# Usando CSP, RSL e o Modelo PopOrg na Especificação Formal de Organizações de SMAs [1]

Raquel de Miranda Barbosa [2]
Antônio Carlos da Rocha Costa [3]
Patrícia Cabral de Azevedo R. Tedesco [4]
Alexandre Cabral Mota [4]

**Resumo:** Este artigo explora o uso de métodos formais tradicionais de engenharia de software para a especificação formal de organizações de sistemas multiagentes. Particularmente são utilizadas as linguagens CSP (*Communicating Sequential Processes)*) e RSL (*Raise Specification Language*) para representar o modelo organizacional PopOrg, sendo a primeira utilizada para especificar partes do nível micro-organizacional de sistemas PopOrg (comportamentos de papéis organizacionais e processos de troca entre papéis organizacionais) e a segunda, RSL, utilizada para a representação da organização estrutural de sistemas PopOrg. O artigo apresenta alguns testes e resultados obtidos com o uso destes formalismos.

**Abstract:** This paper explores the use of traditional formal methods of software engineering for the formal specification of multiagent systems organizations. In particular, the CSP and RSL languages are used to represent the PopOrg organizational model, the first being used to specify parts of the micro-organizational level of PopOrg systems (behaviors of organizational roles and exchange processes between organizational roles) and the second, RSL, used to represent the structural organization of PopOrg systems. The paper presents some tests and results obtained using these formalisms.

## 1 Introdução

O uso da abordagem de sistemas multiagentes no desenvolvimento de sistemas complexos tem aumentado consideravelmente nos últimos anos. Isto ocorre devido às características desta abordagem e à adequabilidade das técnicas orientadas a agentes para o desenvolvimento de sistemas complexos [19].

[1] A versão preliminar deste artigo foi apresentada no WESAAC 2010, classificado em $2^o$ lugar, entre os melhores artigos do evento.

[2] Instituto de Informática, UFRGS, Caixa Postal 15064
miranda@inf.ufrgs.br
[3] Centro de Ciências Computacionais, FURG, Caixa Postal 474
ac.rocha.costa@gmail.com
[4] Centro de Informática, UFPE, Caixa Postal 7851
{pcart,acm}@cin.ufpe.br

Dentre as características fundamentais da noção de modelagem conceitual orientada a agentes está a noção de "autonomia dos agentes" [25]. A presença de autonomia como característica fundamental dos componentes de um sistema, no entanto, adiciona uma complexidade considerável às questões de modelagem do sistema, quando comparada à modelagem como sistema de objetos.

Se na modelagem de sistemas orientada a objetos o uso de métodos formais bem estabelecidos é tido como, no mínimo, conveniente para o projetista do sistema, em termos de prevenção de erros e execução de propriedades desejadas no sistema [1], parece-nos que no caso da modelagem orientada a agentes o uso de métodos formais bem estabelecidos torna-se não menos obrigatório, devido à complexidade das questões envolvidas.

Diversas metodologias de desenvolvimento de software têm sido propostas para dar suporte à adoção da abordagem orientada a agentes, assim definindo uma área específica da engenharia de software, conhecida como Engenharia de Software Orientada a Agentes [18]. Dentre estas metodologias estão GAIA [26], O-MaSE [6], MAS-CommonKADS [17], MessageUML [2], Tropos [12] e Prometheus [24].

Muitas das metodologias AOSE existentes atualmente surgem como modificações de metodologias orientadas a objetos já existentes, e existem ainda muitas lacunas no que se refere a aspectos particulares de sistemas multiagentes. Poucas metodologias utilizam métodos formais em seu desenvolvimento, e a especificação formal de organizações de sistemas multiagentes não foi, ainda, completamente definida como área de pesquisa na área de Engenharia de Software.

Um importante aspecto nesta questão está relacionado aos aspectos sociais dos sistemas de agentes. Eles referem-se aos relacionamentos entre agentes e são importantes devido às questões centrais de agentes, como cooperação, competição, negociação, etc. Em particular, a organização social de um sistema deve ser vista como um aspecto central de um modelo orientado a agentes, porque ela define o conjunto de papéis que agentes podem desempenhar no sistema, o conjunto de possíveis relacionamentos que estes papéis podem ter, bem como características reguladoras do sistema, como normas, compromissos e acordos.

Na área de sistemas multiagentes, existem diversos estudos preocupados com a modelagem de organizações de sistemas de agentes, tais como MOISE$^+$ [16], AGR [10], ISLANDER [9], OPERA [8] and PopOrg [20].

Neste artigo, exploramos a ideia de usar métodos formais padrões de Engenharia de Software para especificar a organização de sistemas multiagentes. Em particular, mostramos como CSP [15] [21] pode ser relacionado semanticamente ao modelo PopOrg, e verificamos algumas propriedades de parte da especificação micro-organizacional de um sistema exemplo, usando FDR2 (*Failures-Divergences Refinement*) [11].

Desta forma, usamos CSP para a especificação formal de alguns aspectos operacionais (comportamentos de papéis organizacionais e processos de troca entre papéis organizacionais) do nível micro-organizacional de sistemas que podem ser modelados através do modelo semântico PopOrg [3]. Adicionalmente, utilizamos a linguagem RSL para representar os elementos estruturais do modelo PopOrg.

O artigo está estruturado da seguinte forma. A seção 2 apresenta a linguagem CSP. Na seção 3 o modelo PopOrg é resumido, destacando-se seu nível micro-organizacional. A seção 4 define a conexão semântica entre traços CSP e comportamentos e processos de troca PopOrg. A seção 5 apresenta um estudo de caso usando CSP para a especificação formal de aspectos operacionais do modelo micro-organizacional de um sistema PopOrg. Na seção 6 apresentamos o método RAISE e a linguagem RSL, juntamente com a especifcação dos aspectos estruturais do modelo PopOrg. Conclusões e trabalhos futuros são apresentados na seção 7.

## 2 A linguagem CSP

CSP *(Communicating Sequential Processes)* é uma notação para a descrição de sistemas concorrentes nos quais os processos componentes interagem através de comunicação [21].

O modelo conceitual usado por CSP considera componentes, ou processos, como entidades independentes (autônomas) com interfaces particulares através das quais eles interagem com seu ambiente. A interface de um processo é descrita como um conjunto de eventos, cada um deles descrevendo um tipo particular de ação atômica que pode ser executada ou sofrida pelo processo. Esta interface pode ser relacionada à especificação estática de um processo, enquanto sua especificação dinâmica descreve como ele irá se comportar nesta interface.

Como linguagem formal, CSP pode ser entendida através das semânticas: operacional, denotacional e algébrica [21]. No estilo denotacional, CSP pode ser interpretada em três níveis de detalhes: *traces*, falhas e falhas-divergências. Neste artigo fazemos uso do modelo de *traces* (que mostra a história dos eventos de cada processo em um dado sistema) para relacionar CSP com as estruturas organizacionais de sistemas multiagentes baseados em PopOrg.

## 3 O modelo PopOrg

O modelo organizacional PopOrg de sistemas baseados em agentes foi proposto em [7] como uma base semântica para modelos formais de sistemas multiagentes com organizações dinâmicas. Sua proposta é capturar separadamente os dois aspectos de sistemas de agentes:

suas populações e suas organizações.

A população de um sistema baseado em agentes consiste do conjunto de agentes que nele habitam, juntamente com o conjunto de todos os comportamentos que estes agentes são capazes de realizar e o conjunto de processos de troca que eles podem ter. A organização de um sistema baseado em agentes é uma estrutura composta pelos papéis organizacionais e links organizacionais, onde um papel organizacional é definido em relação ao conjunto de processos organizacionais nos quais o agente está envolvido, e os links organizacionais entre um subconjunto de agentes são os processos que estes agentes executam dentro de processos organizacionais mais abrangentes.

