))
Free Names  T2: c,b
Bound Names T2: a,x,x,x,x,x
Agent T2 was built successfully!

VTUBAINA> onthefly T1 T2
Agents <T1> and <T2> are not strong bisimilar!

VTUBAINA> createautomata aut1 T1
Active Names:  c, b
Automata aut1 was built successfully!

VTUBAINA> printautdot aut1 teste1out1
Automata [aut1]: File teste1out1.dot was created successfully!

VTUBAINA> createautomata aut2 T2
Active Names:  c, b
Automata aut2 was built successfully!

VTUBAINA> printautdot aut2 teste1out2
Automata [aut2]: File teste1out2.dot was created successfully!
```

A Figura 7 mostra os autômatos reduzidos gerados pela VTubaina para os agentes T_1 e T_2 do exemplo acima. Para ambos os agentes, o autômato da direita foi gerado com o uso das buscas sintáticas durante os desdobramentos (a nova técnica sugerida), enquanto o da esquerda foi gerado com a aplicação da técnica de particionamento de Montanari e Pistore. Vale salientar que a solicitação da verificação "onthefly" gera apenas os autômatos da direita. Os da esquerda são gerados apenas quando a opção "equivalence" é solicitada (a técnica de particionamento).

Além da verificação de equivalências entre agentes, podemos também gerar a simulação das transições dos processos usando a ferramenta. Neste caso, a simulação é assistida, na qual o usuário designa as informações das ações de entrada dos agentes.

4 Conclusão e Trabalhos Futuros

Neste artigo, apresentamos um ambiente para especificação e verificação que pode ser utilizado como ferramenta para auxiliar a manipulação de agentes móveis modelados utilizando o cálculo de processos π-calculus. Este cálculo foi primordialmente proposto como teoria base para agentes móveis, e não como linguagem de especificação. Apesar de ser um abuso de notação, existe um grande uso da teoria na especificação como forma de ilustrar a verificação de agentes. Desta forma, o ambiente proposto se presta mais para fins didáticos, com o objetivo de ilustrar a especificação e verificação de agentes móveis, do que para fins industriais. Para tais objetivos, linguagens de especificação baseadas tanto em π-calculus quanto em outras teorias têm sido propostas.

A construção do protótipos nos trabalhos de pesquisa científica constituí uma parte fundamental durante um estudo. Geralmente, protótipos são criados para obtenção de novos resultados e melhor visualização da automação do processo de verificação. Isso tem ocasionado na não reutilização de grande parte dos códigos encontrados nos vários trabalhos de pesquisa realizados em π-calculus por diferentes cientistas. Neste caso, tornaria-se interessante uma biblioteca contendo um ambiente de manipulação sintática e de simulação de processos descritos em π-calculus no intuito de ajudar a confecção de novos protótipos para estudos nessa área.