O modelo PopOrg é baseado na distinção entre as noções de descrições intensionais e extencionais de sistemas [20]. Descrições intensionais estão relacionadas a aspectos subjetivos, pertencendo ao funcionamento interno dos agentes que operam no sistema modelado (como normas, valores, etc.), enquanto descrições extencionais referem-se a aspectos objetivos, pertencendo ao funcionamento externo dos agentes (como ações executadas, objetos trocados, etc.). O modelo PopOrg concentra-se na representação de aspectos externos do sistema, e considera os aspectos intensionais como estrutura adicional, opcional, que é superimposta na extencional. Desta forma, este modelo é considerado um modelo mínimo de organização de sistemas multiagentes, pois ele representa apenas os componentes-chave de uma organização, permitindo que outros modelos mais complexos possam ser representados através dele.

Em [3], este modelo foi estendido de forma a contemplar a divisão entre os níveis micro e macro-organizacional de um sistema (cf. Figura 1). O nível micro-organizacional é onde ocorrem as interações organizacionais entre papéis individualizados, enquanto o nível macro-organizacional, é o nível onde unidades organizacionais (grupos de papéis) são introduzidas para permitir a estruturação da organização hierarquicamente, com as interações acontecendo entre estas unidades organizacionais. Um par de relações de implementação define a forma na qual as unidades organizacionais são implementadas por conjuntos de papéis organizacionais e a forma na qual papéis organizacionais são implementados por agentes individuais da população.

O modelo PopOrg com as estruturas micro e macro-organizacional (estrutura-$\omega\Omega$) é uma estrutura $POPORG = (POP, ORG, IMP)$ onde $POP$ é a estrutura populacional, $ORG_{\omega\Omega} = (ORG_\omega, ORG_\Omega)$ é a estrutura organizacional-$\omega\Omega$, sendo $ORG_\omega$ o nível micro-organizacional e $ORG_\Omega$ o nível macro-organizacional, e $IMP_{\omega\Omega} = (IMP_\omega, IMP_\Omega)$ é a relação de implementação de $ORG_{\omega\Omega}$ sobre $POP$ [4].

Seja $Bh$ o universo dos comportamentos de papéis possíveis no modelo, e $Ep$ o universo de todos os possíveis processos de trocas entre papéis, a estrutura micro-organizacional de uma estrutura populacional POP é a estrutura $ORG_\omega = (R_\omega, L_\omega, lc_\omega)$, onde:

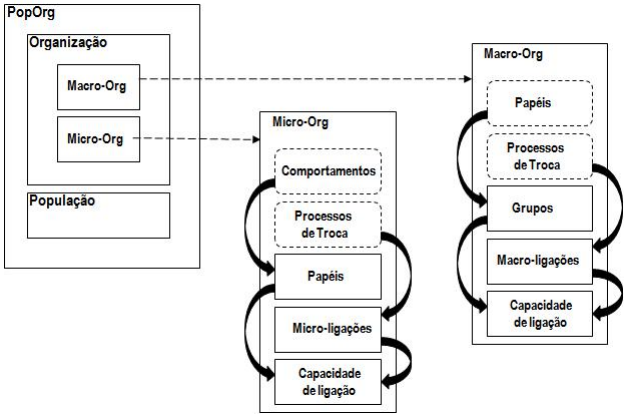


**Figura 1.** PopOrg Model

- $R_\omega \subseteq \wp(Bh)$ é o conjunto dos papéis existentes na organização, onde um papel consiste de um conjunto de comportamentos que um agente que desempenha o papel pode executar.

- $L_\omega \subseteq R_\omega \times R_\omega \times Ep$ é o conjunto das ligações (*links*) que existem na organização entre pares de papéis, cada ligação especificando um processo de troca que os agentes que desempenham os papéis ligados podem realizar.

- $lc_\omega : R_\omega \times R_\omega \rightarrow \wp(L_\omega)$ é a capacidade de ligação de pares de papéis, ou seja, o conjunto de ligações que os pares de papéis podem estabelecer entre eles.

A seguir, explicamos como CSP pode ser usado para a especificação dos conjuntos $R_\omega$ e $L_\omega$ de sistemas multiagentes baseados em PopOrg.

## 4 A relação entre traces CSP e Comportamentos PopOrg

Para fazer uso significativo da linguagem CSP como formalismo para a especificação de papéis e processos de troca da estrutura micro-organizacional de modelos PopOrg, deve-se ter certeza de que as propriedades dos programas CSP escritas como uma especificação organizacional são preservadas quando transformadas em implementações PopOrg.

Na teoria de CSP [21], a preservação de propriedades entre especificações e implementações é garantida através da relação de refinamento entre os dois modelos semânticos

envolvidos: é garantido que o modelo mais refinado herda propriedades do modelo que ele refina.

Neste artigo é utilizado o modelo semântico básico de CSP, modelo de *traces*, observando-se como é possível construir o modelo comportamental PopOrg como refinamento do modelo de *traces* de CSP. Isto é suficiente para estabelecer que qualquer propriedade de segurança (*safety*) provada verdadeira na especificação CSP é também verdadeira para qualquer implementação PopOrg daquela especificação.

A Figura 2 apresenta a relação entre a linguagem CSP e o modelo PopOrg. As setas curvas mostram que papéis e ligações são compostos de comportamentos e processos de troca e estão incluídos nas capacidades de ligações (conforme formalmente definido em [4]). As demais setas representam a relação semântica entre traces CSP e comportamentos PopOrg, formalizada na seção 4.3.

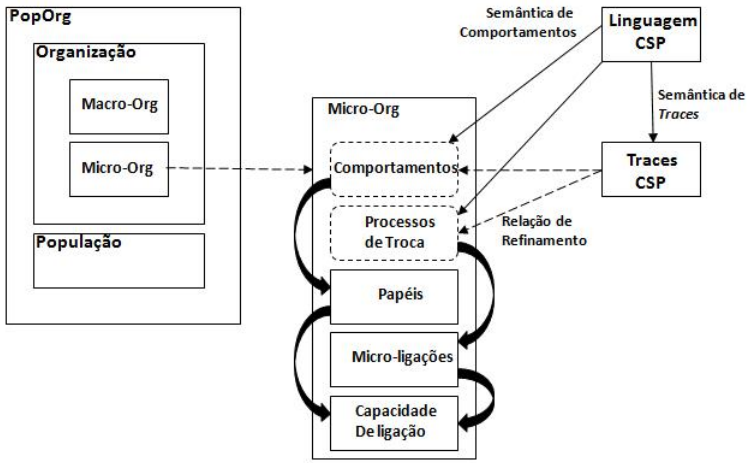


**Figura 2.** Conexão entre CSP e PopOrg

### 4.1 Comportamentos PopOrg

Assuma que T seja uma estrutura de tempo linear discreta e A, o universo finito de ações a partir das quais são definidos os comportamentos organizacionais.

Um comportamento de um papel organizacional PopOrg é uma sequência indexada no tempo de subconjuntos de ações $b : T \rightarrow \wp(A)$ da forma $b = \{0 \rightarrow \alpha, 1 \rightarrow \beta, ...\}$, cada subconjunto $\alpha, \beta, ... \in \wp(A)$ indicando um possível conjunto de ações que o papel

organizacional que executa o comportamento b pode realizar no tempo $t \in T$. O conjunto de todos os comportamentos organizacionais é denotado por $Bh = [T \rightarrow \wp(A)]$.