Inicialmente, o ambiente aqui proposto utiliza duas ferramentas, entretanto a junção de novos verificadores na construção de um único ambiente é de vital importância. Temos como objetivo acoplar novos verificadores a este ambiente gráfico de forma que o usuário possa editar seus agentes em um único ambiente, e ao mesmo tempo, utilizar as facilidades dos vários verificadores.

Como trabalho futuro também temos o desenvolvimento de novos componentes para a PiG, capazes de exibir e editar processos através de outros tipos de representações gráficas, ampliando a comunidade de usuários e a versatilidade da ferramenta.

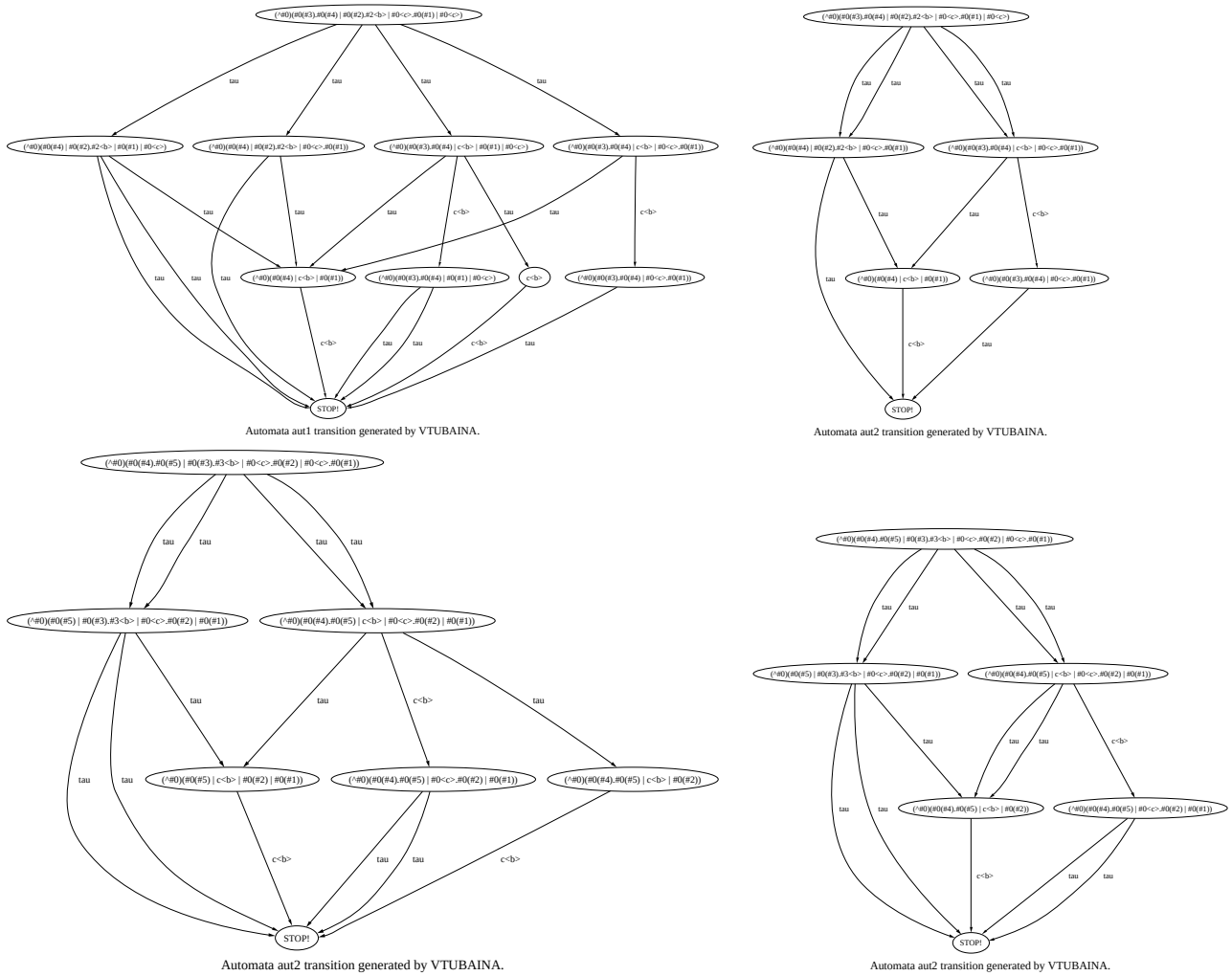

Figura 7: Autômatos para T_1 e T_2 construídos pela VTubaina

Outra possível aplicação seria o desenvolvimento de componentes capazes de integrar a PiG e o próprio Pi-Calculus com o processo de verificação de software adotado nas empresas que fazem uso de verificação formal. Este tipo de medida permitiria que o ambiente deixasse de ter apenas fins didáticos para ter uma real aplicação industrial.

Referências

[1] M. M. Amorim. Uma técnica de verificação para π-calculus baseada em bi-simulação up-to e algoritmos de particonamento. Master's thesis, IME/USP - Instituto de Matemática e Estatística da Universidade de São Paulo, 2003.

[2] J-C. Fernandez and L. Mounier. "on the fly" verification of behavioural equivalence and preorders. In *Proc. 3rd International Computer Aided Verification Conference*, pages 181–191, 1991.

[3] G. Ferrari, S. Gnesi, U. Montanari, M. Pistore, and G. Ristori. Verifying mobile processes in the HAL environment. In Alan J. Hu and Moshe Y. Vardi, editors, *Proceedings of CAV '98*, volume 1427 of *LNCS*. Springer, 1998. Tool Poster.

[4] G. Ferrari, G. Modoni, and P. Quaglia. Towards a Semantics-Based Environment for the π-Calculus. In *Proceedings of 5th Italian Conference on Theoretical Computer Science (ICTCS-95)*. World Scientific, 1995.

[5] C. Fournet and G. Gonthier. The reflexive chemical abstract machine and the join-calculus. In *Proceedings of POPL '96*, pages 372–385. ACM, January 1996.

[6] Andrade. Andre G. Pig - pi calculus grafico. *Atas do XVII Colóqui de Iniciação Científica*, pages 1–8, 2003.

[7] S. Gnesi. A formal verification environment for concurrent systems design. In *Proceedings Workshop on Automated Formal Methods*, volume 5 of *ENTCS*. University of Oxford, 1996.