Um processo de troca PopOrg, entre dois papéis organizacionais, é uma sequência, indexada no tempo, de pares de subconjuntos de ações, $e : T \rightarrow \wp(A) \times \wp(A)$, cada par de subconjuntos $e(t) \in \wp(A) \times \wp(A)$ indicando o conjunto de ações que cada papel organizacional pode executar no tempo t, quando eles estão interagindo. O conjunto de todos os processos de troca organizacionais é denotado por $Ep = [T \rightarrow \wp(A) \times \wp(A)]$.

Assim, por exemplo, assumindo que $A = \{a, b, c, ...\}$ e que $T = (0, 1, 2, ...)$, temos que $b = \{0 \rightarrow \{a\}, 1 \rightarrow \emptyset, 2 \rightarrow \{b, c\}, 3 \rightarrow \{b\}, 4 \rightarrow \emptyset, 5 \rightarrow \{a, c\}, ...\}$ pode ser um comportamento PopOrg que escolhe aleatoriamente executar no instante zero, uma ou duas ações obtidas do subconjunto {a,b,c} e que $e = \{0 \rightarrow (\{a\}, \emptyset), 1 \rightarrow (\emptyset, \{b, c\}), 2 \rightarrow (\{a\}, \emptyset), 3 \rightarrow (\emptyset, \{b, c\}), ...\}$ pode ser um processo de troca onde os dois papéis organizacionais envolvidos alternam suas ações, o primeiro papel sempre executando a ação $a$ quando ele age e o segundo executando simultaneamente as ações $b$ e $c$ quando age.

### 4.2 O subconjunto de CSP usado para especificar comportamentos organizacionais

Para esta análise foi considerado apenas um subconjunto dos construtores da linguagem CSP [15] [21]. O único programa primitivo é o programa de *deadlock* STOP. Para introduzir sequencialidade em programas CSP, usamos o construtor de prefixo $\rightarrow$. Para introduzir não-determinismo e concorrência entre dois ou mais processos, utilizamos os operadores de escolha externa ($\Box$) e entrelaçamento ($|||$).

No modelo PopOrg, os comportamentos organizacionais são sequências de conjuntos de ações que papéis organizacionais podem realizar. Desta forma, especificações CSP devem ter o alfabeto $\sum = \wp(A)$. Nós fazemos uso do construtor de prefixo na forma $\alpha \rightarrow P$, com $\alpha \in \wp(A)$. Entretanto, por conveniência, permitimos que $\{a\} \rightarrow P$ seja denotado por $a \rightarrow P$.

### 4.3 A semântica de comportamentos de CSP

A semântica PopOrg de CSP é uma semântica de *timed traces*. Entretanto, ela difere da semântica de *timed traces* de *Timed* CSP [5] , pois *timed traces* PopOrg são completos, no sentido em que eles denotam explicitamente todos os conjuntos de ações que são possíveis em cada instante de tempo, enquanto os *traces* de *Timed* CSP apenas denotam explicitamente os tempos em que as ações ocorreram.

Por outro lado, quando definimos o conjunto de comportamentos PopOrg que correspondem a um programa CSP, sabemos que o sequenciamento abstrato de ações prescritas pelo programa CSP não é traduzido em um sequenciamento temporal rígido. Ou seja, nós ga-

rantimos que um número arbitrário de eventos ociosos pode ser inserido entre quaisquer dois eventos concretos, tal que qualquer expressão seja interpretada como um conjunto infinito de comportamentos PopOrg.

Desta forma, a semântica dos programas CSP que modelam comportamentos PopOrg é dada pela função $bhs^t : CSP \mapsto \wp(Bh)$, que modela um programa CSP como um conjunto de comportamentos organizacionais PopOrg, partindo do tempo 0.

Para definir $bhs^t$, são adotadas as seguintes notações:

- para qualquer comportamento $b = \{t \mapsto \alpha, t+1 \mapsto \beta, ...\}$, o deslocamento de t por k unidades de tempo é definido por:
  $b^k = \{t+k \mapsto \alpha, t+k+1 \mapsto \beta, t+k+2 \mapsto \gamma, ...\}$, para todo k tal que $t + k \geq 0$.
- para qualquer evento comportamental $\{t \mapsto \alpha\}$, a exponenciação $\{t \mapsto \alpha\}^n$ do evento é dada por:
  $\{t \mapsto \alpha\}^0 = \emptyset$
  $\{t \mapsto \alpha\}^n = \{t \mapsto \alpha\} \cup \{t+1 \mapsto \alpha\} \cup ... \cup \{t+n-1 \mapsto \alpha\} =$
  $\{t \mapsto \alpha, t+1 \mapsto \alpha, ..., t+n \mapsto \alpha\}$, se $n \geq 1$
- para quaisquer comportamentos $b_1$ e $b_2$, iniciando no tempo t, seu entrelaçamento $b_1 \ ||| \ b_2$ é definido por:
  $\langle\rangle \ ||| \ b_2 = b_2$
  $b_1 \ ||| \ \langle\rangle = b_1$
  $\{t \mapsto \alpha\} \cup b'_1 \ ||| \ \{t \mapsto \beta\} \cup b'_2 =$
  $\{\{t \mapsto \alpha\} \cup b | b \in (b'^{(+1)}_1 \ ||| \ \{t+1 \mapsto \beta\} \cup b^{(+2)}_2)\} \cup$
  $\{\{t \mapsto \beta\} \cup b | b \in (\{t+1 \mapsto \alpha\} \cup b^{(+2)}_1 \ ||| \ b'^{(+1)}_2)\} \cup$
  $\{\{t \mapsto \alpha \cup \beta\} \cup b | b \in (b'^{(+1)}_1 \ ||| \ b'^{(+1)}_2)\}$

A função semântica $bhs^t$ é definida por:

$$bhs^t(STOP) = \{\{t \mapsto \emptyset, t+1 \mapsto \emptyset, t+2 \mapsto \emptyset, ...\}\}$$

$$bhs^t(\alpha \to P) = \{\{t \mapsto \emptyset\}^n \cup \{t+n \mapsto \alpha\} \cup b | b \in bhs^{t+n+1}(P), n \geq 0\}$$

$$bhs^t(\alpha \to P \ \Box \ \beta \to Q) = \{\{t \mapsto \emptyset\}^n \cup \{t+n \mapsto \alpha\} \cup b | b \in bhs^{t+n}(P \ \Box \ \beta \to Q), n \geq 0\} \cup \{\{t \mapsto \emptyset\}^n \cup \{t+n \mapsto \beta\} \cup b | b \in bhs^{t+n}(\alpha \to P \ \Box \ Q), n \geq 0\}$$

$$bhs^t(P \ ||| \ Q) = \bigcup \{b_P \ ||| \ b_Q | b_P \in bhs^t(P), b_Q \in bhs^t(Q)\}$$

Denotamos $bhs^0(P)$ simplesmente por $bhs(P)$. Alguns exemplos são apresentados a seguir:

$bhs(a \rightarrow STOP) = \{$
$\{0 \mapsto \{a\}, 1 \mapsto \emptyset, 2 \mapsto \emptyset, ...\},$
$\{0 \mapsto \emptyset, 1 \mapsto \{a\}, 2 \mapsto \emptyset, 3 \mapsto \emptyset, ...\},$
$\{0 \mapsto \emptyset, 1 \mapsto \emptyset, 2 \mapsto \{a\}, 3 \mapsto \emptyset, 4 \mapsto \emptyset, ...\},$
$...$
$\}$

$bhs(\{a, b\} \rightarrow c \rightarrow STOP) = \{$
$\{0 \mapsto \{a, b\}, 1 \mapsto \{c\}, 2 \mapsto \emptyset, 3 \mapsto \emptyset, ...\},$
$\{0 \mapsto \{a, b\}, 1 \mapsto \emptyset, 2 \mapsto \{c\}, 3 \mapsto \emptyset, 4 \mapsto \emptyset, ...\},$
$\{0 \mapsto \{a, b\}, 1 \mapsto \emptyset, 2 \mapsto \emptyset, 3 \mapsto \{c\}, 4 \mapsto \emptyset, 5 \mapsto \emptyset...\},$
$...,$
$\{0 \mapsto \emptyset, 1 \mapsto \{a, b\}, 2 \mapsto \{c\}, 3 \mapsto \emptyset, 4 \mapsto \emptyset, ...\},$
$\{0 \mapsto \emptyset, 1 \mapsto \{a, b\}, 2 \mapsto \emptyset, 3 \mapsto \{c\}, 4 \mapsto \emptyset, 5 \mapsto \emptyset, ...\},$
$...,$
$\{0 \mapsto \emptyset, 1 \mapsto \emptyset, 2 \mapsto \{a, b\}, 3 \mapsto \{c\}, 4 \mapsto \emptyset, 5 \mapsto \emptyset...\},$
$\{0 \mapsto \emptyset, 1 \mapsto \emptyset, 2 \mapsto \{a, b\}, 3 \mapsto \emptyset, 4 \mapsto \{c\}, 5 \mapsto \emptyset, ...\},$
$...,$
$\}$

$bhs(a \rightarrow STOP \,\Box\, b \rightarrow STOP) = \{$
$\{0 \mapsto \{a\}, 1 \mapsto \emptyset, 2 \mapsto \emptyset, ...\},$
$\{0 \mapsto \{b\}, 1 \mapsto \emptyset, 2 \mapsto \emptyset, ...\},$
$\}$

$bhs(a \rightarrow STOP \,|||\, b \rightarrow STOP) = \{$
$\{0 \mapsto \{a\}, 1 \mapsto \emptyset, 2 \mapsto \emptyset, ...\},$
$\{0 \mapsto \{b\}, 1 \mapsto \emptyset, 2 \mapsto \emptyset, ...\},$
$\{0 \mapsto \{a\}, 1 \mapsto \{b\}, 2 \mapsto \emptyset, ...\},$
$\{0 \mapsto \{b\}, 1 \mapsto \{a\}, 2 \mapsto \emptyset, ...\},$
$\{0 \mapsto \{a\}, 1 \mapsto \emptyset, 2 \mapsto \{b\}, ...\},$
$\{0 \mapsto \{b\}, 1 \mapsto \emptyset, 2 \mapsto \{a\}, ...\},$
$...,$
$\{0 \mapsto \emptyset, 1 \mapsto \{a\}, 2 \mapsto \emptyset, ...\},$
$\{0 \mapsto \emptyset, 1 \mapsto \{b\}, 2 \mapsto \emptyset, ...\},$
$\{0 \mapsto \emptyset, 1 \mapsto \{a\}, 2 \mapsto \{b\}, ...\},$
$\{0 \mapsto \emptyset, 1 \mapsto \{b\}, 2 \mapsto \{a\}, ...\},$
$\{0 \mapsto \emptyset, 1 \mapsto \{a\}, 2 \mapsto \emptyset, 3 \mapsto \{b\}, ...\}],$
$\{0 \mapsto \emptyset, 1 \mapsto \{b\}, 2 \mapsto \emptyset, 3 \mapsto \{a\}, ...\},$
$...,$
$\}$

Fazemos uso da notação [.] para o conjunto de todas as diferentes associações temporais de um dado comportamento, tal que $bhs(\{a, b\} \rightarrow c \rightarrow STOP)$ é denotado simplesmente por:

$$bhs(\{a, b\} \rightarrow c \rightarrow STOP) = \\ \quad \{[0 \mapsto \{a, b\}, 1 \mapsto \{c\}]\}$$

E, sendo $P = a \rightarrow P$ e $Q = b \rightarrow Q$, temos:

$$bhs(P) = \{ \\ \quad [0 \mapsto \{a\}, 1 \mapsto \{a\}, 2 \mapsto \{a\}, ...], \\ \quad \}$$

$$bhs(Q) = \{ \\ \quad [0 \mapsto \{b\}, 1 \mapsto \{b\}, 2 \mapsto \{b\}, ...], \\ \quad \}$$

$$bhs(P\,;Q) = \{ \\ \quad [0 \mapsto \{a\}, 1 \mapsto \{a\}, 2 \mapsto \{a\}, ...], \\ \quad \}$$

$$bhs(P \,\square\, Q) = \{ \\ \quad [0 \mapsto \{a\}, 1 \mapsto \{a\}, 2 \mapsto \{a\}, ...], \\ \quad [0 \mapsto \{b\}, 1 \mapsto \{b\}, 2 \mapsto \{b\}, ...], \\ \quad \}$$

$$bhs(P \,|||\, Q) = \{ \\ \quad [0 \mapsto \{a\}, 1 \mapsto \{a\}, 2 \mapsto \{a\}, ...], \\ \quad [0 \mapsto \{b\}, 1 \mapsto \{a\}, 2 \mapsto \{a\}, ...], \\ \quad [0 \mapsto \{a, b\}, 1 \mapsto \{a\}, 2 \mapsto \{a\}, ...], \\ \quad [0 \mapsto \{a\}, 1 \mapsto \{b\}, 2 \mapsto \{a\}, ...], \\ \quad [0 \mapsto \{a\}, 1 \mapsto \{a, b\}, 2 \mapsto \{a\}, ...], \\ \quad [0 \mapsto \{b\}, 1 \mapsto \{b\}, 2 \mapsto \{a\}, ...], \\ \quad [0 \mapsto \{a, b\}, 1 \mapsto \{a, b\}, 2 \mapsto \{a\}, ...], \\ \quad ..., \\ \quad [0 \mapsto \{b\}, 1 \mapsto \{b\}, 2 \mapsto \{b\}, ...], \\ \quad [0 \mapsto \{a\}, 1 \mapsto \{b\}, 2 \mapsto \{b\}, ...], \\ \quad [0 \mapsto \{a, b\}, 1 \mapsto \{b\}, 2 \mapsto \{b\}, ...], \\ \quad [0 \mapsto \{b\}, 1 \mapsto \{a\}, 2 \mapsto \{b\}, ...], \\ \quad [0 \mapsto \{b\}, 1 \mapsto \{a, b\}, 2 \mapsto \{b\}, ...], \\ \quad [0 \mapsto \{a\}, 1 \mapsto \{a\}, 2 \mapsto \{b\}, ...], \\ \quad [0 \mapsto \{a, b\}, 1 \mapsto \{a, b\}, 2 \mapsto \{b\}, ...],$$

...,
$[0 \mapsto \{a, b\}, 1 \mapsto \{a, b\}, 2 \mapsto \{a, b\}, ...],$
$\}$

Claramente, para cada programa CSP existe um mapeamento do seu conjunto de comportamentos PopOrg para seu conjunto de *traces*, abstraindo-se a relação temporal dos eventos de comportamento: para cada comportamento do programa CSP, da forma $\{0 \mapsto \alpha_0, 1 \mapsto \alpha_1, 2 \mapsto \alpha_2, ...\}$ , existe um *trace* daquele programa, da forma $\langle \alpha_{i0}, \alpha_{i1}, \alpha_{i2}, ... \rangle$, tal que todos os eventos $\alpha_k$ no *trace* são não-vazios, $\alpha_k \neq \emptyset$, e o *trace* é tal que, para quaisquer dois de seus eventos $\alpha_j$ e $\alpha_k$, se k<j, então os eventos de comportamento $t_k \mapsto \alpha_k$ e $t_j \mapsto \alpha_j$ estão no comportamento e $t_k < t_j$.