[8] D. Hirschkoff. Automatically proving up to bisimulation. In Petr Jancar and Mojmir Kretinsky, editors, *Proceedings of MFCS '98 Workshop on Concurrency*, volume 18 of *ENTCS*. Elsevier Science Publishers, 1998.

[9] P. C. Kanellakis and S. A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86(1):43–68, May 1990.

[10] E. Koutsofios. Editing graphs with *dotty*. Technical report, AT&T Bell Laboratories, Murray Hill, NJ, USA, July 1994. This report, and the program, is included in the `graphviz` package, available for non-commercial use at `http://www.research.att.com/sw/tools/graphviz/.`

[11] E. Koutsofios and S. North. Drawing graphs with *dot*. Technical Report 910904-59113-08TM, AT&T Bell Laboratories, Murray Hill, NJ, USA, September 1991.

[12] R. Milner. *Communication and Concurrency*. Prentice-Hall, Englewood Cliffs, New Jersey, 1989.

[13] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, part I/II. *Journal of Information and Computation*, 100:1–77, September 1992.

[14] U. Montanari and M. Pistore. Checking bisimilarity for finitary π-calculus. In Insup Lee and Scott A. Smolka, editors, *Proceedings of CONCUR '95*, volume 962 of *LNCS*, pages 42–56. Springer, 1995.

[15] F. Orava and J. Parrow. An algebraic verification of a mobile network. *Journal of Formal Aspects of Computing*, 4:497–543, 1992.

[16] R. Paige and R. E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, December 1987.

[17] J. Parrow. An introduction to the π-calculus. In Bergstra, Ponse, and Smolka, editors, *Handbook of Process Algebra*, pages 479–543. Elsevier, 2001.

[18] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis, LFCS, University of Edinburgh, 1993. CST-99-93 (also published as ECS-LFCS-93-266).

[19] D. Sangiorgi. On the bisimulation proof method. *Mathematical Structures in Computer Science*, 8(5):447–479, 1998. An extended abstract appeared in the *Proceedings of MFCS '95*, LNCS 969: 479–488.

[20] D. Sangiorgi and R. Milner. The problem of "weak bisimulation up to". In W. R. Cleaveland, editor, *CONCUR '92: Third International Conference on Concurrency Theory*, volume 630 of *Lecture Notes in Computer Science*, pages 32–46, Stony Brook, New York, 24–27August 1992. Springer-Verlag.

[21] B. Thomsen. A theory of higher order communicating systems. *Journal of Information and Computation*, 116(1):38–57, 1995.

[22] B. Victor. *A Verification Tool for the Polyadic π-Calculus*. Licentiate thesis, Department of Computer Systems, Uppsala University, Sweden, May 1994. Available as report DoCS 94/50.

[23] D. Walker. Objects in the π-calculus. *Journal of Information and Computation*, 116(2):253–271, 1995.