Nota-se, então, que a semântica PopOrg de CSP dá uma semântica de programa que é um refinamento da semântica de *traces*. A relação de refinamento entre interpretações CSP tem a propriedade de permitir a interpretação mais refinada herdar as propriedades de programas provadas através da interpretação mais abstrata. Assim, todas as propriedades de programas CSP provadas usando a semântica de *traces*, são herdadas pela interpretação PopOrg daquele programa, mostrando que programas CSP podem ser usados para a especificação de partes operacionais (comportamentos de papéis e processos de troca) do modelo micro-organizacional PopOrg.

### 4.4 A semântica de Processos de Troca de CSP

Da mesma forma que os papéis foram definidos como um processo CSP, agora definimos os processos de troca.

Um processo de troca é uma sequência, indexada no tempo, de pares de subconjuntos de ações $e : T \rightarrow \wp(A) \times \wp(A)$, e o conjunto de processos de troca é denotado por $Ep$ (conforme definido na seção 4.1).

Um link especifica um conjunto de processos de troca que os agentes, que desempenham os papéis ligados pelo link, podem ter de realizar. Assim, como exemplo, um link é $l = (r_1, r_2, E)$, onde $E \subseteq Ep$, $E = \{e_1, e_2, \ldots\}$, e cada processo é representado como

$$e_1 = [0 \mapsto (\{a\}, \{b\}), 1 \mapsto (\{a\}, \{b, c\}), 2 \mapsto (\{b, c\}, \emptyset), 3 \mapsto (\{a\}, \{b\}), \ldots]$$

A semântica de um programa CSP que especifica um conjunto de processos de troca PopOrg é dada pela função $eps^t : CSP \rightarrow \wp(Ep)$, a qual modela o programa CSP como um conjunto de processos de troca, todos iniciando no tempo t. Os processos CSP que determinam os processos de troca são da forma $P = (\alpha, \beta) \rightarrow (\gamma, \delta) \rightarrow ...$, sendo o seu alfabeto representado por pares de conjuntos de ações $\Sigma \subseteq \wp(A) \times \wp(A)$, cada conjunto executado por um papel.

$eps^t(STOP) = \{[t \mapsto (\emptyset, \emptyset), t+1 \mapsto (\emptyset, \emptyset), t+2 \mapsto (\emptyset, \emptyset), ...]\}$
$eps^t(P) = \{[t \mapsto (\alpha, \beta), t+1 \mapsto (\gamma, \delta), ...]\}$

Denotamos $eps^0(P)$ simplesmente por $eps(P)$.

Por exemplo, seja $P = (\{a\}, \{b\}) \rightarrow (\{c\}, \{d\}) \rightarrow STOP$

Aplicando-se a função $eps$ ao processo $P$, obtemos o conjunto com as diferentes maneiras de distribuir os pares de ações $(\{a\}, \{b\})$ e $(\{c\}, \{d\})$ no tempo:

$$
\begin{aligned}
eps(P) = \{ & \\
& [0 \rightarrow (\{a\}, \{b\}), 1 \rightarrow (\{c\}, \{d\}), ...], \\
& [0 \rightarrow (\emptyset, \emptyset), 1 \rightarrow (\{a\}, \{b\}), 2 \rightarrow (\{c\}, \{d\}), ...], \\
& [0 \rightarrow (\{a\}, \{b\}), 1 \rightarrow (\emptyset, \emptyset), 2 \rightarrow (\{c\}, \{d\}), ...], \\
& [0 \rightarrow (\emptyset, \emptyset), 1 \rightarrow (\{a\}, \{b\}), 2 \rightarrow (\emptyset, \emptyset), 3 \rightarrow (\{c\}, \{d\}), ...], \\
& ... \\
& \} \\
& = \{[0 \rightarrow (\{a\}, \{b\}), 1 \rightarrow (\{c\}, \{d\}), ...]\}
\end{aligned}
$$

Para verificar a compatibilidade de um processo de troca com o par de papéis envolvidos, é necessário verificar se cada papel pode executar o conjunto de comportamentos que lhe corresponde no processo de troca. Para isto define-se duas funções de projeção $prj_1(e)$ e $prj_2(e)$, onde $e \in Ep$, que recuperam os comportamentos desejados para cada um dos papéis envolvidos em um processo de troca:

$$prj_1(e) = \{0 \mapsto \{a\}, 1 \mapsto \{a\}, 2 \mapsto \{b, c\}, 3 \mapsto \{a, b\}, \ldots\}$$

$$prj_2(e) = \{0 \mapsto \{b\}, 1 \mapsto \{b, c\}, 2 \mapsto \emptyset, 3 \mapsto \{b\}, \ldots\}$$

Desta forma podemos verificar a compatibilidade entre papéis e processos de troca, necessária para a formação do link $l = (r_1, r_2, E)$, analisando as restrições de compatibilidade a seguir:

$$\forall\, e \in E\, \forall\, t\, \exists\, b \in r_1 : prj_1(e)(t) \subseteq b(t)$$

$$\forall\, e \in E\, \forall\, t\, \exists\, b \in r_2 : prj_2(e)(t) \subseteq b(t)$$

Chamamos essa restrição de *compatibilidade fraca*, pois em cada instante pode existir um comportamento diferente em cada papel satisfazendo a restrição.

No nosso exemplo, considerando os papéis $r_1$ e $r_2$:

$$r_1 = \{[0 \mapsto \{a, d\}, 1 \mapsto \{a, e\}, 2 \mapsto \{a, b, c\}, 3 \mapsto \{a, d\}, \ldots], \ldots\}$$

$$r_2 = \{[0 \mapsto \{a, b\}, 1 \mapsto \{b, c, d\}, 2 \mapsto \{c\}, 3 \mapsto \{b, d\}, \ldots], \ldots\}$$

é possível verificar que os papéis satisfazem esta restrição e, portanto, formam o link $l = (r_1, r_2, P)$.

Uma restrição de *compatibilidade forte* exige que em cada papel exista um comportamento responsável por viabilizar, sozinho, todo o processo de troca, sendo definida como:

$$\forall e \in E \, \exists b \in r_1 \, \forall t : prj_1(e)(t) \subseteq b(t)$$

$$\forall e \in E \, \exists b \in r_2 \, \forall t : prj_2(e)(t) \subseteq b(t)$$

## 5 Um Estudo de Caso

Para o estudo prático da adequabilidade de utilização de CSP para PopOrg, foi analisado o jogo VTEAM [23], um jogo de simulação para treinamento de gerentes de software com a ajuda de atores sintéticos.

Neste jogo, o jogador assume o papel de gerente de projeto cujo objetivo é conquistar uma missão estabelecida no início do jogo. Além do jogador, existem outros dois tipos de personagens no VTEAM: os membros de equipe e o cliente. Membros de equipe são personagens autônomos, capazes de interagir com os outros a fim de executar as atividades atribuídas a eles pelo gerente de projeto. O cliente é responsável por aceitar as entregas finais produzidas pelo projeto, aprovando ou não os produtos.

Em nosso experimento, escolhemos uma cena particular do jogo e descrevemos em CSP as interações entre Gerente e Membros de Equipe em relação à alocação e realocação de atividades, reuniões e pedidos de ajuda. Para verificar as propriedades de comportamentos organizacionais do jogo, foi utilizada a linguagem $CSP_M$ e o verificador de modelos FDR2 [11].

O exemplo a seguir mostra a especificação CSP do papel Gerente.

```
Gerente(e,t) =
  t!={} & (|~| ag:ME @ (|~| at:t @
    (alocaAtividade.ag!at->Gerente(e,(diff(t,{at})))))))
  []
  (negaAtividade.ag1.at->|~| ag2:diff(ME,{ag1}) @
    realocaAtividade.ag1!at.ag2->
    desalocaAtividade.ag1!at->alocaAtividade.ag2!at->
    Gerente(e,t))
  []
  reuniaoAcompanhamento.n:{ME} -> Gerente(RA,t)
  []
  e == RA & (emReuniao->Gerente(e,t) []
    fimReuniaoA->Gerente(W,t))
  []
  (reportaPT.ag1.at -> |~| ag2:diff(ME,{ag1}) @
    realocaAtividade.ag1!at.ag2->
      desalocaAtividade.ag1!at->alocaAtividade.ag2!at->
      Gerente(e,t))
  []
  t == {} & fimJogo -> SKIP
```

onde $t = \{at_1, at_2, at_3, at_4, ..., at_n\}$ e $ME = \{0, 1, 2, 3\}$ são o conjunto de tarefas e os índices dos Membros de Equipe, respectivamente.

Os *traces* do processo Gerente são os seguintes:

$Traces(Gerente) =$
$\{\langle\rangle,$
$\langle\tau\rangle,$
$\langle\tau, \tau\rangle,$
$\langle\tau, \tau, alocaAtividade\rangle,$
$\langle\tau, \tau, alocaAtividade, negaAtividade\rangle,$
$\langle\tau, \tau, alocaAtividade, negaAtividade, \tau,$
$realocaAtividade, \rangle,$
$\langle\tau, \tau, alocaAtividade, negaAtividade, \tau,$
$realocaAtividade, \tau, desalocaAtividade\rangle,$
$\langle\tau, \tau, alocaAtividade, negaAtividade, \tau,$
$realocaAtividade, \tau, desalocaAtividade, alocaAtividade\rangle,$
$\ldots,$
$\langle reuniaoAcompanhamento\rangle,$
$\langle reuniaoAcompanhamento, emReuniao\rangle,$
$\langle reuniaoAcompanhamento, fimReuniao\rangle,$
$\langle reuniaoAcompanhamento, fimReuniao,$
$alocaAtividade\rangle,$
$\ldots, \}$

Para traduzir os *traces* do papel Gerente para comportamentos PopOrg correspondentes, é necessário substituir cada evento $\tau$ pela ação correspondente (e.g. o primeiro $\tau$ refere-se à escolha interna $\sqcap\, ag2 : diff(ME, ag1)$, que corresponde à escolha de um agente, então é substituído por escolheAgente).

$Bh_G = \{$
$[t_0 \mapsto escolheAgente, t_1 \mapsto escolheTarefa, t_2 \mapsto alocaAtividade, ...],$
$...,$
$[t_0 \mapsto solicitaReuniaoAcompanhamento, t_1 \mapsto emReuniao, ...],$
$[t_0 \mapsto solicitaReuniaoAcompanhamento, t_1 \mapsto fimReuniaoA, ...],$
$...$
$\}$

Este conjunto de comportamentos coincide com o resultado da função bhs:

$bhs(Gerente) = \{$
$[0 \mapsto \{\tau\}, 1 \mapsto \{\tau\}, 2 \mapsto \{alocaAtividade\}, ...],$
$...,$

$[0 \mapsto \{solicitaReuniaoAcompanhamento\}, 1 \mapsto \{emReuniao\}, ...],$
$[0 \mapsto \{solicitaReuniaoAcompanhamento\}, 1 \mapsto \{fimReuniaoA\}, ...],$
...
$\}$

O trecho em CSP a seguir mostra uma parte da especificação dos Membros de Equipe:

```
MembEquipe(i,estado,l) =
      (#l <=3) & alocaAtividade.i.at ->
        MembEquipe(i,estado,l^<at>)
    []
      (l==<>) & fimJogo->SKIP
    []
      (estado == I) and (l!= <>) &
        (
          executaAtividade!head(l) ->
            MembEquipe(i,W,l)
        []
          negaAtividade.i!head(l) ->
            desalocaAtividade.i.at ->
              desaloca(i,estado,l,at,<>)
        []
          desalocaAtividade.i.at ->
            desaloca(i,estado,l,at,<>)
        []
          reuniaoAcompanhamento.n ->
            MembEquipe(i,RA,l)
        )
    []
      (estado == W) &
        (
         (executandoAtividade -> MembEquipe(i,W,l))
        []
          fimAtividade -> MembEquipe(i,I,tail(l))
        []
          reuniaoAcompanhamento.n ->
            MembEquipe(i,RA,l)
          []
          dispersao -> MembEquipe(i,I,l)
        []
          problemaT -> MembEquipe(i,PT,l)
        )
    []
      (estado == RA) &
        (
          emReuniao -> MembEquipe(i,RA,l)
        []
          fimReuniaoA -> MembEquipe(i,I,l)
        )
```

Após definidos os conjuntos de comportamentos, podemos definir o conjunto de papéis do sistema, representado no PopOrg como:

$R_\omega = \{Gerente, MembEquipe\}$

Utilizamos a linguagem CSP$_M$ e o verificador de modelos FDR2 para verificar propriedades como *deadlock*, *livelock* e não-determinismo, bem como propriedades de segurança *(safety properties)* que podem ser verificadas através de refinamentos na semântica de traces.

Alguns exemplos de propriedades de segurança são:

i. VTEAM $\sqsubseteq_{\mathrm{T}}$ pedeAjuda.1.0 ->STOP

ii. VTEAM $\setminus$ diff(Events,{|reuniaoAcompanhamento|}) $\sqsubseteq_{\mathrm{T}}$ reuniaoAcompanhamento.n -> STOP

iii. VTEAM $\sqsubseteq_{\mathrm{T}}$ alocaAtividade.1.a1 -> STOP

O primeiro exemplo verifica se um pedido de ajuda pode ocorrer no início do jogo. Obviamente esta propriedade é falsa, pois de acordo com a especificação, um pedido de ajuda só pode ocorrer após a identificação de um problema técnico. A segunda propriedade é verdadeira, pois verifica se uma reunião pode ser solicitada a qualquer momento. Por fim, a terceira verificação analisa a possibilidade de alocação de atividades no início do jogo, o que também é verdadeiro.

A verificação de propriedades deste tipo auxiliam o projetista a garantir a correção no projeto de organizações de SMAs, a partir dos requisitos apresentados, evitando a descoberta posterior de falhas.

Para a representação de processos de troca utilizando a ferramenta FDR2, encontramos algumas limitações, pois ela não permite que o alfabeto de eventos seja definido em termos de operações de conjuntos das partes de um alfabeto básico finito, embora isto seja formalmente possível na estrutura de CSP. Além disto, FDR2 não permite a definição de novas relações entre símbolos do alfabeto, tornando impossível verificar as restrições de compatibilidade exigidas no modelo PopOrg.

A escolha do método formal RAISE e sua linguagem RSL surgiu como alternativa para estas limitações.

## 6 O método RAISE e a linguagem RSL

RAISE (*Rigorous Approach to Industrial Software Engineering*) foi inicialmente desenvolvido com o objetivo de fornecer uma melhoria em relação a outros métodos como VDM, Z, CSP, CCS, Larch e OBJ. O método formal RAISE consiste no método de desenvolvimento RAISE [14] e a linguagem de especificação RSL (*Raise Specification Language*) [13].

RSL oferece uma notação rica, baseada em matemática, na qual requisitos, especificações e etapas do projeto de software podem ser formulados e verificados. RSL é uma linguagem de amplo espectro, podendo ser utilizada para expressar tanto especificações abstratas, de alto nível, quanto projetos concretos, de baixo nível.

### 6.1 Formalização da Estrutura PopOrg em RSL

A linguagem RSL foi utilizada no modo aplicativo [14] para representar a estrutura do modelo PopOrg. Inicialmente foram definidos os módulos necessários para a especificação dos tipos utilizados nesta representação (cf. Fig.1).

O primeiro módulo, POP_TYPE, define os tipos de dados e funções RSL correspondentes à estrutura populacional do modelo PopOrg.

Na definição **type** são assumidos dois tipos abstratos: *Agent* and *Action*. O tipo *T* é definido como o tipo pré-definido **Nat**, e dois novos tipos são adicionados: *Behavior* and *ExchProc*. Os componentes individualizados da estrutura populacional são definidos como valores.

Axiomas, em RSL, expressam propriedades dos valores definidos e podem ser identificados através de nomes entre colchetes ([ ]) para fins de documentação. O axioma $[bc]$, por exemplo, declara que a função $bc$ é definida apenas para o conjunto de agentes que estão em $Ag$ (conforme definido na pré-condição) e retorna um elemento $b$ que satisfaz a pós-condição, ou seja, faz parte do conjunto $Bh$. O axioma [exchange_capability] define restrições para uma capacidade de troca.

POP_TYPE =
**class**
**type**
$Agent$,
$Action$,
$T =$ **Nat**,
$Behavior = T \xrightarrow{m} Action$**-set**
$ExchProc = T \xrightarrow{m} Action$**-set** $\times$ $Action$**-set**,
**value**
$Ag : Agent$**-set**,
$Act : Action$**-set**,
$Bh : Behavior$**-set**,
$Ep : ExchProc$**-set**,
$bc : Agent \xrightarrow{m} Behavior$**-set**
$ec : Agent \times Agent \xrightarrow{m} ExchProc$**-set**,

$prj_1 : ExchProc \times T \rightarrow Action$-**set**
$prj_1(e,t) \equiv first(e(t))$,

$prj_2 : ExchProc \times T \rightarrow Action$-**set**
$prj_2(e,t) \equiv second(e(t))$,

$first : (Action$-**set** $\times\ Action$-**set**$) \rightarrow Action$-**set**
$first(x,y) \equiv x$,

$second : (Action$-**set** $\times\ Action$-**set**$) \rightarrow Action$-**set**
$second(x,y) \equiv y$,

$empty : Behavior$-**set** $= \{\}$,

$actTime : Behavior$-**set** $\times\ Action$-**set** $\times\ T \rightarrow Action$-**set**
$actTime(b,x,t) \equiv$,
**case** $b$ **of**
$empty \rightarrow x$,
$\_ \rightarrow actTime(b \backslash \{\textbf{hd}(b)\}, (\textbf{hd}(b))(t)$ **union** $x,t)$
**end**

**axiom**
$[bc]$
$\forall\, a : Agent \bullet$
$bc(a)$ **as** $b$ **post** $b \subseteq Bh$
**pre** $a \in Ag$,
$[ec]$
$\forall\, a1, a2 : Agent \bullet$
$ec(a1,a2)$ **as** $e$ **post** $e \subseteq Ep$
**pre** $a1 \in Ag \wedge a2 \in Ag$,
$[prj1]$
$\forall\, e : ExchProc, t : T \bullet$
$prj1(e,t)$ **as** $a$ **post** $a \subseteq Act$
**pre** $e \in Ep$,
$[prj2]$
$\forall\, e : ExchProc, t : T \bullet$
$prj2(e,t)$ **as** $a$ **post** $a \subseteq Act$
**pre** $e \in Ep$,
$[exchange\_capability]$
$\forall\, a1, a2 : Agent, e : ExchProc, t : T \bullet$

$prj1(e,t) \subseteq unionbh(t,a1) \wedge prj2(e,t) \subseteq unionbh(t,a2)$
**pre** $a1 \in Ag \wedge a2 \in Ag \wedge e \in ec(a1,a2)$

**end**

O módulo MICROORG_TYPE foi criado para definir a estrutura micro-organizacional do modelo PopOrg. Neste módulo é feita uma extensão do POP_TYPE pois alguns tipos definidos anteriormente são novamente usados. Os tipos *Role* e *Link* são adicionados e alguns valores pertencentes à estrutura micro-organizacional são definidos. Os valores $Bh_\omega$ e $Ep_\omega$ são introduzidos para representar o conjunto de comportamentos e o conjunto de processos de troca (respectivamente), que podem ser realizados pelos papéis. O axioma [microlink_capability] garante que cada micro-ligação possível entre dois papéis esteja na capacidade de micro-ligação dos papéis envolvidos e que cada micro-ligação existente entre dois papéis esteja no conjunto das micro-ligações possíveis.

MICROORG_TYPE =
**extend** POP_TYPE **with**
**class**
**type**
$Role = Behavior$-**set**,
$Link = Role \times Role \times ExchProc$,
**value**
$Bh_\omega : Behavior$-**set**,
$Ep_\omega : ExchProc$-**set**,
$R_\omega : Role$-**set**,
$L_\omega : Link$-**set**,
$ec_\omega : Role \times Role \to ExchProc$-**set**,
$lc_\omega : Role \times Role \to Link$-**set**,
**axiom**
$[lc_\omega]$
$\forall\, r1, r2 : Role \bullet$
$lc_\omega(r1,r2)$ **as** $l$ **post** $l \subseteq L_\omega$
**pre** $r1 \in R_\omega \wedge r2 \in R_\omega$,
$[microlink\_capability]$
$\forall\, l : Link \bullet \exists\, r_1, r_2 : Role \bullet$
$(l \in L_\omega \wedge (r_1 \in R_\omega \wedge r_2 \in R_\omega)) \Rightarrow l \in lc_\omega(r_1, r_2)$
**end**

A escolha pela divisão em módulos permite o reuso de informações já especificadas e a inclusão de novos elementos, específicos em um módulo, sem afetar os demais.

Finalmente, o módulo MACROORG_TYPE apresenta a definição dos tipos e valores correspondentes à estrutura macro-organizacional. Grupos de papéis são conjuntos de papéis e suas micro-ligações. Grupos de papéis são capazes de interagir entre si, através de macro-ligações. O axioma que as macro-ligações devem satisfazer é análogo ao axioma que deve ser satisfeito pelas micro-ligações.

MACROORG_TYPE =
**extend** MICROORG_TYPE **with**
**class**
**type**
$Group = Role$-**set** $\times$ $Link$-**set**
$MacroLink = Group \times Group \times Link$-**set**
**value**
$G_\Omega : Group$-**set**,
$L_\Omega : MacroLink$-**set**,
$lc_\Omega : Group \times Group \to MacroLink$-**set**,
**axiom**
$[macrolink\_capability]$
$\forall\, l : MacroLink \bullet \exists\, g_1, g_2 : Group \bullet$
$(l \in L_\Omega \wedge (g_1 \in G_\Omega \wedge g_2 \in G_\Omega)) \Rightarrow l \in lc_\Omega(g_1, g_2)$
**end**

Para a verificação de tipos dos módulos definidos foi utilizada a ferramenta *rsltc (RSL Type Checker)* [22]. Esta especificação em RSL pode ser utilizada como base para a especificação do sistema VTEAM, pois ele pode ser visto como uma instância do PopOrg definido em RSL.

## 7 Conclusão

Neste artigo foi apresentado o uso das linguagens formais CSP e RSL, para a especificação de alguns aspectos organizacionais de um sistema multiagente.

O artigo apresentou a adequabilidade de CSP como um formalismo para a especificação de aspectos operacionais do nível micro-organizacional de modelos PopOrg (i.e., da especificação formal de comportamentos de papéis organizacionais e processos de trocas entre estes papéis). Esta viabilidade foi mostrada com a ajuda da função bhs e a relação de refinamento entre os modelos semânticos de CSP, que estabelecem a herança de propriedades válidas entre eles.

Papéis e processos de troca de um sistema multiagente exemplo foram especificados

em CSP e algumas propriedades da especificação (*deadlock*, *livelock* e não-determinismo) foram verificadas utilizando-se o verificador de modelos FDR2. Através da relação de refinamento elas mostraram-se válidas também para a implementação PopOrg daquela especificação.

Os *traces* que o FDR2 produziu para os papéis foram traduzidos para mostrar a forma que eles assumem na implementação PopOrg da especificação.

Na utilização da ferramenta FDR2 para a representação de processos de troca, foram encontradas algumas limitações, pois ela não permite que o alfabeto de eventos seja definido em termos de operações de conjuntos das partes de um alfabeto básico finito, embora seja formalmente possível em CSP. Além disto, FDR2 não permite a definição de novas relações entre símbolos do alfabeto, tornando impossível verificar as restrições de compatibilidade exigidas no modelo PopOrg.

Finalmente, o método formal RAISE e sua linguagem RSL foram apresentados como alternativa para estas limitações. Inicialmente eles foram utilizados para a representação estrutural do modelo PopOrg e atualmente estão sendo verificados quanto à adequabilidade para a demonstração de propriedades dos sistemas.

## Referências

[1] D. Bjørner. *Software Engineering I*. Springer Science, 2007.

[2] G. Caire, W.Coulier, F. Garijo, J. Gomez, J. Pavon, F. Leal, P. Chainho, P. Kearney, J. Stark, R. Evans, and P. Massonet. Agent oriented analysis using message/uml. In M. J. Wooldridge, G. Weiss, and P. Ciancarini, editors, *Agent-Oriented Software Engineering II*, volume 2222 of *Lecture Notes in Computer Science*, pages 119–135, Berlin Heidelberg, 2001. Springer Verlag.

[3] A. C. R. Costa and G. P. Dimuro. Introducing social groups and group exchanges in the poporg model. In *AAMAS '09: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems*, pages 1297–1298, Richland, SC, 2009. International Foundation for Autonomous Agents and Multiagent Systems.

[4] A. C. R. Costa and G. P.Dimuro. A minimal dynamical mas organization model. In V. Dignum, editor, *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models*, pages 419–445, Hershey, 2009. IGI Global.

[5] J. Davies and S. Schneider. A brief history of timed csp. *Theoretical Computer Science*, 138(2):243–271, 1995.

[6] S. A. Delloach. Engineering organization-based multiagent system. In A. Garcia et al., editor, *International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS'05)*, volume 3914 of *Lecture Notes in Computer Science*, pages 109–125, St. Louis, MO, 2006. Berlin, Springer.

[7] Y. Demazeau and A. C. R. Costa. Populations and organizations in open multi-agent systems. In *Proceedings of the 1st. National Symposium on Parallel and Distributed AI (PDAI'96)*, Hyderabad, India, 1996.

[8] V. Dignum. *A model for organizational interaction: based on agents, founded in logic*. PhD thesis, Utrecht University, Utrecht, 2004.

[9] M. Esteva, D. Cruz, and C. Sierra. Islander: an electronic institutions editor. In *Proceedings of the First International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2002)*, pages 1045–1052, Bologna, 2002.

[10] J. Ferber, O. Gutknecht, and F. Michel. From agents to organizations: an organizational view of multi-agent systems. In P. Giorgini, J. Muller, and J. Odell, editors, *Agent-Oriented Software Engineering VI*, volume 2935 of *Lecture Notes in Computer Science*, pages 214–230. Springer Verlag, 2004.

[11] Formal Systems (Europe) FSE. Failures-divergence refinement: Fdr2 user manual, 2005.

[12] P. Giorgini, M. Kolp, J. Mylopoulos, and M. Pistore. The tropos methodology: An overview. In M.-P. Gleizes Bergenti and F. Zambonelli, editors, *The Tropos Methodology: An Overview*, Methodologies And Software Engineering For Agent Systems, page 505. Kluwer Academic Press, New York, 2003.

[13] The RAISE Language Group. *The RAISE Specification Language*. TERMA A/S, Denmark, 1992.

[14] The RAISE Method Group. *The RAISE Development Method*. TERMA A/S, Denmark, 1995.

[15] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, New York, 1985.

[16] J. F. Hübner and J. S. Sichman. Organização de sistemas multiagentes. In F. Osório R. Vieira and S. Rezende, editors, *III Jornada de Mini-Cursos de Inteligência Artificial (JAIA'03)*, volume 8 of *III Jornada de Mini-Cursos de Inteligência Artificial (JAIA'03)*, pages 247–296, Campinas, Brasil, 2003. Campinas, SBC.

[17] C. A. Iglesias, M. Garijo, J. C. González, and J. R. Velasco. Analysis and design of multiagent systems using mas-commonkads. In *Intelligent Agents IV Agent Theories,*

*Architectures, and Languages*, volume 1365 of *Lecture Notes in Computer Science*, pages 313–327. Berlin, Springer, 1998.

[18] N. Jennings and M. Wooldridge. Agent-oriented software engineering. In J. Bradshaw, editor, *Handbook of Agent Technology*. AAAI/MIT Press, 2000.

[19] N. R. Jennings. An agent-based approach for building complex software systems. *Communications ACM*, 44(4):35–41, 2001.

[20] A. C. R.Costa and G. P. Dimuro. Semantical concepts for a formal structural dynamics of situated multiagent systems. In *Proceedings of COIN@Durham*, pages 41–52, Durham, UK, 2007. Durham, University of Durham.

[21] A. W. Roscoe. *The theory and practice of concurrency*. Prentice Hall, Englewood Cliffs NJ, 1998.

[22] UNU/IIST. Iii/3/1 raise tools, 2008.

[23] VTEAM. Projeto vteam [online], 2009. Disponível em: <http://vteam.cin.ufpe.br>. Acesso em: setembro 2009.

[24] M. Winikoff and L. Padgham. The prometheus methodology. In F. Bergenti, M-P. Gleizes, and F. Zambonelli, editors, *Methodologies and Software Engineering for Agent Systems. The Agent-Oriented Software Engineering handbook*, volume 11 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, pages 217–234. Kluwer Publishing, 2004.

[25] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10:115–152, 1995.

[26] F. Zambonelli, N. Jennings, and M. Wooldridge. Developing multiagent systems: The gaia methodology. *ACM Transactions on Software Engineering and Methodology*, 12(3):317–370, 2003.